DEVELOPMENT OF THE COMPONENTS OF A LOW COST, DISTRIBUTED FACIAL VIRTUAL CONFERENCING SYSTEM

A thesis submitted in fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

of

RHODES UNIVERSITY

by

SOTERIOS PANAGOU

January 2000

Abstract

This thesis investigates the development of a low cost, component based facial virtual conferencing system.

The design is decomposed into an encoding phase and a decoding phase, which communicate with each other via a network connection. The encoding phase is composed of three components: model acquisition (which handles avatar generation), pose estimation and expression analysis. Audio is not considered part of the encoding and decoding process, and as such is not evaluated.

The model acquisition component is implemented using a visual hull reconstruction algorithm that is able to reconstruct real-world objects using only sets of images of the object as input. The object to be reconstructed is assumed to lie in a bounding volume of voxels. The reconstruction process involves the following stages:

- Space carving for basic shape extraction;
- Isosurface extraction to remove voxels not part of the surface of the reconstruction;
- Mesh connection to generate a closed, connected polyhedral mesh;
- Texture generation. Texturing is achieved by Gouraud shading the reconstruction with a vertex colour map;
- Mesh decimation to simplify the object.

The original algorithm has complexity *O(n)*, but suffers from an inability to reconstruct concave surfaces that do not form part of the visual hull of the object. A novel extension to this algorithm based on Normalised Cross Correlation (NCC) is proposed to overcome this problem. An extension to speed up traditional NCC evaluations is proposed which reduces the NCC search space from a 2D search problem down to a single evaluation.

Pose estimation and expression analysis are performed by tracking six fiducial points on the face of a subject. A tracking algorithm is developed that uses Normalised Cross Correlation to facilitate robust tracking that is invariant to changing lighting conditions, rotations and scaling. Pose estimation involves the recovery of the head position and orientation through the tracking of the triangle formed by the subject's eyebrows and nose tip. A rule-based evaluation of points that are tracked around the subject's mouth forms the basis of the expression analysis. A user assisted feedback loop and caching mechanism is used to overcome tracking errors due to fast motion or occlusions. The NCC tracker is shown to achieve a tracking performance of 10 fps when tracking the six fiducial points.

The decoding phase is divided into 3 tasks, namely: avatar movement, expression generation and expression management. Avatar movement is implemented using the base VR system. Expression generation is facilitated using a Vertex Interpolation Deformation method. A weighting system is proposed for expression management. Its function is to gradually transform from one expression to the next. The use of the vertex interpolation method allows real-time deformations of the avatar representation, achieving 16 fps when applied to a model consisting of 7500 vertices.

An Expression Parameter Lookup Table (EPLT) facilitates an independent mapping between the two phases. It defines a list of generic expressions that are known to the system and associates an Expression ID with each one. For each generic expression, it relates the expression analysis rules for any subject with the expression generation parameters for any avatar model. The result is that facial expression replication between any subject and avatar combination can be performed by transferring only the Expression ID from the encoder application to the decoder application.

The ideas developed in the thesis are demonstrated in an implementation using the CoRgi Virtual Reality system. It is shown that the virtual-conferencing application based on this design requires only a bandwidth of 2 Kbps.

Keywords: Three Dimensional Graphics and Realism, Reconstruction, Scene Analysis: Tracking, Shape, Time-Varying Imagery, Videoconferencing, Coding and Information Theory.

Computing Review Categories: I.3.7, I.4.5, I.4.8, H.1.1, H.4.3

Acknowledgements

This account represents two years of hard work. I owe much to my mentor and supervisor Professor Shaun Bangay. He has guided me in ways I cannot describe to arrive at this point. His door is always open and he is always willing to listen and help out in any way he can. More than once he has gone out of his way to assist me in the paths of research and point me in the right direction. Those countless supervisor meetings that were spent ironing out inconsistencies and finding the simplest solutions for problems that, at the time, seemed insurmountable, have all paid off.

A special thanks to Keith, Fred and Clint, who very willingly listened to all my world-conquering theories. Additionally I would like to thank Alfredo Terzoli for encouraging me in ways I cannot describe. Thanks also to the creators and maintainers of that wonderful tool, Coco/R, Prof. Terry and Mosenbock.

I have to thank Austin, Professors Terry and Shaun for the rigorous proofreading and advice on the restructuring that led to the final layout of the thesis.

To Yoda, Psychotic Espresso, Barbarian, X, Gryffon, Griffon, Lord Goran, Guanobomber, Daemon and Odinga, and everyone else, thanks for the countless hours of Starcraft and Quake that made the time pass by so quickly.

To the rest of the Computer Science Department, thanks for the wonderful atmosphere that is fostered. It has been such a pleasure to be part of this institution.

Finally, thanks have to go to my family for understanding my needs and assisting where possible.

I acknowledge the financial support of the National Research Foundation of South Africa.

Contents

List of Figures

Chapter 1 - Introduction

1.1 Problem Statement

This dissertation investigates the use of a model-based coding system (overlaid on top of a VR system) to perform compression of facial information in a networked video conferencing system using standard desktop workstations. It answers the question: "Is it possible, using an avatar[1], to replicate a subject's head movements and expressions cheaply and in real time?"

Compression of a video stream containing a subject is performed by extracting avatar control data. An incoming video stream is analysed to give avatar geometry, head movements, and to identify the expression exhibited on the subject's face. The decompression mechanism involves the use of the avatar control data to animate the model's head movements and expressions. The constraint that the system be inexpensive implies that the system development occur using no proprietary hardware solutions.

The problem, as stated above, falls into a category known as model-based coding (Eisert *et al.* [15]). This field deals with:

- The generation of an avatar resembling the subject;
- The determination of his/her head pose;
- The classification of the expression depicted on his/her face; and
- The remote replication of the subject's expression via the avatar.

The fields of research that fall into the realm of model-based coding include:

- Model acquisition for the purposes of avatar generation;
- Subject tracking for the purposes of head pose determination and expression classification; and
- Model deformations for expression generation.

The proliferation and acceptance of this approach for remote communication is highlighted by the incorporation of such an architecture into the recently ratified MPEG-4 standard (Koenen *et al.* [34]).

[1] The term "avatar" refers to a 3D representation of a person in a virtual environment.

1.2 Design Aims

The aim with the research detailed here is the development of a framework for a low cost, component-based virtual conferencing system. Additionally, an implementation to test the framework is envisaged, and optimisations to enhance this base implementation are discussed.

The final system must be:

- Low Cost: the technology must be practical using standard desktop workstations. As mentioned in the previous section, this also implies that no proprietary hardware solutions be employed in the final system. The use of electromagnetic trackers, for example, is thus prohibited;
- Component-Based: the final system is composed of individual components connected to one another in a specific order, each performing a specific task. This component design details the functionality required from each component, thereby abstracting its implementation. Another advantage of a component-based design is the issue of reusability. The technology thus becomes available to other systems and is not constrained to any specific implementation;
- Virtual: a Virtual Reality system is envisaged as a platform of implementation.

There is no mention of the audio aspect of the conferencing application. It is considered beyond the scope of this discussion, since the focus of this research is primarily on the conversion of the video stream into avatar control data.

The ultimate goal is the development of a low cost model-based coding system built on top of a VR system. The following goals form part of this vision, namely:

- The development of a system that is able to generate a realistic avatar of the subject. This avatar is meant to represent the subject. The primary focus is on the use of a low cost reconstruction technique to achieve the desired result;
- The development of a framework for the implementation of a model-based coding system that works in real time on standard desktop workstations. The framework is intended to tie together the pose estimation, expression classification, and expression generation tasks;
- Identification of appropriate technologies for implementing each component;
- Real time tracking of a subject’s head to firstly determine his/her head position and orientation, and then to classify the expression exhibited on his/her face;
- The replication of expressions overlaid on top of the avatar;
- A sample implementation of the framework developed as a part of a VR system.

1.3 Virtual Conferencing

This dissertation discusses the development of a model-based coding system using an existing Virtual Reality system as a platform of implementation. With this added layer, the system falls into the realm of Virtual Conferencing. The VR system, called CoRgi, can be classified as both distributed and component-based (Bangay *et al.* [3], Casanueva [9], Fourie [25], Killian [33], Preston [52]).

To classify CoRgi as only a VR system would be to deny a large part of its functionality; a more apt description is "a distributed component-based multimedia authoring system". In addition to acting in its traditional VR role, it also provides a collection of multimedia enhancements, such as the capture of video and audio from devices, and networking support to handle the transmission of video and audio. These facilities meet the needs of model-based coding.

Its use as an implementation platform for a coding system has three primary benefits, namely:

- There is a close coupling between the component structure of the coding system and CoRgi's component-based nature;
- Technology developed for the coding system is absorbed into the existing VR system and does not form part of an autonomous system. The VR system is enriched and enhanced in this way. Examples of additions to the system include deformation support and image-based tracking;
- It facilitates high-level development of the coding system. This arises because the VR system provides most of the low-level functionality (such as networking and graphics rendering) required.

1.4 Summary of Results

The major contribution of the research presented here is the development of a component-based framework for a complete virtual videoconferencing system, and an implementation thereof using a VR system, namely CoRgi. Specific achievements include:

- The development of a generic, yet complete model acquisition system for avatar generation. The system is based on the implementation of a Visual Hull reconstruction algorithm that is able to generate low cost reconstructions of real-world objects that are completely textured using only image of the real-world object as input. Texturing is

achieved through the use of a vertex colour map to Gouraud-shade the reconstruction. The object to be reconstructed is assumed to lie in a bounding volume of voxels. A reconstruction pipeline is applied to the bounding volume to generate the final reconstruction. It is composed of the following parts, namely: space carving (for basic shape extraction), isosurface extraction (to remove voxels not part of the surface of the reconstruction), mesh connection (to generate a closed, connected polyhedral mesh), texture generation and mesh decimation (to simplify the object). The algorithm has the advantage that the reconstruction process is completely automated. Additionally, it is shown that the pipeline has complexity *O(n)*.

- The development of an algorithm to handle concave surface reconstruction. This algorithm is based on an Normalised Cross Correlation (NCC) matching strategy;
- An NCC-based tracking system that works robustly and in real-time; the tracking system is able to achieve 10 frames per second when tracking six points on a subject's face. This facilitates monocular pose estimation and expression analysis;
- The development of the concept of an Expression Parameter Lookup Table (EPLT). It provides an independent mapping between the expression analysis phase and the process of expression generation phase. It contains a list of Expression IDs representing a collection of generic expression classifications. The subject-specific expression analysis rules are related to the model-specific expression generation rules using an Expression ID. During a conference, this translates to the transfer of a single Expression ID between encoder and decoder to facilitate expression replication;
- Development of a three-tier expression editing based on the notion that an expression is composed of a collection of indivisible facial movements, translating to a collection of deformations representing each expression. A vertex interpolation deformation system forms the basis of the expression editing system;
- A complete implementation of the framework using a distributed, component-based Virtual Reality system, namely CoRgi. The result is that each component has been implemented as a separate entity, thus allowing the different technologies to be merged into a more generic system. The implementation consists of two applications, namely: the encoder and the decoder. The encoder manages to achieve 10 fps, while the decoder's rendering performance is approximately 16 fps.
- A transmission rate of 2 Kbps, thereby facilitating communications with very low-bandwidth networking infrastructure. This represents a 1000-fold decrease of the

bandwidth required to transmit a 100x100 uncompressed video sequence at 10 frames per second.

1.5 Thesis Organisation

Chapter 2 (Design) provides a broad overview of the design of the virtual conferencing system that is the focus of this dissertation. The design is decomposed into an encoder and a decoder. The encoder is responsible for the generation of an avatar representation, pose estimation expression analysis. For each frame of the video stream, the control information is generated, and then transmitted across the network in place of the video. At the receiving end, the decoder applies the control information to the avatar, to mimic the user's movements (**avatar movement**) and expressions (**expression generation** and **expression management**). Furthermore, the concept of an EPLT is introduced to facilitate an independent mapping between expression analysis and expression generation.

Chapter 3 (Related Work) presents research related to each component in the design. Concepts common to more than one part of the system are initially highlighted. This is followed by an in-depth discussion of the theory and algorithms underlying each component. Additionally, a quantitative analysis of model acquisition approaches is performed to determine the most cost-effective 3D reconstruction solution.

Chapter 4 (Model Acquisition) focuses on the generation of low cost avatar representations of users. The discussion focuses on the development of a general and low cost reconstruction method that can be used for avatar generation. The final reconstruction system is low cost, because it requires only the use of a charge coupled device (CCD) based camera. The result is a fully textured, polyhedral representation of a real-world object. A novel method using Normalised Cross Correlation (NCC) is introduced to solve the problem of reconstructing surface concavities.

Chapter 5 (Encoding) discusses the application of Normalised Cross Correlation to solving the problem of robust image-based tracking in real-time. This tracking mechanism is then used to perform head-tracking and facial feature tracking. The results of the head tracking are used to perform head pose determination, while the tracked facial features are used to classify the expression exhibited by the user. Experimental results are provided for the performance of the NCC tracker. The discussion in this chapter forms the basis of the encoder, since the contents of this chapter underlie the core design of the transmitter or encoding application within the set framework.

Chapter 6 (Decoding) discusses a vertex-interpolation based deformation system that is used as a basis for the expression replication process. Expressions are classified as a collection of individual deformations that are performed in lockstep on the avatar. This facilitates the facial expression replication process. A three-tier expression editor is discussed that allows the parameters to be specified for each generic expression listed in the EPLT. The implementation of this application is based on the vertex-interpolation based deformation system.

Chapter 7 (Virtual Conferencing under CoRgi) discusses the implementation of the final system using the CoRgi architecture. The focus of this chapter is on how the CoRgi system has been designed and then how this design has facilitated the implementation of the final virtual conferencing system. Networking aspects are addressed in this chapter. The overall system performance is discussed.

Chapter 8 (Conclusion) presents concluding remarks and suggests directions for future research.

Chapter 2 - Design

2.1 Introduction

The design of a framework for a component-based virtual conferencing system is discussed in this chapter. A typical scenario is used to put the problem in context and define the requirements of the system. This is followed by a discussion of related work in virtual conferencing.

Following this, the design framework that underlies the virtual conferencing system is introduced. Each of the components is discussed in terms of the functionality it provides.

The chapter is concluded with a discussion on the requirements of the envisaged implementation. As will be seen, the CoRgi VR system meets all these requirements, and is thus well suited for the implementation of the system. Finally, a summary of the design goals is provided.

2.2 Typical Scenario

Every user requires a virtual presence that is intended to represent him/her in a conferencing situation. The primary intention is for the presence to mimic his movements and expression mannerisms. The subject sits down in front of the CCD camera. A number of snapshots of him are taken from different angles, and a 3D representation of his head is generated. This avatar can be used to represent him in the virtual conference. This process is handled by the Model Acquisition subsystem.

During the conferencing session, the user moves his/her head with respect to the camera. The facial expressions change as well. The head movement is tracked and the exhibited expression analysed. This task is the responsibility of the **encoder**. The encoder communicates the information it has deduced to the remote **decoder**. The decoder is responsible for moving the 3D representation of the person around according to the information gathered by the encoder. Additionally, the decoder animates the avatar expression.

2.3 Related Work

This section discusses existing approaches to constructing model-based coding systems. The section is partitioned firstly into a discussion on existing video coding approaches. This is then

followed by a discussion on an image–base video coding and eigentemplate neural network-based solutions.

2.3.1 Model-Based Video Coding

These approaches use a model to simulate user movement and expressions.

2.3.1.1 Face Cloning System

Escher *et al.* [19] discuss the development of a face cloning system. The system is aimed at "videoconferencing and tele-cooperative work at various sites with shared virtual environment inhabited by virtual clones". They provide a breakdown of the process into an **input stage**, an **analysis stage** and an **output stage**. The input stage provides a video stream of the person and an audio stream of what the subject says.

The analysis phase is further decomposed into three functional components, namely: face initialisation, video processing and audio processing. The face initialisation component takes as input two images of the user from the front and side, as well as a generic head polygonal mesh. The output of this component is a 3D textured facial model representing the subject. The video processing takes as input the live video feed, performs "real-time emotion tracking" to obtain geometric measures, generates Minimal Perceptible Actions (MPAs) using an **Emotion Compiler**, and combines all the determined MPAs into the MPA buffer. The Audio processing component is responsible for converting the live audio input stream into a collection of phonemes, which are then passed onto a **Phoneme Compiler** that generates a collection of MPAs specifying mouth movement based on the input audio.

The video and audio processing take place in parallel. The MPAs generated by each component are combined and synchronised using an MPA Synchroniser, which essentially synchronises the audio MPAs with the video MPAs. The result is an array of MPAs that determines how a Face Animator component deforms the 3D face model using a Dirichlet Free Form Deformation approach.

The output of the system is an animated face; the mouth is synchronised with the speech. Their final implementation is based around four SGI workstations because of the parallel nature of the analysis component. They claim to achieve 25 frames per second (fps) with the MPA processing and 40 fps with the deformation subsystem. An aspect they do not discuss is pose recovery of the subject.

2.3.1.2 MPEG-4 Compatible System

Eisert *et al.*[15] discuss the development of an MPEG-4 compatible head coding system, which is similar to the previous system discussed, but differs in one respect. They include the possibility of network communications in their classification of the structure of a typical model-based coding system. The decomposition of the problem is along the following lines:

A) A **model** component that extracts the "shape, texture, illumination, and dynamics" of a scene containing a head. A 3D laser scanner is used to obtain texture and depth information of a subject's head. Second-order triangular B-splines are used to define the face shape. They make use of a generic head consisting of 101 triangular B-spline patches. The control points attached to each B-spline are then manipulated according to the depth information obtained from the laser scanner. This deforms the generic head to assume the structure of the subject. To simplify the rendering process, deformed B-spline head is simulated using a mesh of triangular polygons. Use is made of the MPEG-4 FAPs (Face Animation Parameters) to parameterise all possible expressions that may arise (see the next chapter for a more in-depth discussion regarding the workings of the MPEG-4 standard). A table is employed for each model that described how the control points associated with each FAP will be translated and rotated to generate the appropriate expression;

B) An **analysis** component for the estimation of motion and facial expressions. They perform 3D motion estimation of the FAP points. Instead of evaluating the movement of a few core facial points, they "use the whole face image for the estimation" of the motion and use a feedback loop for error correction. Given an input image, estimation is performed to determine where the FAPs will be in the next image. In the next image, the estimated FAP positions are compared to their true locations. An optical flow mechanism is employed as a basis for the motion estimation process. This is based on the assumption that the inter-frame image coherency will be high. To overcome problems with this assumption when high degrees of motion are present, a hierarchical estimation process is used. This translates to using low-pass filtered and subsampled versions of the input images;

C) A **parameter coding** that encodes the analysed data;

D) A **parameter decoding** component to decode the analysed data. This component is also responsible for deforming the model;

E) A **synthesis** component responsible for the rendering of the head model.

They develop a mechanism to estimate FAPs using 2D image sequences. Furthermore, their use of optical flow methods in conjunction with 3D motion models facilitates the development of a robust and low complexity algorithm. They claim to achieve a transmission rate of approximately 1Kbps (as low as 0.6 Kbps).

2.3.1.3 Image-Based Coding

Ezzat *et al.* [23] discuss an image-based facial modelling method where expressions are simulated using a number of example images. Their system takes as input a number of pictures of a subject's head under different pose orientations and exhibiting different expressions. A neural network is then used to evaluate a sample image and extract pose estimation and expression from the image.

Use is made of a synthesis module that is provided as input to the analysis phase. A mask-based segmentation approach is used to perform segmentation of the head from the rest of the scene in the images. For each subject, an initialisation step is performed with the pose of the head recorded in nine different positions. This forms what they call a 3x3 network, as illustrated in **Figure 2.1**.

The generation of an image representing a user's pose and expression from the input database is decomposed into two parts, namely: warping and blending. The warping process makes use of

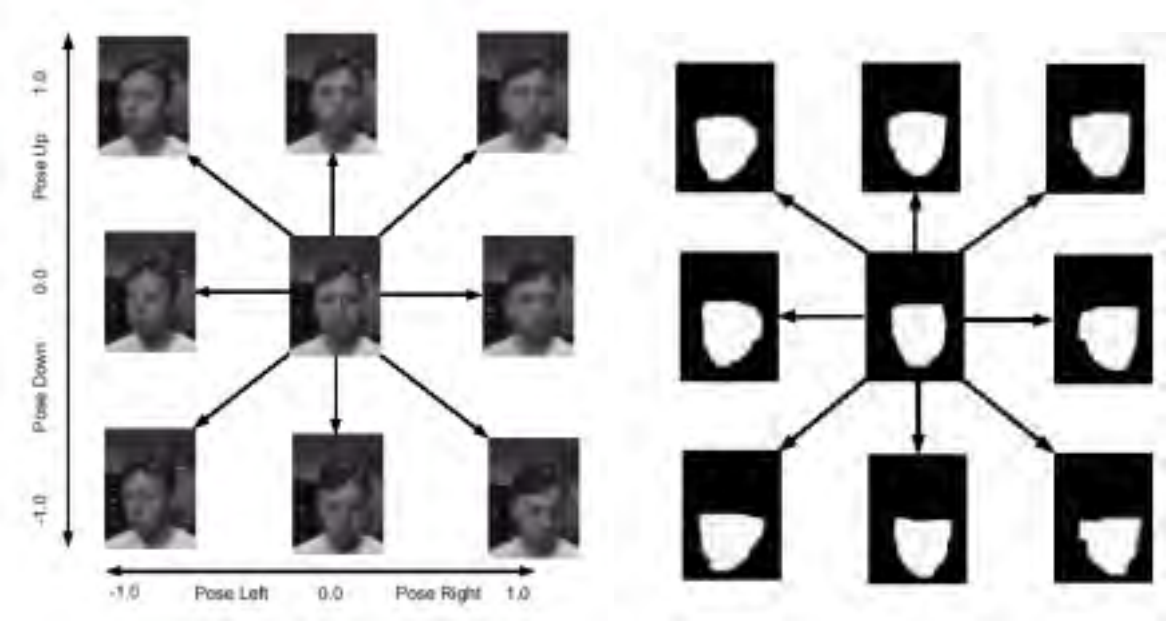


Figure 2.1 3x3 network illustrating all possible poses of a specific subject.

Images courtesy Ezzat *et al.* [23]

a correspondence vector to identify those parts of the images in the 3x3 network that correspond to one another. Each of the images is then altered at a pixel level using the correspondence vector. During the blending process, all of the warped images are combined together to generate the final image.

Although this method generates visually correct results, it is nonetheless computationally inefficient (Ezzat *et al.* [23] report that it takes up to half an hour to perform the process for a single frame).

2.3.1.4 Eigentemplates and Neural Networks

Moghaddam *et al.* [44] have developed a proof of concept system that works particularly well in video-telephony situations, whereby the head is located within the video stream. The head's position is located, its scale normalised and finally represented in terms of a "parametric image model obtained with a Karhunen-Loeve basis". They claim that this leads to a very compact representation of the specific face and high amounts of compression. Their system makes use of eigentemplates and neural networks to perform the coding. They have developed a parallel algorithm to improve system performance. Although they state that the system seems to handle head rotations up to 15^0, it is unlikely that this is adequate. The result of their system is that they are able to represent an arbitrary head with about 100 bytes of data.

2.4 System Design

This section builds on work discussed by Panagou *et al.* [48], with the system design being derived from that discussion. The system is decomposed into an encoder and a decoder, as illustrated in **Figure 2.2**. The components in bold are those that are fundamental to the design.

The encoder is responsible for determining the user pose and analysing his/her expression. The decoder's responsibility is to position the avatar representing the user according to the pose information, and replicate the expression identified. The expression replication is performed using geometric deformation. The concept of an Expression Parameter Lookup Table is introduced as a way of allowing the expression analysis and replication process to be independent. It contains a list of generic expressions. The encoder classifies the user's expression according to the list of expression in the EPLT. The decoder performs deformations based on the

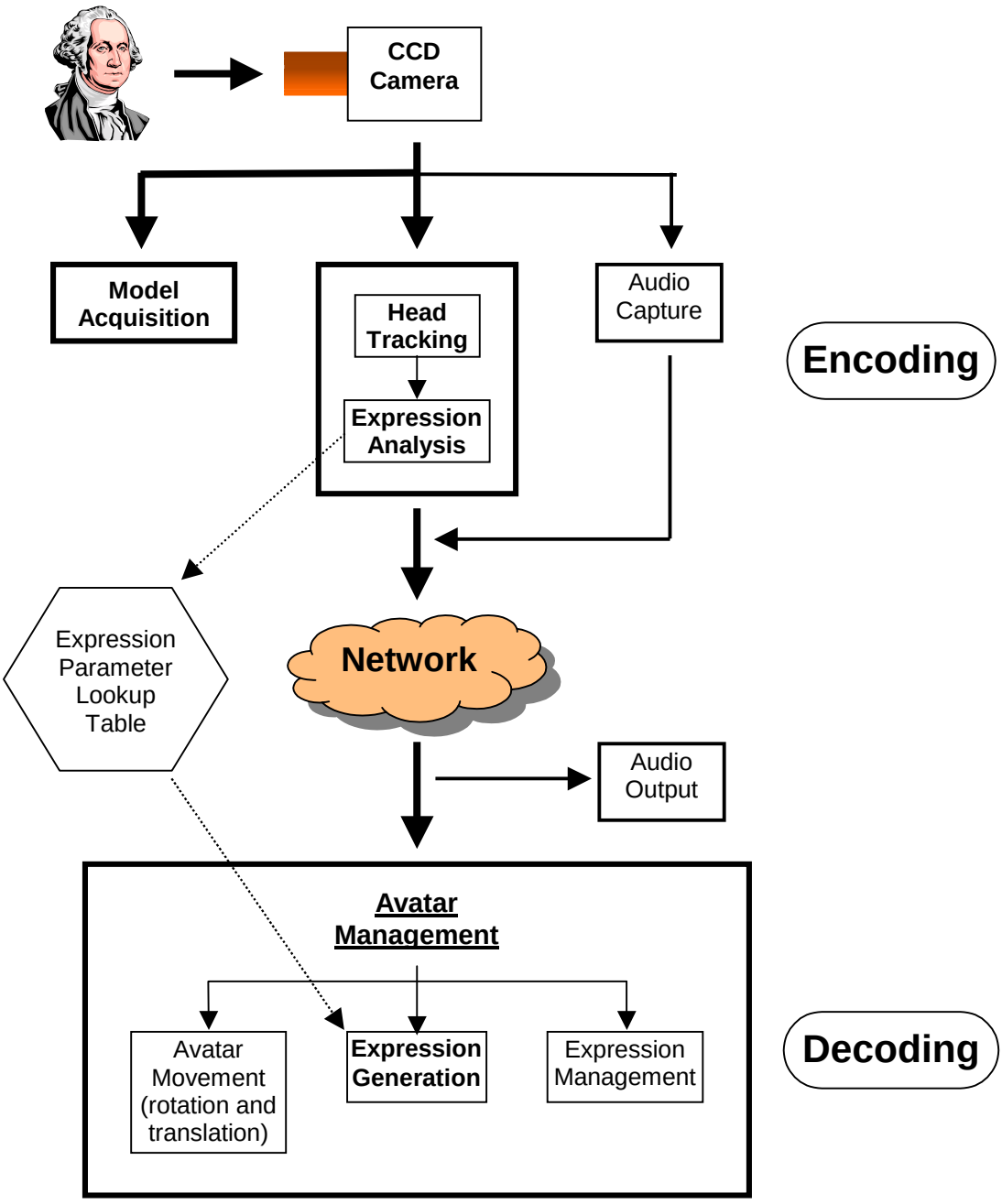


Figure 2.2 A conceptual design of the virtual conferencing framework.

expression that has been identified by the encoder. The use of the EPLT facilitates an independent mapping between expression analysis and expression generation

The encoding part is responsible for: **model acquisition**, audio capture, **head tracking** and **expression analysis**. The model acquisition component is not linked to any other component on the encoding side because it represents a process that occurs only once, when a new user is introduced to the system.

The decoding part of the system is responsible for the process of avatar management, that in turn is composed of three more specific tasks: controlling the movement of the avatar (such as rotation and translation of the head), expression generation and expression management.

The last major component that is responsible for communication between the encoding and decoding parts of the system is the networking component. This component defines the low-level structure of the network traffic that is passed from the encoder to the decoder.

Although audio transmission is important, it is considered beyond the scope of this discussion (for a discussion regarding the use of speech to enhance facial animation, refer to Pelachaud *et al.* [51] and Lewis [41]).

2.4.1 Model Acquisition

The role of the model acquisition component is to facilitate the generation of 3D avatar representations of users of the system. The aim here is to provide users with the ability to represent themselves in a virtual conference. This translates to performing reconstruction of the user resulting in the generation of a 3D model that represents the user. Reconstruction approaches vary from general approaches that are able to reconstruct any 3D object, to the reconstruction of heads via profile fitting techniques specifically. Unfortunately, the traditionally generic reconstruction approaches, which typically involve the use of 3D laser scanners, are very expensive.

2.4.2 Head Tracking

The head tracking component is responsible for pose determination of the subject. Common techniques include optical flow-based image-processing approaches such as the one presented below in the implementation, or the use of electro-magnetic trackers similar to the ones available from Polhemus Corporation. Trackers have the advantage of being invariant to occlusions and shadows that plague image-processing approaches, and also provide faster than real-time feedback; the Polhemus trackers typically return 3D coordinate position and orientation information at a rate of sixty updates a second. They are ideal for any real-time tracking requirements, including pose estimation. The downside of these electromagnetic trackers is their expense.

2.4.3 Expression Analysis

The expression analysis component deals with the process of facial expression classification. This can typically be performed using a MPA (Minimal Perceptible Action) or FAP (Facial Animation Parameter) based system, where each expression is defined in terms of a combination of lower level primitives (the definition of these terms MPA and FAP is provided in section 3.6.1). Given that an expression has been identified, a mapping function (pre-defined table) defines the way in which the expression identified will be overlaid over the avatar. This is handled by the EPLT (discussed below in section 2.4.6).

2.4.4 Networking

Having classified the subject's expression and determined his/her head position and orientation, the data must be packaged up for transmission across the network. This component deals with cross-platform networking issues at a low level. At a high level, it defines the structure of the network packet the system will employ.

2.4.5 Expression Generation

This component is used to generate the expression with the avatar. This translates to some deformation of the avatar representation to achieve the correct expression visually. This brings into question the whole concept of an expression. In this design, an expression is classified as a collection of indivisible deformations that are applied to the avatar topology. Taken together, the result of all the deformations is the generation of the expression defined by those deformations.

Figure 2.2 places the expression generation component as a subset of the **Avatar Management** component. The avatar management entry in the diagram is not an actual component, but more a conceptual grouping of all factors that affect an avatar. Included as part of this conceptual grouping is: **Avatar Movement**, referring to the task of moving an avatar, and **Expression Management**. Expression management comes into play when expression analysis occurs more rapidly than expression generation. A smile may be in the process of being generated and a request for a frown is received. The expression management component handles this task. Of these three subdivisions, namely avatar movement, expression management and expression generation, only expression generation is considered complex enough to be considered a true component within the framework.

2.4.6 Expression Parameter Lookup Table

The EPLT defines a collection of generic expressions (and associated Expression IDs) that the system understands. To facilitate the mapping process, a collection of subject-specific expression analysis rules/MPAs must be defined for each Expression ID. Additionally, each expression in the EPLT must have an associated collection of model-specific expression generation or deformation rules. It is assumed that each avatar that is used with the system has associated with it the necessary EPLT information, namely the expression analysis rules as well as the deformations rules for each expression.

The use of an EPLT has the advantage that it abstracts the avatar used from the subject being evaluated.

The EPLT thus contains the following information:

- A generic expression description;
- An Expression ID;
- Subject-specific expression analysis rules;
- Avatar-specific expression generation rules.

The result of the above lookup table is that any model can be used with any subject, provided that the expression analysis and generation rules are specified for each entry in the lookup table. This avoids the problem of constraining the subject to use a specific avatar. Furthermore, it allows expression replication to occur by transferring only an Expression ID to the decoder.

2.4.7 Expression Editing

The use of the EPLT assumes that there exists a collection of deformation parameters for each generic expression listed. An expression editing/generation mechanism is required to allow expressions to be specified in this way. This is the function of the expression editing system. Expressions can be constructed with the system by applying a whole host of different deformations and selecting the deformations that generate the most desirable results for a specific expression.

2.5 Implementation Requirements

The realisation of the design involves the creation of two applications, the one serving as a container for the encoder, and the other being the decoder. The two applications communicate with each other physically via the network and logically through the EPLT.

The virtual reality system envisaged for the final implementation of the design is required to have the following desirable characteristics:

- Component-based: it must support a component-based structure;
- Rendering engine: it must have an efficient rendering subsystem;
- Platform-independent networking: the networking infrastructure must be present to facilitate communication between the encoder and decoder;
- Video/Audio capture: there must be facilities to enable capture from CCD cameras.

Any VR system not offering these facilities, not only affects the final design, but also severely affects the functionality of the final implementation. The provision of the above functionality allows the implementation of the design to take place at a conceptually higher level, and at a much faster pace.

The system chosen, CoRgi, provides for all the above functionality, and has allowed the implementation of the design to take place at conceptually a higher level, and at a much faster pace.

The implementation of each component should be based on the cheapest technology possible. This will promote the use of desktop technology as the platform of implementation. The component-based nature of the design facilitates ease of extensibility and reuse.

2.6 Summary

This chapter discusses the component-based breakdown of a model-based coding system capable of performing virtual conferencing. A typical scenario is discussed, and is used to introduce the design.

The component-based system design is logically decomposed into a process of encoding and decoding. The encoding process involves the generation of an avatar representation (model acquisition), as well as head tracking /pose estimation and expression analysis. Decoding consists of avatar management, a logical grouping of three components, namely: avatar management, avatar movement and expression generation. Of these three, only expression generation is considered fundamental.

An expression is defined as a collection of individual deformations that, when applied to an avatar during the expression generation phase, results in an appropriate expression being generated.

The concept of an EPLT has been defined. It facilitates an independent mapping between the expression analysis and expression generation phases. It consists of a list of generic expressions (with associated Expression IDs), and subject-specific expression analysis rules and model-specific expression generation rules. This leads to an abstraction of the subject from the avatar used. The process of expression replication thus requires the transfer of a single Expression ID.

The requirements of the Virtual Reality system to be used as platform of implementation are identified and discussed as part of the final application design. The VR system chosen, namely CoRgi, meets all the required implementation needs. It supports and provides, amongst other functionalities: a component-based application development, an efficient rendering sub-system, a platform-independent networking infrastructure, and video capture from desktop CCD cameras.

Chapter 3 - Related Work

3.1 Introduction

The system design introduced in the previous chapter is composed of a number of components spanning the entire research spectrum. The purpose of this chapter is to put each component in perspective, by discussing related research pertaining to that component. Although the components in the system design fulfill very different roles, some of the components share core concepts that underlie their implementation. These core concepts are first discussed.

The remainder of this chapter is structured according to the system design. Each component is introduced. A qualitative assessment is performed to determine the approach most suited to the component as specified in the design. Additionally, the section discussing model acquisition includes a quantitative evaluation of the different approaches that fall under this category and the subsequent selection of an approach that is low cost, both computationally and economically.

3.2 Principles

This section introduces principal concepts that manifest themselves in more than one of the system components.

3.2.1 Normalised Cross Correlation

The term Normalised Cross Correlation can be described in terms of a correlation coefficient. According to Easton *et al.* [14], a correlation coefficient is a number between -1 and 1 that measures the degree to which two variables are linearly related. If there is perfect linear relationship with positive slope between the two variables, the correlation coefficient has a value of 1; if there is positive correlation, whenever one variable has a high (low) value, so does the other. If there is a perfect linear relationship with negative slope between the two variables, the correlation coefficient has a value of -1; if there is negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of zero means that there is no linear relationship between the variables.

“There are a number of different correlation coefficients that might be appropriate depending on the kinds of variables being studied”. (Easton *et al.* [14])

Rummel [56] describes correlation coefficients in terms of vectors for the purposes of deriving an explanation of Normalised Cross Correlation. Vectors pointing in the same direction have a positive correlation, while vectors pointing in opposite directions have a negative correlation. Each variable affecting the correlation coefficient is represented as a vector in space. As such, each variable has both a magnitude and direction. Given two variables with independent vector parameters, one way to correlate them is to normalise their lengths. This is done by dividing the vector's magnitude by its length, as follows:

X_j = vector j or variable j (1)

$\left|X_j\right|$ = length of vector j = $\sqrt{\sum X_{ij}^2}$ (2)

In other words, the length of a vector is the square root of the squared sum of its magnitudes. Then the normalisation of a vector is the division of its separate magnitudes by the vector's lengths. Furthermore, the direction is normalised as well. Given two variables, this is done by transforming the vectors to mean deviation vectors and computing the angle between the two vectors. This is done as follows:

$$\cos(\theta_{jk}) = \sum x_{ij}^* x_{ik}^* \Big/ \left|X_j^*\right|\left|X_k^*\right| \quad (3)$$

where X_j^* and X_k^* are vectors of mean deviationed data $\left(x_{ij} - \overline{X_j}\right)$ and $\left(x_{ik} - \overline{X_k}\right)$ respectively. The computation of θ gives a measure of correlation between the two mean deviationed vectors. The Normalised Cross Correlation between variables *j* and *k* is the result of combining equations (2) and (3), as follows:

$$\cos(\theta_{jk}) = \sum \left(x_{ij} - \overline{X_j}\right)\left(x_{ik} - \overline{X_k}\right) \Big/ \sqrt{\sum \left(x_{ij} - \overline{X_j}\right)^2 \sum \left(x_{ik} - \overline{X_k}\right)^2} \quad (4)$$

3.2.2 Background Removal Techniques

The techniques described in this section, namely chroma-keying and background elimination, attempt to isolate an object of interest in a video sequence by removing the parts of the image not representing the object.

3.2.2.1 Chroma-Keying

A pure blue or green screen is used and an actor is then captured in front of this screen. This technique is referred to as **Blue Screen Processing** and can be defined as “a process that allows for the combination of moving silhouettes and special effects to be combined” Unknown [65]. A direct extract from Unknown [65] on the implementation of blue screen methods is as follows: “Action is filmed before a brightly illuminated blue screen involving actors, objects (such as models), or clay figurines. That which has once been filmed is filmed two more times and then combined; once with a filter to remove the blue -- rendering a black background -- and once to filter out all colours except white resulting in dark silhouettes of the actors (or objects) with a white background. This second filtered result is applied over raw film (which has not been exposed) and then passed through a camera in the filming process of the "effects". This newly effected film now contains the photographic image of the effects with the performers' silhouettes. A special optical printer is then employed in the process to join the special effects with the initial pictures of the actors, objects or figures. Occasionally the light from the original blue screen spills over onto the image with the performers resulting in "blue spill" and creating an unwanted haze around the image”.

Digital solutions to chroma-keying involve the location of actors in a “uniformly lit blue room”, and then the use of either hardware or software chroma-keying solutions to “separate the image of the actor from the blue background” (Van der Bergh *et al.* [66]). Subsequent to this, the keyed image is be overlaid on top of another video stream; the application Van der Bergh *et al.* [66] discuss is the merging of the keyed video stream with a virtual environment to facilitate Immersive Telepresence. Unfortunately, hardware solutions tend to be expensive, and to that end, Van der Bergh *et al.* [66] introduce a software solution that is able to perform chroma-keying in real time.

3.2.2.2 Background Elimination

Chroma keying tends to be problematic in real-world situations; the requirement of a background of predetermined colour limits its usefulness and generality. Background elimination generalises this approach by allowing the background to be composed of any real-world scene, the only constraint being that the background be static.

The general approach is to take a number of calibrating snapshots containing only the static background. Subsequent to this, a template background image is obtained by averaging all the calibrating snapshots. This is done to minimise lighting side-effects (due to fluorescent light sources that constantly flicker etc.) and obtain a true averaged template.

Given a sample image at a later stage, the template is subtracted from the sample pixel by pixel; if the result of the subtraction falls within a certain threshold, the pixel is classified as background and eliminated. The result is a frame that contains only objects that are not part of the background scenery. Rekimoto [54] discusses such an approach.

3.2.2.3 Blob Growing

Oliver *et al.* [47] discuss the development of a system based on the use of a blob-growing algorithm. It is similar to background elimination, with the primary difference being that the elimination occurs by retaining objects of interest and eliminating those areas of the image that do not meet the search criteria. They define a blob as “a compact set of pixels that share a visual property that is not shared by the surrounding pixels”. They use blobs as an underlying representation to a tracking system that, they claim, allows them to accurately track objects even in complex environments in real time.

3.3 Model Acquisition

This section introduces general reconstruction approaches and provides a quantitative assessment of the approaches discussed to determine the method most suited for the purpose of low-cost avatar reconstructions. For completeness, techniques for modelling faces specifically are then discussed. The section is concluded with a brief overview of decimation as an approach to simplifying mesh reconstructions obtained from model acquisition systems.

3.3.1 General Reconstruction Techniques

In general, there are a number of approaches to object reconstruction. The degree to which post-processing techniques have to be applied to the results obtained from each method depends to a large degree on the complexity of the object being reconstructed. Post-processing here refers to the filling of holes in the reconstructed object.

3.3.1.1 Laser Range Finder Techniques

This approach involves the measurement of the phase difference of a laser pulse reflected off a surface being reconstructed (Roach [55]). The aim is the generation of a dense depth map representing the object, the depth depending on the measured difference. These measurements are taken for every point on the surface of the object being reconstructed. This technique is repeated until an acceptable number of points have been measured.

This technique has been successfully applied to the reconstruction of a cave for the movie Starship Troopers (Ashley [2]). The entire cave was scanned at a resolution of 2mm, up to a 50m range. The reconstruction of the entire cave took only 3 days, a task that would be very difficult and time-consuming with any other method. The major advantage this approach has over other approaches is the sheer scale of reconstructions that are possible. This system typically involves the large-scale reconstruction of buildings and other engineering structures very quickly. As such, it is a very specialised tool that is extremely expensive.

3.3.1.2 Light Illuminated Triangulation

This method is also classified as Laser Scanning. It is a very costly approach, typically requiring the purchase of dedicated hardware. A barcode laser source projects a line of laser light onto the surface of the object. The light intersects the surface and this intersection is recorded by a camera that is positioned directly above the laser light source. Techniques are then employed to connect the lines formed by the intersection.

A typical example of this method is the HLS handheld scanner from Polhemus InsideTrak[2]. In this case an electromagnetic tracking device capable of 6 degrees of freedom (DOF) orientation

[2] Information on this tool is available at Polhemus' web site: www.polhemus.com

Figure 3.1 Scanning results using the Polhemus InsideTrak HandHeld Scanner.

Source: www.polhemus.com

recovery in real-time is also connected to the camera. The entire setup is nicely packaged, with the barcode scanner/laser source, the tracker and the camera all part of a single unit.

The use of this tool results in a very accurate 3D reconstruction, as illustrated in **Figure 3.1** (the images have been obtained from the Polhemus web page at www.polhemus.com). Although these results are very impressive, the cost of this tool is prohibitively expensive: a quotation of approximately $17 000 was given for the tool and software. These types of tools can be classified as being specialised equipment and are designed to meet special needs. These two factors, namely the price and the specialised nature of the equipment, prevented the investigation of this technique.

Cyberware scanners work by shining low-intensity laser light on an object to create a lighted profile. A high-quality video sensor captures this profile from two viewpoints. Using an automated motion platform, the system can digitise thousands of these profiles in a few seconds to capture the shape of the entire object. Simultaneously, a second video sensor acquires colour information.

Another example of this type of system is the Minolta VIVID non-contact laser scanner system. This system is composed of a turntable attached to a rotating motor and a scanning/texture unit. The latter component consists of a camera and a laser unit. This unit is placed on a tripod and a picture of the object taken for later texturing. A laser stripe is swiped

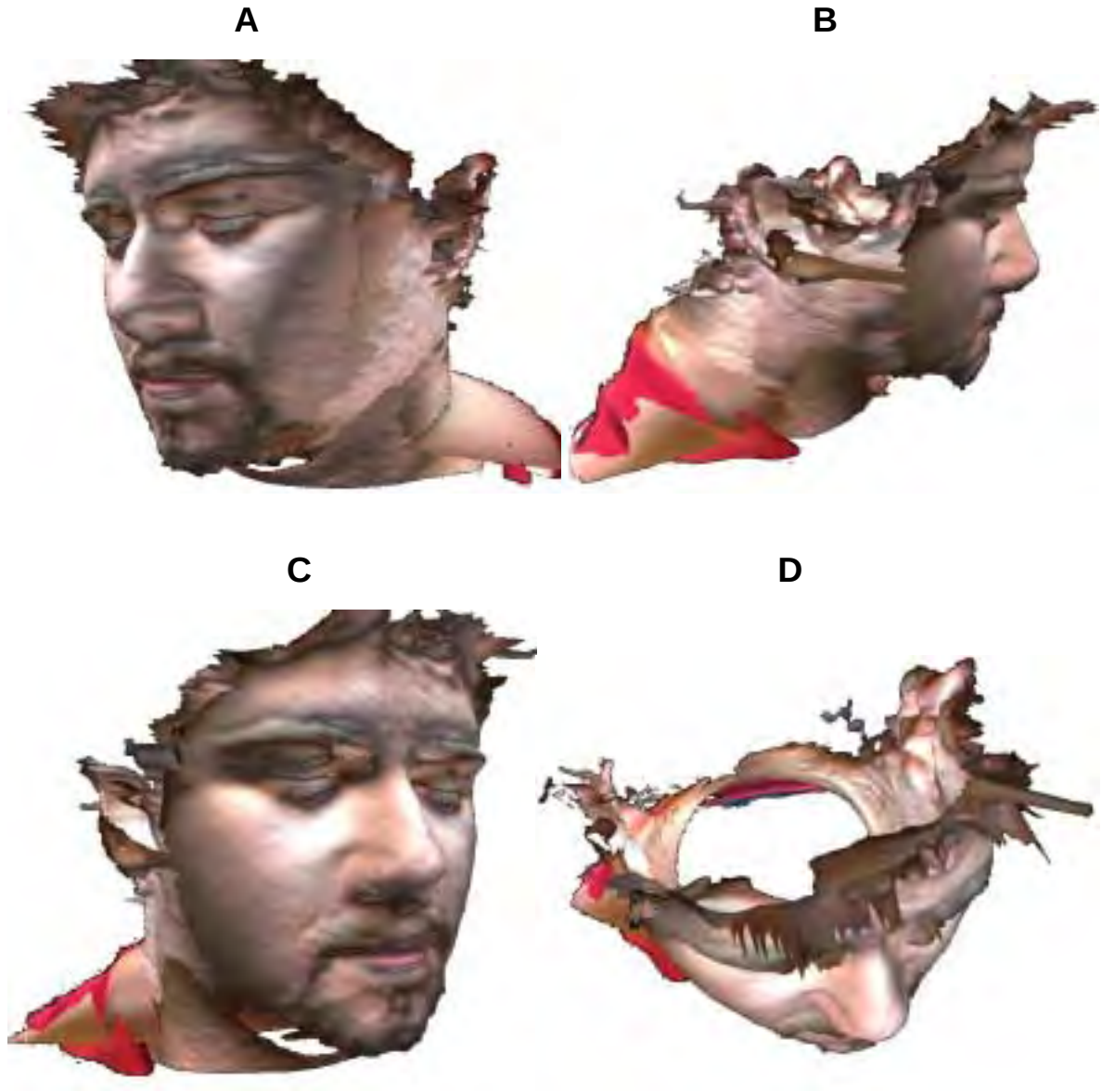


Figure 3.2 Reconstruction of Avatar Steri with the Minolta Vivid Scanner.

over the surface of the object, and the intersection of the stripe with the object during the swipe is used to determine the structure of the surface. The turntable is then rotated through a fixed angle and the process repeated. To determine the pose of the camera, a calibrating chart is scanned to conclude the scanning process. The pose of the camera can be recovered, and the scanned data corrected for the camera possibly scanning at a slant.

Once all the scans have been completed, the final reconstruction involves the stitching each of the scans together. The images of each of the scans are stitched together to obtain the final cylindrical texture map. **Figure 3.2** illustrates the reconstruction of Avatar Steri with the Minolta Vivid Scanner.

Using this system has revealed a number of aspects that affect the quality of the scanned results.

Unfortunately, because the laser stripe is red, it will be absorbed by a very black surface or a surface that diffracts the intersecting laser stripe, such as Avatar Steri's hair. As such, the results of the reconstruction are very much dependent on the surface colour, as well as the extent to which the surface diffracts the incoming laser. The lack of reconstruction of the subject's chin is attributed to the subject's beard occluding that part of the face from the scanning unit. Futhermore, additional post-processing is required to line up the textures with the scanned data.

The result is that the final reconstruction has many holes, particularly in places such as the subject's hair. The generation of a closed connected model from this requires much attention during post-processing.

The major advantage a setup like this has over the HLS scanner is that of texture extraction. Used in conjunction with a tool such as the HLS scanner, the problem of occlusions can be eliminated, while the texture generation is maintained.

3.3.1.3 Depth Estimation via Correlation

The problem with utilising a camera is that it inherently represents a perspective (3D) projection of a real-world scene onto a 2D plane, thus leading to loss of depth information. Most of the work of reconstruction algorithms is in trying to determine and recover this lost depth information. The previous methods are able to accurately determine the curvature/structure of the real-world object. They are able to determine the distance of the camera from every position on the object's surface reliably and accurately, but at a high economic cost. The method in this section makes use of NCC to perform reconstruction of an object using a stereoscopic pair of images of the object. Refer to Deriche *et al.* [12] for an in-depth discussion about the following

sections regarding the application of Normalised Cross Correlation, and more particularly, Epipolar Geometry, to depth recovery and thus surface reconstruction. Below is a summary of the concepts underlying this research.

3.3.1.3.1 Normalised Cross Correlation

The application of NCC to the problem of scene reconstruction is states as follows:

"Assuming a stereoscopic pair of images (a left and right image pair) of an object, depth information can be obtained if a pixel in the first image can be correctly matched to a corresponding pixel in the second. If a match is found, the Euclidean distance between the matched pair (often called the disparity of the pair) can be used as a representative depth value. Given (u_1,v_1) in the first image, find its match in the second image."

Given a pixel m_1 (u_1,v_1) in the left image, a match m_2 (u_2,v_2) can be found in the right image. Once a match is found, the depth of that pixel can be estimated by simple triangulation. The process of finding a match thus becomes important for the recovery of depth information.

To identify a matching pixel pair (m_1 and m_2) it is necessary to perform a 2D search in the second image for m_2. The search criterion is simply a ratio of intensities, namely m_1's intensity and a rectangular patch surrounding m_1 (m_1 is at the centre of this rectangular patch). This ratio (m_1's intensity / average rectangular patch intensity) is thus the search criterion (and is represented by the square denoted **Correlation Window** in **Figure 3.3**. The algorithm proceeds to do a search for the closest matching pixel/rectangular patch in the second image. The search area (denoted **Search Window**) in the second image is typically ¼ the size of the image. The position of m_1 in the first image serves as the central position of the new search area. Correlation

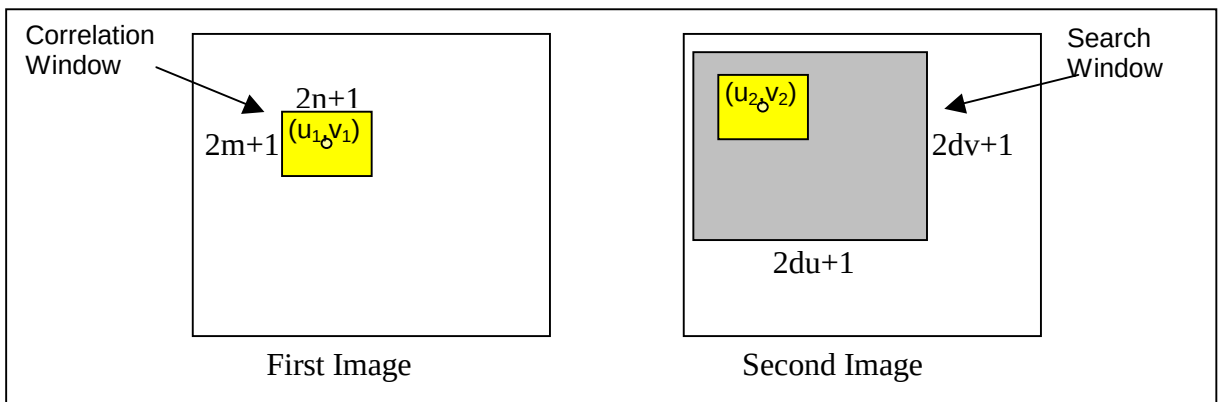


Figure 3.3 Normalised Cross Correlation between two images.

scores are computed by comparing a fixed window in the first image to a shifting window in the second image.

The NCC equation, when used for depth reconstruction, is expressed mathematically as:

$$NCC(m_1, m_2) = \frac{\sum_{i=-n}^{m}\sum_{j=-m}^{m}\left[I_1(u_1+i, v_1+j) - \overline{I_1(u_1, v_1)}\right]\times\left[I_2(u_2+i, v_2+j) - \overline{I_2(u_2, v_2)}\right]}{\sqrt{\sum_{i=-n}^{m}\sum_{j=-m}^{m}\left[I_1(u_1+i, v_1+j) - \overline{I_1(u_1, v_1)}\right]^2} \times \sqrt{\sum_{i=-n}^{m}\sum_{j=-m}^{m}\left[I_2(u_2+i, v_2+j) - \overline{I_2(u_2, v_2)}\right]^2}} \quad (5)$$

where:

- *m* and *n* indicate the size of the correlation window (also called the window size);
- m_1 and m_2 are the two pixels for which a matching score needs to be determined;
- u_1, v_1, u_2, v_2 are the x and y coordinates of the pixels respectively; and
- $\overline{I_1(u_1, v_1)}$ and $\overline{I_2(u_2, v_2)}$ represent the average intensity of the correlation window; the intensity of each pixel that forms part of the correlation window is used to determine the average intensity.

The process is illustrated in **Figure 3.3**. The correlation window in the second image is shifted around the search window and continually compared with the correlation window in the first image using the NCC evaluation above. This process is repeated until every possible correlation window in the second image has been compared to the correlation window in the first image and a maximising combination is found. The above evaluation (*NCC*) returns a matching score between –1 for a very weak match and 1 for a perfect match, depending on the strength of the match. This return value is matched against a minimum threshold; if a given matching score is below this threshold, the match can be ignored. If the threshold value is set too low, multiple matches may be returned, more so than if a high threshold (such as 0.9) is used. Additional overhead must then be incurred to try find the best match from these possibilities. A match is classified as good if the pixels surrounding the pixel at position (u_1,v_1) have matches occurring in the same vicinity as (u_2,v_2). A process is then followed which performs this check.

Once pixel matches for all pixels in the images have been determined, a reconstruction of the scene depicted in the two images can be generated. The way this is traditionally done is to make use of the **disparity** of the matches as a measure of depth for each pixel, as illustrated in **Figure 3.4**. The disparity D is the distance between the projection of P in the first image and P in the second image. The disparity of a pixel pair in this sense refers to the Euclidean distance between the pixel pair. The resulting reconstruction is referred to as a 2½D reconstruction because it represents the object only from one side.

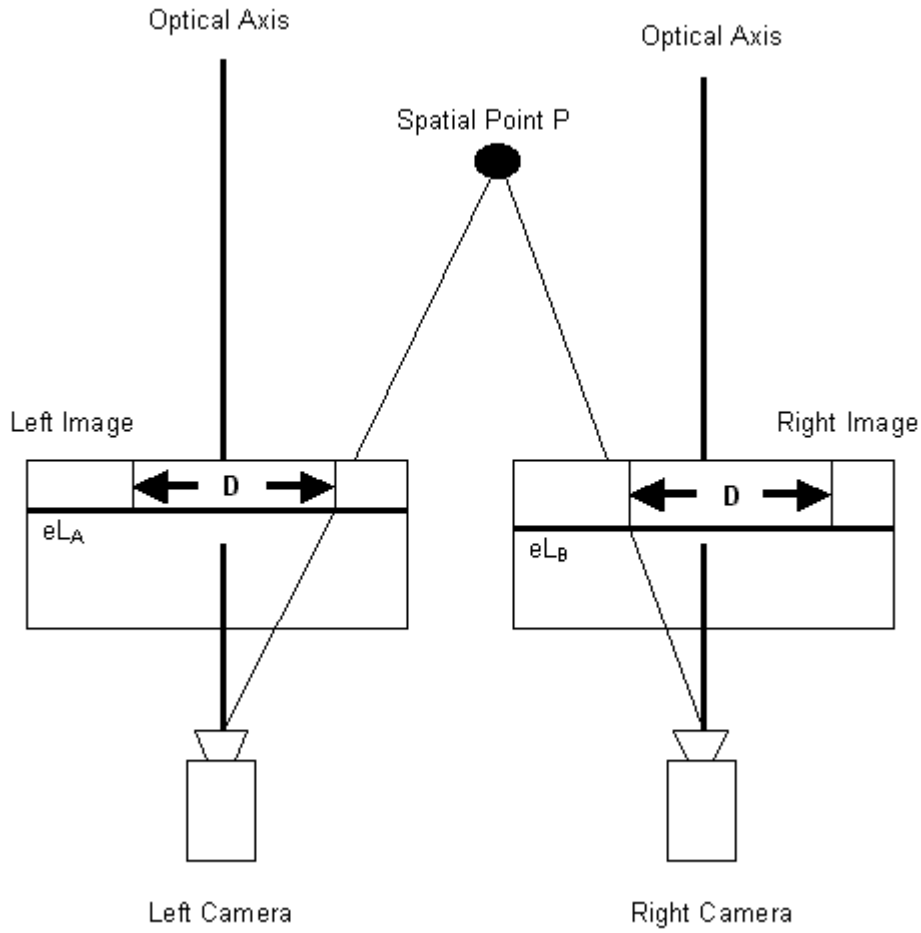


Figure 3.4 Recovery of Disparity.

The motivation for NCC has been mentioned namely, the recovery of depth information. As such, NCC offers the following advantages:

- It is robust; matches generated using the algorithm are illustrated in **Figure 3.5**. The algorithm was asked to match up the areas represented by the white circles in the left image with the associated pixels in the right image. The circles in the right image indicate the areas that were found to match those in the left image. The white arrows indicate false or incorrect matches. The correlation window size in this case is 8x8;
- In any real-world situation, both diffuse and ambient light sources are present. Rotating an object thus results in the colour on the surface of the object subtly changing. NCC overcomes any problems arising from this dynamic lighting situation by its inherent use of pixel-to-background ratios.

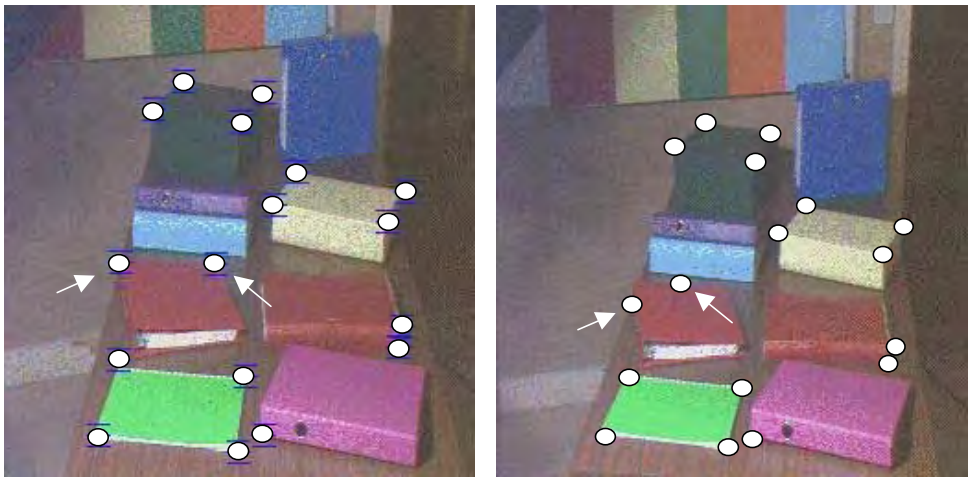


Figure 3.5 A stereoscopic image pair.

Images courtesy of the INRIA institute in France.

NCC evaluations also suffer from problems such as occlusions (where information in the first image is hidden because of rotation or translation) and false matches. Because the algorithm makes use of luminance ratios to determine the best possible match, false matches are often reported in cases where a pixel's colour matches the background colour, and in regions of very similar colour.

It is a very slow algorithm; this arises because the NCC algorithm performs a brute-force evaluation every time it tried to find a match. There is hope; what follows in the next section is a theoretical technique that offers performance enhancements for traditional Normalised Cross Correlation. Section 3.4.2.1 offers a further enhancement that can be made to the traditional, brute-force NCC implementation.

3.3.1.3.2 Epipolar Geometry

Typical reconstruction algorithms utilising NCC tend to be very slow, although hardware NCC implementations speed up the process (see Faugeras *et al.* [24] for a discussion on a hardware-based NCC implementation). The sluggishness of the NCC algorithm in its basic form can be attributed to the two-dimensional search for matches. In order to reduce the search space of NCC use is made of the Epipolar Geometry of a scene. This is referred to as the epipolar constraint. A brief summary of the basic concepts underlying this topic is provided here. See Zhang [70] for a more complete discussion regarding the theory behind Epipolar Geometry.

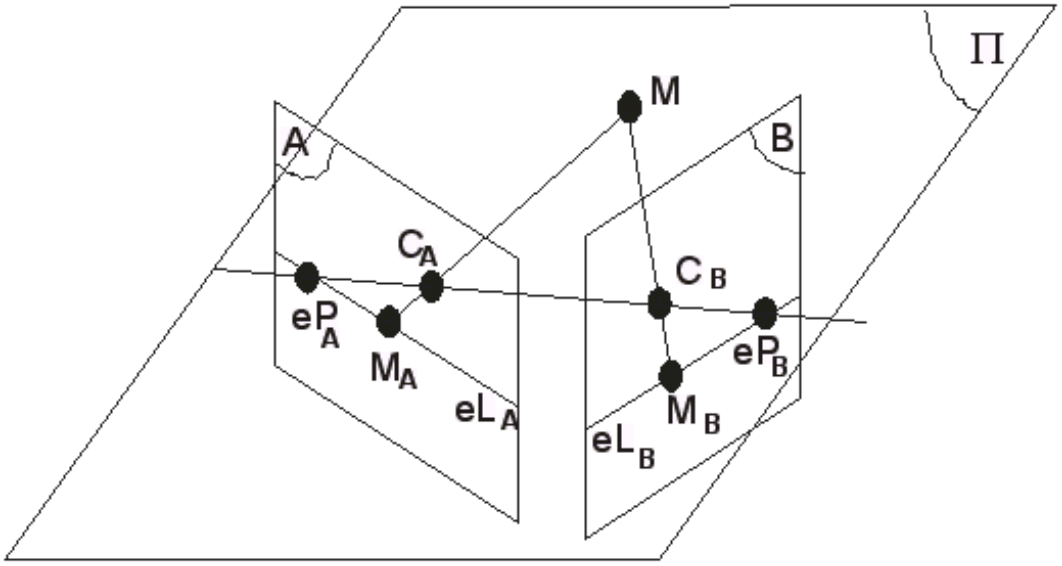


Figure 3.6 The Epipolar Geometry as it applies to stereoscopic image pair A and B.

Image courtesy of Deriche *et al.* [12], pp7.

The benefit epipolar geometry brings to the NCC algorithm in terms of performance enhancement is that it reduces the NCC search space from a 2D problem down to a 1D problem. In other words, it constrains the NCC search space to a straight line.

In a stereo setup, such as the one illustrated in **Figure 3.6**, the cameras are constrained by the epipolar geometry. This means that for any pixel M_A in the first image, the epipolar geometry will simplify the search for the matching pixel (M_B) to a single line called the **epipolar line**.

An epipolar line can be described by the following derivation: each camera A and B has an associated optical centre C_A and C_B. The images taken of the real-world scene represent a perspective projection onto some 2D plane (the input images). The real-world point M intersects image A at position M_A and image B at M_B. The plane formed by triangulating M, C_A and C_B is called the epipolar plane's (Π) intersection with each image plane in turn is a line called the epipolar line (eL_A and eL_B in **Figure 3.6**). The matching pixel for M_A, namely M_B is constrained to lie along eL_B.

The exploitation of the epipolar geometry in a stereoscopic rig requires that much *a priori* knowledge of both intrinsic and extrinsic parameters be known. The intrinsic parameters refer to the camera information, such as the optical lengths C_A and C_B, while the extrinsic parameters refer to the amount of rotation (R) and translation (T) that will position the first image plane with the second image plane. Both the intrinsic and extrinsic parameters are then combined into what is called the Fundamental Matrix. The relationship between M_A and M_B is then defined as:

$$M_A F M_B = 0 \quad (6)$$

It is possible for the Fundamental Matrix to be estimated without prior knowledge of either the intrinsic or the extrinsic parameters. This is done by manually determining at least eight matches between the stereoscopic pair of images and then substituting these into the Fundamental Matrix. This provides a unique solution for **F,** defined up to a scale factor. This approach is known as the eight-point algorithm and is attributed to Longuet-Higgins. The result of this operation is that given M_A and a partially computed F, M_B will lie on the line formed by $M_A F$.

3.3.1.3.3 Face Reconstruction

Lengange *et al.* [40] discuss a face geometry reconstruction mechanism using an epipolar geometry-enhanced NCC mechanism to generate a depth map of a face, which is then provided as input into a face recognition pipeline that consists of feature segmentation, segmentation approximation and finally, recognition. The problem with this approach is that it assumes the presence of a dense depth map. Without dedicated hardware, this approach is not feasible, since it involves the recovery of the disparity for each stereoscopic pixel pair.

3.3.1.3.4 Summary

The use of NCC to perform true 3D reconstructions suffers from a number of problems, namely:

- It is slow; dense disparity maps are required to obtain a true indication of the depth in a stereoscopic image pair. Real-time implementations (Faugeras *et al.* [24]) have made this feasible, but it is still nonetheless, problematic.
- Reconstructing a truly 3D object from multiple views (more than 2), suffers from a number also becomes problematic. The literature (Deriche *et al.* [12]) suggests that NCC is predominantly used to recover a 2½D reconstruction of any scene, and not a true 3D reconstruction. In order to generate a complete 3D model using this technique 2½D reconstructions of an object from different angles would have to be combined to generate the final reconstruction. This is a non-trivial problem.

3.3.1.4 Visual Hull Reconstruction

A number of images of the object to be reconstructed are used as input to the reconstruction process. The idea is that each image contains what is called a visual cone enclosing the object of interest. The final reconstruction is simply the intersection of all the visual cones in all the images. A major advantage these reconstruction methods have over the other methods mentioned above is cost. The cost involved is minimal, typically requiring only a CCD camera to capture the images that will be used for the reconstruction process.

The research into this reconstruction approach is quite extensive. These reconstruction algorithms typically assume that the subject lies within some known volume. This volume is usually composed of voxels. The space carving process thus involves removing voxels that do not form part of the object. Kutulakos *et al.* [37] discuss a space-carving method and proceed to classify the visual hull class of reconstructions as **least commitment** – the algorithms make no assumptions of the scene being reconstructed. This is highlighted by the fact that they do not constrain positioning of the camera around their object of interest – they simply specify that the camera position must be known. The only factor that is relied upon is the outline formed by the shape in the input images. The result of their algorithm is a number of unconnected voxels that represent the final object. Texture mapping is achieved by a process called **voxel colouring**, a term introduced by Seitz *et al.* [58]. This method reconstructs the colour (radiance) at surface points in an unknown scene. One aspect that neither Kutulakos nor Seitz address is the generation of efficient polyhedral meshes from the remaining voxels.

A structured grid of voxels is typically employed. “One distinguishes **structured** and **unstructured** meshes by the way the elements meet; a structured mesh is one in which the elements have the topology of a regular grid. Structured meshes are typically easier to compute with (saving a constant factor in runtime) but may require more elements” (Eppstein [18]).

In addition to the structured grid of voxels, the images used as input to the space carving process are aligned with the volume. This can be described as a scanline-type constraint that has the advantage that each pixel in the image maps onto a voxel in the bounding volume. This leads to a one-to-one mapping between each pixel and voxel in the volume. The scanline approach can be likened to the shear warp reconstruction method, where images are altered in some way to ensure that there exists a direct alignment and mapping between pixel and voxel i.e. each pixel in the image maps to directly to a voxel. This ensures that voxel **scanlines** map directly to the input image **scanlines**. Scanline alignment can be ensured by eliminating two problems, namely:

- **Perspective projection**: the pinhole camera model means that each image captured with any image will be subject to perspective projection. An inverse perspective mapping must be performed on the images captured. This implies some sort of camera calibration;
- **Radial distortion**: this term refers to the image points to be displaced from their proper locations along radial lines from the image centre. Lens distortion is typically caused by faulty grinding of lens elements and is very prevalent with the majority of current desktop CCD cameras.

Zhang [69] has developed a tool that, given a number of input images, generates the perspective inversion and radial distortion correction matrices for the CCD camera used to take the images. Images of a grid of a number of cubes are taken at different angles. These cubes are arranged in an 8x8 array. A minimum of three images must be taken of the grid at different angles. These matrices can then be applied to each of the images the errors mentioned above compensated for. Techniques thus exist for the compensation of these problems.

3.3.1.5 Critical Evaluation

This section provides a critical evaluation of the different reconstruction approaches discussed in this section. The aim is to determine which method is best suited for low cost avatar reconstructions.

The results of the evaluation are tabulated below.

Reconstruction Approaches					
Method	**Initial Setup**	**Economic Cost**	**Computational Cost**	**Post Processing Required**	**Results**
Laser Range Finder	1	5	1	4	5
LIT (HLS)	1	3	1	3	3
LIT (Cyberware)	2	4	1	3	3
LIT (VIVID)	1	3.5	1	3	3
NCC	1	1 – 2	5	4	5
Visual Hull	3	1	3	2	4

The following scales apply to the table:

- Initial Setup: 1 (little setup) → 5 (long time to set up) - how much setup is initially required?
- Economic Cost: 1 (very cheap) → 5 (very expensive) – how much does it cost?
- Computational Cost: 1 (very low) → 5 (very high) – how resource intensive (RAM etc.) is this method? How much processing time is required to generate the final reconstruction?
- Post Processing Required: 1 (none) → 5 (extreme) – how much post-processing is required? What is the extent of hole-filling required, texture alignment and texture generation?
- Results: 1 (very bad) → 5 (excellent) – how good are the final reconstructions? This is affected by whether the method is able to generate texture maps etc.

The breakdown of each entry in the above table is as follows:

1. The Laser Range Finder
    a. Initial setup is very quick; the scanning unit is placed facing the object to be reconstructed;
    b. Economic cost is the highest of all the approaches. This kind of equipment is not bought, it is hired;
    c. The computational cost is low; depth recovery is performed by the unit;
    d. It has the highest post-processing costs of all the approaches because it does not generate a texture map automatically and it requires hole-filling for areas that are not in direct line of site of the scanner;
    e. High resolution scanning ability (2mm up to a range of 50m) results in very high quality reconstructions.

    Comments - The biggest advantage of this approach is that it is the only feasible reconstruction mechanism when large-scale reconstructions are required. An example would be the reconstruction of large buildings for architectural walkthroughs where the blue prints for the building have been lost;
2. Light Illuminated Triangulation (HLS)
    a. Initial setup is very quick; the scanning unit is placed in close proximity of the object. Size of the object is also not a constraint.
    b. The economic cost is quite high ($17000), although it is the lowest of the LIT range of scanners.

c. The computational cost is low. Only the scanned parts of the surface are recorded, leading to low memory requirements. The depth recovery is performed by the unit
d. The post-processing requirements are average because, although holes are minimized (due to the unit being moved around the object thereby minimizing occlusional discontinuities), the generation of a texture map is time-consuming. Additionally, though, the scanning of red/black surfaces that absorb the laser, will lead to holes.
e. The quality of the results that can be achieved is dependent on the object. This system seems to work very well with objects that do not have reflective surfaces, such as faces, but suffers with reflective surfaces. Additionally, the reconstruction of red or black objects is problematic. Very good results can be achieved given the right conditions.

3. Light Illuminated Triangulation (Cyberware)
    a. Initial setup is relatively quick; the largest drawback of this system is the size of the reconstructions that are supported.
    b. Economic cost is very high - it is the highest in this comparison after the Laser Range finder category;
    c. Computational cost is very low again – depth recovery calculations performed by the scanner;
    d. Post-processing requirements are relatively high: textures are automatically generated, but the issue of occlusions means that hole-filling is a necessary post-processing requirement. This method suffers from the same problems as the HLS scanner in that the use of a red laser results in poor reconstructions with objects that are able to absorb the colour of the laser.
    e. The final results are extremely dependent on the colour and surface properties of the object being scanned. An object with a shiny surface or a surface capable of absorbing the red laser light will be reconstructed very poorly. The reconstruction results in general, however, are very good.
4. Light Illuminated Triangulation (VIVID) - The results for the VIVID scanner are the same as for the Cyberware scanners, bar the initial setup and the cost. This range of scanners is more portable than the Cyberware range, and thus received a lower value for initial setup. It is also cheaper than the Cyberware scanners but more expensive than the HLS scanner.
5. NCC approaches

a. Initial setup is relatively quick; a stereo pair of images is taken of an object.
b. Economic cost is very low (1) – it requires only a CCD camera. The cost does increase moderately with the purchase of dedicated hardware to perform Real-Time NCC evaluations.
c. Computational cost is very high: depth is recovered using a time-consuming NCC evaluation of the entire image to obtain a dense depth map
d. Post-processing requirements are quite high, with hole-filling again becoming a problem. More than one 2½D reconstruction from different angles is required to generate a true 3D reconstruction.
e. Although this method also suffers from occlusions, it does not make use of a laser to recover depth. Unlike the LIT reconstruction methods, the quality of the final reconstruction does not depend on the colour of the surface of the object being reconstructed. The only factor affecting the quality of the final reconstruction is the issue of occlusions. Additionally, this method does not suffer from perspective distortions;

6. Visual Hull approaches
    a. Initial setup is slow; the object is placed on a turntable and images taken of the object at different angles. Additionally though, background elimination has to be considered;
    b. Economic cost is very low – it requires only a CCD camera.
    c. Computational cost is average – A voxel space is used that contains the object. A high amount of memory is required for this approach. Depth is recovered using a space carving approach. The computational cost is thus much lower than the NCC based approach because the tedious NCC calculation is avoided.
    d. This approach requires the least amount of post-processing out of all the previous approaches. It is able to automatically generate textures, and because each voxel is initially assumed part of the object, there is no possibility for holes in the polygon mesh being generated due to occlusions.
    e. The fundamental problem with this class of algorithms is that they are unable to accurately reconstruct objects containing many concavities. Additionally, it does also not attempt to correct for perspective distortions, and as such, the quality of the final reconstruction suffers. It therefore received a lower rating than the NCC approach for the final reconstruction result.

With cost being the primary factor, the first four approaches are not feasible for the purposes of low cost avatar reconstructions. From the above evaluation, it can be seen that the NCC approach generates the best results, but is an extremely time-consuming reconstruction approach. Although real-time NCC implementations facilitate use of this method, the issue of lack of availability of this kind of hardware forces this method to be avoided.

The Visual Hull reconstruction approaches are most suited to solve the problem of low cost avatar reconstructions.

3.3.2 Facial Modelling

This section deals with techniques that are employed to reconstruct/create human heads.

3.3.2.1 MPEG-4 Face Specification Standards

The recently ratified MPEG-4 standard includes the specification of what are classified as Face Definition Parameters (FDPs). FDPs are responsible for the deformation of a generic head to suit the real-world subject. An extract from the official MPEG-4 documentation is as follows: “The ‘Face Animation’ part of the standard allow sending parameters that calibrate and animate synthetic faces. These models themselves are not standardized by MPEG-4, only the parameters are.” (Koenen, [34])

The specification defines two types of model coding, namely: facial action parameters (discussed below) and face definition parameters. The standard thus defines the following:

- “ A definition and coding of face animation parameters (model independent):
    - Feature point positions and orientations to animate the face definition meshes
    - Visemes, or visual lip configurations equivalent to speech phonemes
- A definition and coding of face definition parameters (for model calibration):
    - 3-D feature point positions
    - 3-D head calibration meshes for animation
    - Texture map of face
    - Personal characteristics
    - Facial texture coding;” (Koenen, [34])

Thee face animation parameters thus specify model independent animation parameters, while the face definition parameters are used to deform and texture a generic head to suit a specific

subject. The information specified by the FDPs with specific reference to generic face modelling as identified by Lee *et al.*[39] are:

- FeaturePointsCooord – specifies feature points for calibration of the proprietary face. They are responsible for deformation of the generic head template to suit the real-world subject;
- TextureCoords – the texture coordinates corresponding to the feature points;
- TextureType – a hint to the decoder on how best to perform texturing (e.g. cylindrical, spherical);

There are a total of 84 FDPs that are officially listed as part of the standard (Lee *et al.* [39]).

3.3.2.2 Profile Fitting

This technique involves the deformation of a generic, low polygon count head to suit the profile of a specific individual. Photographs of the subject are taken from the front and the side, and then the generic head is deformed to fit the profile of the individual in the photographs. An example of this approach is described by Lee *et al* [39]. They manually demarcate the different FDPs on the input images and then use a Free Form Deformation technique called Dirichlet FFD (DFFD) to deform the generic head to suit the subject's characteristics. The demarcated FDPs are treated as control points for the deformation of the head. The process is completed by applying a texture map (generated from the input images) to the head. The images from the front, left side profile and right side profile are stitched together to form a texture map that is then fitted to the 3D head. To remove artefacts of the stitching process from the texture map, a Gaussian operator is employed to smooth the joins. Finally, the teeth and tongue are added to the model as a post-processing step to improve the realism of the head.

Escher *et al.* [19] discuss a profile fitting approach that is very similar to Lee *et al*'s method. The difference with Escher's approach is to define a number of **characteristic points** that are points of interest measured on the real subject's head. The points on the generic head that correspond to the points on the real head are determined. Additionally, the generic head is also subdivided into rectangular patches. This is to "allow for the estimation of the positions of the non-given characteristic points or the correction of the low accuracy of other given points" (Escher *et al.* [19]). DFFD is then used to deform the head according the front and side profiles of the user. Texturing is performed by mapping a frontal view of a face with a neutral expression orthogonally to the face.

3.3.2.3 Automatic Profile Fitting

Fua *et al.* [26] discuss the reconstruction of heads using epipolar geometry. Disparity maps are computed for a group of stereoscopic images of a subject's head, and a 3D model is generated from the disparity maps. A coarse control mesh is then attached to a generic 3D animation model and a least squares optimisation process applied to the mesh. The result is that the coarse control mesh is deformed to suit the reconstructed shape obtained previously. Finally, the albedo (defined below in section 3.3.3) of the original object's surface is computed to ensure the closest possible resemblance between the animation model and the actual object.

The advantage of their method is that it requires minimal user intervention and it works very well with systems that make use of a predefined animation model. Additionally, outliers that are part of the initial reconstruction are filtered out and do not form part of the deformed animation model. The problem with their method is the requirement of dense disparity maps for the reconstruction of the object.

3.3.2.4 Anthropometric Face Modelling

DeCarlo *et al.* [11] describe anthropometry as the "biological science of human body measurement". Examples of uses for this field include face reconstruction from body remains and the recovery of missing children by ageing their appearance in photographs. Just as above, where the MPEG-4 FDP parameters have been used to deform a generic head to suit the profile of a specific person, the use of Face Anthropometry dictates what the general structure of a specific class of individuals is. Face Anthropometry involves the accurate measurement of facial structure. The result of this accurate measurement implies that measurements taken of the same face at different times will return very similar results. Face Anthropometry has the interesting ability to classify a person based on the specific measurements of his/her head into a specific population group; it has been found that faces are statistically population-dependent (DeCarlo *et al.* [11]). This means that the proportions of a certain person's face dictate the population group to which that person belongs. DeCarlo *et al.* [11] use this observation as a basis for their generation of general head models from anthropometric measurements. The resulting face model is generated by a process called constrained optimisation.

A B-spline surface representing the human head is essentially a tube with openings for the mouth and neck. Anthropometric landmarks are then placed at appropriate positions around the

surface, with some of the landmarks enforcing constraints on what the final shape for that part of the face should be.

The result of this process is a B-Spline representation of a generic human head that fits a specific population profile.

3.3.3 Texture Mapping

Texture mapping, specifically onto reconstructed objects, causes the object to look realistic.

The advantage texture mapping offers is that in general it gives the effect of scene complexity and realism without the need to explicitly model a scene using many polygons. A good analogy to illustrate this is provided by Woo *et al.* [68] (pp 318-319), namely that of a brick wall. A brick wall typically consists of many bricks. To model this type of object without texture mapping would involve representing each brick as a separate polygon and then shading each polygonal brick to look like a real brick. Using texture mapping, however, a picture of a real-world brick wall can be pasted on to a single flat polygon representing the wall. From a distance, the flat plane actually looks like a brick wall. There are many other examples one can mention to illustrate the uses of texture mapping. Another typical case is in the first-person shooter game genre where scenes containing boxes or pipes are rendered using textures.

Texture generation with Cyberware laser scanners occurs as an object is scanned and a depth map generated. The result is a cylindrical texture map that maps to the depth map.

The concept of albedo affects any reconstruction mechanism that employs photographs. Visual Hull Reconstruction approaches, for example, suffer from this problem. The albedo of a particle is given by the “amount of light scattered by this particle in all directions in relation to the amount of incoming light” [53]. The fact that an object is textured from photographs implies that any part of a surface containing specular effects in the original images will be disseminated to the textured object. This issue becomes problematic when there is a strong diffuse light illuminating one side of an object, and causing darkness on the other side. The result is that this illumination will filter through to the texture map. It is thus necessary to maximise ambient light and minimise diffuse lighting and specular reflections.

3.3.4 Polyhedral Decimation

Schroeder *et al.* [57] define traditional decimation as a method of data compression or “polygon reduction”. Traditional approaches to decimation involve reducing the number of polygons in

areas where the curvature of the object surface is small, while maintaining detail (or polygons) in areas of high curvature. Typical decimation algorithms perform the following operations during decimation of an object:

- Evaluate the neighbours surrounding the current vertex (**Vertex Classification**);
- If the current vertex is less than a specified distance from the plane generated by connecting the neighbouring vertices together, remove it; this removal results in a hole in the object; (**Planar Evaluation**);
- Perform reconnection of the neighbouring vertices to remove this hole (**Geometry Restoration**).

Decimation algorithms are simple to implement and typically result in simplified object representations, while maintaining surface continuity of the original shape; the basic shape of the object is preserved.

3.4 Image-Based Tracking

This section firstly introduces a high level classification of different image-based tracking approaches that exist. This is followed by a discussion of practical implementations that a number of authors have employed. The goal of the latter discussion is to illustrate the diversity of approaches that exist.

3.4.1 Classification

Ekman *et al.* [17] provide an overall classification of image-based tracking approaches or the analysis of motion, as they call it. This classification is discussed here.

3.4.1.1 Feature Tracking

Motion analysis is performed by tracking only a certain number of points in the scene, specifically on the face of the subject. Initially, an image frame is evaluated to determine interesting points, such as edges. Features are then tracked between points to determine their respective motion vectors. This method has the advantage of being an efficient tracking algorithm, because the search space is restricted to a few points of interest. The flipside of this is that only a limited number of motion vectors can be used to try to represent the entire motion in

the scene. As a result, the expression analysis phase will not represent the subject's true expression, because his whole face is not tracked.

3.4.1.2 Pattern Tracking (Template Matching)

Whereas feature tracking methods perform tracking on edges and other interesting image features, pattern tracking introduces the concept of tracking higher-level image elements. This is done using a number of template images containing the element that must be tracked. The algorithm thus has a good idea of the structure of the object for which it must search. The primary advantage of this approach is that the tracking process is directed only at the parts of the image representing the object, while ignoring background noise. The disadvantage of this method is that because every object is unique, a number of images of the object to be tracked must be captured, and then used as input to the tracking process. Additionally, the tracking process is susceptible to failure if the object rotates too much. As a result, the input template images must include the object under different pose conditions. This also means that the advantage of decreased search space afforded by the template matching process is offset by an increased number of evaluations that have to be performed as the number of templates is increased; as the number of input templates increases, the performance decreases proportionally.

3.4.1.3 Optical Flow Analysis

This section is a summary of the discussion afforded by Beauchemin *et al.* [5]. Optical flow techniques provide an approximation of image motion defined as the projection of velocities of 3D surface points onto an image plane. It is based on the hypothesis that "the intensity structures of local time-varying image regions are approximately constant under motion for at least a short duration" (Beauchemin *et al.* [5]). This translates to the following intensity function:

$$I(x+dx,\ t+dt) = I(x,t) + f(dx,dt) \tag{7}$$

where dx is the displacement of the local image region at (x,t) after time dt. One can measure the displacement of small image patches by correlation. These image displacements can then be used as an approximation to the image velocity.

"To date, except in limited circumstances, no technique is able to generates sufficiently accurate and dense optical flow fields to allow the general recovery of the motion in a realistic environment" (Beauchemin *et al.* [5], pp 42).

Optical flow complexity is increased by the problems of occlusion, transparent motions (attempting to track transparent surfaces), and non-rigid objects. Whereas the traditional optical flow algorithms only perform computations on an image sequence at a single resolution, recently more hierarchical approaches have been adopted, and are based on coarse-to-fine frameworks.

3.4.1.4 Model-Based Tracking

These approaches make certain assumptions about the shape of the object of interest. The shape is then used as a parameter for the tracking process. Birchfield [6] makes use of an ellipse to approximate the head's contour. The perimeter formed by the ellipse is used to maximise the normalized sum of the image gradient around the ellipse. The ellipse has a fixed orientation and aspect ratio. According to the author, the tracker is able to overcome several problem common to image-based trackers, namely, occlusions, full body rotations and reacquisition (tracking an object over an extended period of time, given that it does not appear in the image for a certain duration). He claims real time tracking rates of 30 fps on a 133 MHz Intel Pentium.

3.4.2 Correlation-Based Tracking

Most of these methods originate from computational stereopsis, which attempts to correlate areas in different images under perspective projection. It is assumed that, locally, distortions caused by a shift of the viewing angle are negligible. The biggest problem with correlation based tracking methods is that of occlusions. The occlusion of tracked points often leads to false matches being tracked.

With a high amount of motion in the scene, non-hierarchical correlation algorithms also become very sensitive to false matches. This is due to the increased search space that has to be accommodated to handle the faster motion. Okutumi *et al.* [46] propose a statistical model of disparity within correlation windows that assumes that disparity values are constant, but the further they are away from the centre of the search window, the higher the uncertainty of disparity. The further the matched point is from the centre of the search window, the more unlikely it is that the point has been correctly matched.

3.4.2.1 Enhanced Normalised Cross Correlation

Lewis [42] discusses an enhancement to the basic NCC matching strategy introduced previously (section 3.3.1.3.1). The aim is to utilise NCC's reliable matching ability to perform tracking of objects in video sequences. To do this in software, Lewis discusses the inherent inefficiencies with the traditional brute force NCC approach, and introduces optimisations to speed up the algorithm. Normalised Cross Correlation implies that the pixel-correlation window ratio in the source image is compared to every pixel-correlation window ratio in the search window area of the target image. This algorithm in its most basic form has complexity $O(n^2)$. The pixel-correlation window ratio involves repeatedly calculating the average for the XY correlation window, as illustrated in **Figure 3.7**, during the evaluation of the search window area. This process is repeated for the entire search window area, and the match returning the highest correlation is selected. The effect an increasing correlation window size has on performance has been illustrated and discussed in section 3.3.1.3.1; as the window size increases, so the performance degrades. An intuitive approach to improving the performance is to tackle the

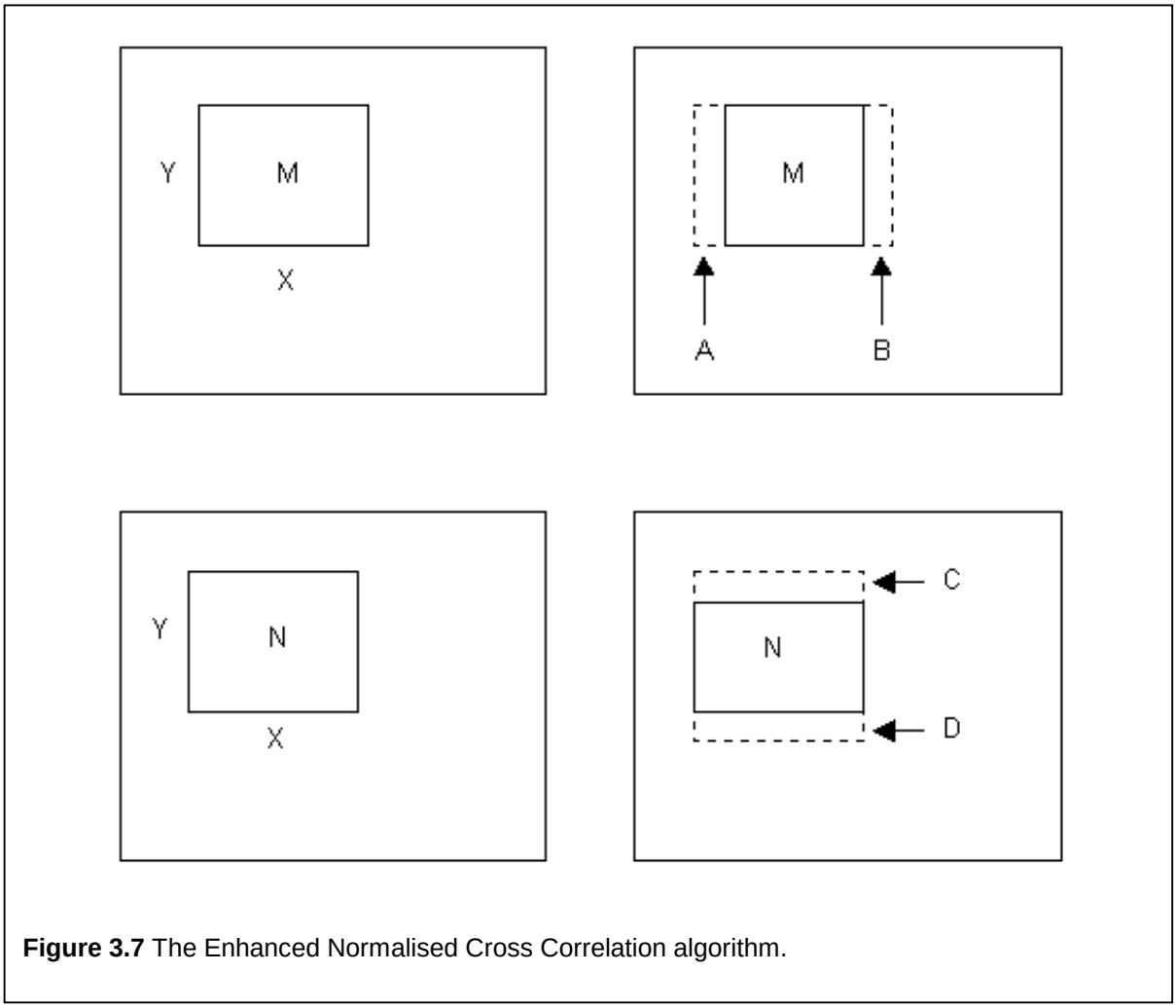


Figure 3.7 The Enhanced Normalised Cross Correlation algorithm.

repeated calculation of the correlation window.

The enhancements that can be made to the traditional brute force NCC algorithm are twofold. The first involves the horizontal movement of the correlation window M. This is illustrated by the first row in **Figure 3.7**, where M moves to position M+B-A. latter enhancement occurs when the background area moves vertically; N moves to N+D-C.

Firstly, the ratio of the pixel to background average the pixel being searched in the right image is calculated. This is of the form:

$$Pixel\_Ratio_{left} = Pixel_{left}/CW\_Avg_{left} \quad (8)$$

where CW_Avg_{left} refers to the calculated average of all the background pixels surrounding the $Pixel_{left}$.

The search part of the algorithm involves similar Pixel_Ratio calculations for each pixel in the search window in the right search window. The first step during this process is to calculate the $Pixel\_Ratio_{right}$ for the top left pixel in the search window in the right image. Once this area has been evaluated and compared to $Pixel\_Ratio_{left}$ the search advances. CW_Avg_{right} for the next pixel (X+1) is recalculated and the NCC comparison performed again. This process is repeated until all pixel-correlation window combinations have been compared to $Pixel\_Ratio_{left}$.

A major source of inefficiency with the brute force approach is the number of repeated additions that are performed during the calculation of the CW_Avg value for each Pixel_Ratio in the right search window. This is because the CW_Avg rectangles overlap, as illustrated in **Figure 3.7**. The solution to this repetition involves the addition and subtraction of rows and columns from a total average. Instead of completely discarding a given CW_Avg calculated in the first step, the second CW_Avg (moving horizontally by 1 pixel) can be calculated by subtracting the first row of pixels and adding the new row of pixels. Thus, when rectangle M in **Figure 3.7** moves horizontally, CW_Avg_m can by updated by adding column B and subtracting column A to the total used to calculate CW_Avg_m. Similarly, moving rectangle N vertically involves adding the row D to the total and subtracting row C from the total used to compute CW_Avg_n.

Lewis [42] reports that the result of this alteration to the algorithm is that the dependence of performance on correlation window size is partially lifted; as the correlation window size increases, the performance penalty incurred is minimal.

3.4.2.2 Hierarchical Approaches

Hierarchical methods attempt to improve the accuracy of traditional correlation-based methods particularly in the presence of large motion. At the same time, they promise to reduce the computational expense of these algorithms by restricting the search space. Images are decomposed into different scopes of resolution in the form of Gaussian or Laplacian pyramids. It has been mentioned that these algorithms are based on coarse-to-fine frameworks and this is illustrated in **Figure 3.8**. "Low frequency channels are used to estimate large disparities which can be refined by adding higher frequency channels into the matching process. The Laplacian pyramid allows for the estimation of large inter-frame disparities and helps to enhance image structure" (Beauchemin *et al.*[5]). Well-known implementations that adopt this approach include Anandan [1] and Singh [59]. Anandan [1] makes use of a Laplacian pyramid and SSD-based (sum of squared difference) matching strategy. Singh's [59] approach is similar to that of

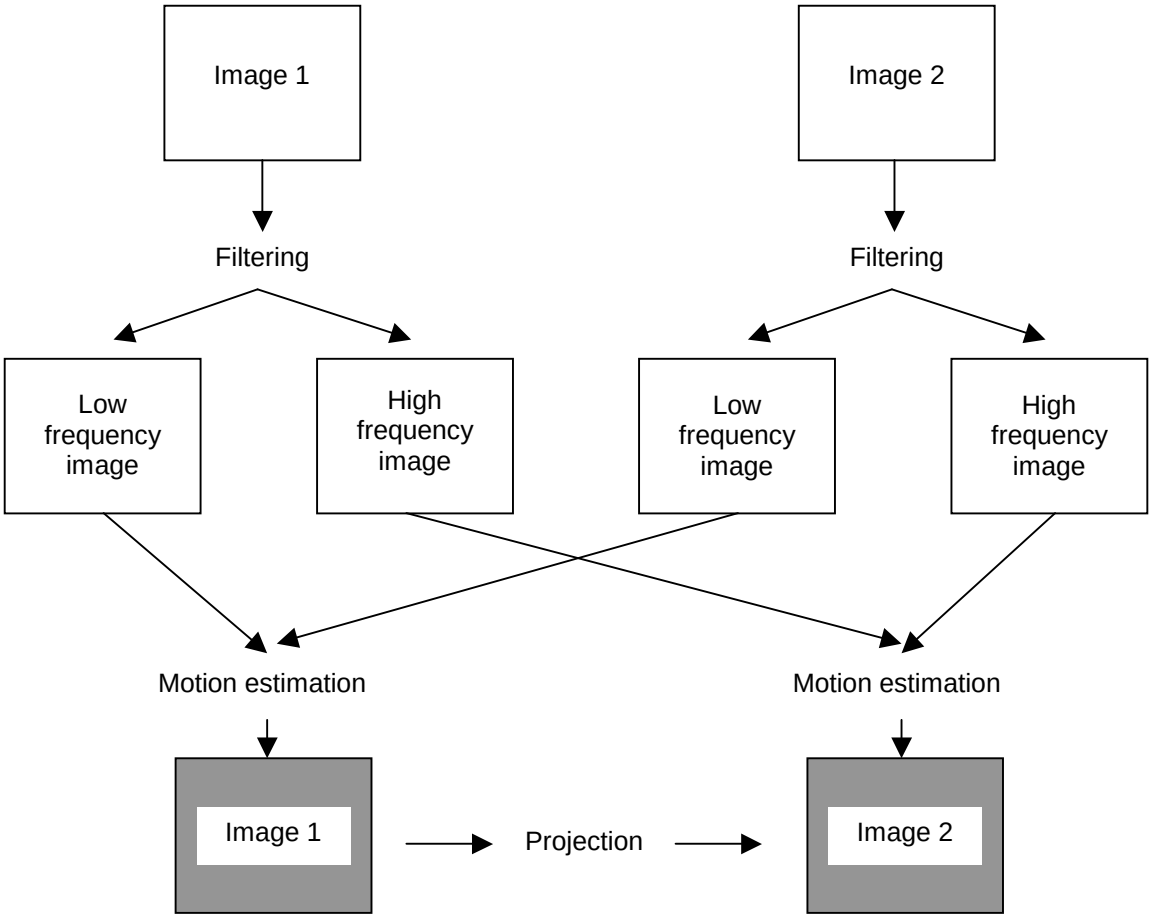


Figure 3.8 Coarse to fine motion estimate subdivision.

Source: Beauchemin *et al.* [5]

Anandan in that an SSD-minimisation process is adopted, but velocities calculated with the SSD-minimisation are propagated using a neighbourhood smoothness constraint.

Applying this approach to successive frames in a video sequence means that the areas of motion can be quickly determined, and the correlation approach then applied to these areas only, and not blindly to the entire image. These methods thus add a level of intelligence to tracking implementations.

3.4.3 Background Elimination

Motion analysis via background elimination offers quite attractive benefits. A simplifying assumption with an application like videoconferencing is that some part of the scene will remain constant; this tends to be the background scenery. It is thus possible, using this assumption to reduce the optical flow search space by removing the background from a conferencing video stream. This leaves only the subject currently involved in the videoconference in each of the frames.

Rekimoto [54] has completely implemented a motion tracking system based on background elimination. His subtraction implementation makes use of the YUV colour spectrum to perform colour comparisons because, according to him, it provides more robust calculation results than intensity calculations. These findings have been confirmed by the author, since lighting conditions affect RGB measurements. This occurs because intensity affects each of the principal components. With the YUV colour space, intensity is constrained primarily to the Y component. Rekimoto, [54] makes use of face template matching to determine the exact position of the user's head in the frame. The system calculates correlation coefficients between the template and each pixel in the foreground area. The area returning the highest score is classified as the subject's face. The disadvantage of this approach is that the tracking fails if the subject rotates his/her head too much, or if the subject's facial expression is very different from the original template.

Background elimination nonetheless offers benefits in terms of search-space reduction for later correlation approaches.

3.4.4 Blob Growth

Oliver *et al.* [47] have developed a tracking system based on the use of the blob-growing mechanism discussed in section 3.2.2.3. The blob they search for has a normalised colour that, according to them, can be used to reliably locate flesh-coloured parts of the image. A connected-

blob algorithm is then employed to generate blobs of the skin-coloured parts of the image. The face is located using this mechanism. Use is then made of Hidden Markov Models to classify patterns of behaviour, namely facial expressions and head movements. They mention that they have used the algorithm to perform real-time whole-body tracking, as well as the real-time recognition of American Sign Language hand gestures. The advantage of the system, that they highlight, is its ability to perform tracking, even in complex scenes.

3.4.5 Fiducial Point Tracking

Burford *et al.* [8] discuss the use of **fiducial points** for the purposes of tracking. The term fiducial refers to an interesting point on a subject's face. This method to tracking involves the placement of brightly coloured points on the face of the subject. The process of tracking then involves the quick filtering of the input image to determine the location of the points. An extension to this involves the use of different coloured dots to easily determine the relative positions of the different dots.

3.5 Pose Estimation

Pose estimation or pose determination refers to the recovery of position and orientation of a real world object from a number of images. The process of pose estimation can be defined as "extrinsic (external) camera calibration" (Carceroni *et al.* [10]). This field is central to computer vision and has found application in augmented reality. Carceroni *et al.* [10] classify the process of tracking real-world objects into 2 distinct categories, namely **Acquisition**, and **Tracking Proper**. Acquisition involves evaluating the image and locating the target object, while Tracking Proper refers to the process of continually tracking the target object.

Firstly, a summary is provided of the classifications of pose recovery, and this is followed by a number of practical implementations that have been used to perform pose estimation.

3.5.1 Pose Recovery Classifications

There are a number of traditional ways of extracting pose. These are (as listed by Carceroni *et al.* [10]):

- **Analytical Perspective Solutions**: actual image-based measurements are used to perform pose determination. The general approach here is to track a fixed number of points and

then convert the image coordinates of the tracked points into pose information. Approaches based on this method can be further classified as Point Correspondences, Line Correspondences, and Angle Correspondences. The most important problem with this class of algorithms is that of **ambiguity**. Ambiguity here refers to the generation of multiple solutions, with only one of these solutions representing the true pose of the subject. A simple method employed to overcome this ambiguity involves re-projecting the object features back into the image plane "so as to build a synthetic image of the scene. Then, the discrepancy between the appearance of each feature in the re-projected image and the appearance of the same feature in the actual image can be used to determine whether the pose used in the re-projection is likely to be the desired solution or not." (Carceroni *et al.* [10], pp 17). To further refine the solution space, visibility tests are performed as well. Solutions representing points that are not visible in the image, are removed as well. The source of most of the problems with this class of algorithms, is the inherent non-linearity in images due to perspective transformations. This is further complicated by the introduction of lens distortion;

- **Weaker Camera Models (affine solutions)**: effects of perspective projections can ignored if all the points in a 3D object are roughly at the same distance from the image plane of the camera. This is a valid assumption when the distance between the object and the camera is relatively big when compared to the size of the object. This model can yield precise pose estimation at a low computational cost;
- **Numerical Perspective Solutions**: these methods combine the generality of the perspective camera model with the robustness of affine solution methods. Lowe's Algorithm is discussed as an elegant approach falling into this category (Carceroni *et al.* [10], pp37). "He assumes that the relative orientation between the camera and the target can be represented by three angles measured about coordinate axes that are fixed with respect to the camera. This representation system is known as *roll*, *pitch* and *yaw* (RPY) angles".

3.5.2 Model-Based Approaches

Models can be used to constrain the estimation process. Ekman *et al.* [17] classify the tracking process into two categories, namely: tracking using simple and general object, and then the use of more complex and object-specific models. The simple model is used to locate the object in a scene quickly, according to the object's general shape. The second category deals with pose

recovery, specifically with the alignment of a 3D model of an object such as a face and a head, to the image data. Additionally, it also facilitates the detection of subtle facial features.

Black *et al.* [7] propose a model of rigid facial motion using a collection of local parametric models. The image motion of the face, mouth, eyebrows and eyes are modelled using optical flow models. They motivate this choice by focussing on the on the accuracy of the pose estimation; whereas more enhanced models (such as ellipsoids) could be used to obtain tracking results that are more accurate, the approximate results using a plane are acceptable.

They model the majority of the face using a plane. They do state that models that are more complex can be employed, but the attractiveness of a planar approach is in its simplicity (when compared to other models); the motion of a plane can be described by eight parameters. The advantage of more complex models is that they return more accurate information about the 3D motion of the head.

They make the point that, taken as a whole, facial motion can be quite complicated. By decomposing the face into different regions, however, the motions for each can be modelled very simply.

They classify facial feature tracking into a number of categories, namely:

- **Rigid** : movements due to head rotation;
- **Articulated :** movements due to lower jaw motion during speech;
- **Deformable motions :** movements due to muscle contractions and expansions that accompany speech and facial expression changes.

They make use of a prediction mechanism to guess where the face will be in the next frame.

A more complicated model-based tracking approach is discussed by Basu *et al.* [4]. The human head in their system is approximated using a 3D ellipsoid[3]. The assumption here is that this shape accurately models the general shape of the human head, and this allows for more accurate pose estimation.

3.5.3 Monocular Pose Recovery

Horprasert *et al.* [30] discuss the estimation of orientation of a human face using a single monocular image. They make use of the symmetry inherent in the human face to recover yaw and roll, and use anthropometric measurements to determine the pitch component.

---

[3] Hence the derogatory term “egghead”.

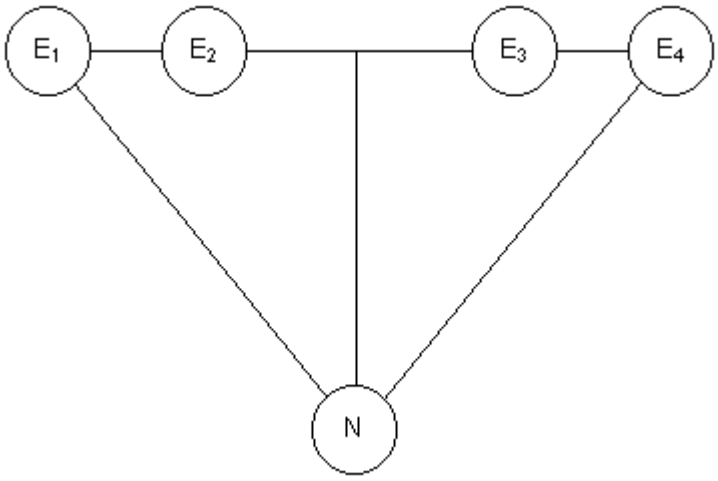


Figure 3.9 Monocular Pose Estimation.

They assume both eyes and the nose are visible. They perform feature-based tracking of five fiducial points, one on each side of every eye and one at the tip of the nose, as illustrated in **Figure 3.9**. E_{1-4} denotes the points surrounding each eye and N represents the nose tip. They state that "although five points are not sufficient for recovering orientation" when nothing is known about the camera parameter, use can be made of cross ratios to facilitate projective invariance from the face symmetry. Additionally, they make use of statistical modelling for face structure from anthropometry to estimate the three rotation angles.

They assume that the focal length of the camera is known in advance. Each of the three angles that combine to specify the orientation of the head, namely roll, pitch and yaw is calculated independently as follows:

- Roll recovery is relatively straightforward and utilises projected eye displacements as a measure of roll;
- Yaw is the most complex of the calculations. The recovery of this angle is based on the assumption mentioned previously, namely that the four eye points are collinear. The projective invariance of cross-ratios is then employed to determine the final angle. They do state that the calculation of the yaw angle is dependent only on relative distances between the four corners of the eyes and the focal length. It is independent "of the face structure and distance of the face from the camera. It is also not influenced by other parameters such as the translation of the face along any axis;
- Pitch (Rotation about the X axis) recovery makes use of a lookup table on statistical anthropometric measurements in the final angle determination. The distance from each of

the outer eye fiducials to the nose tip. Additionally the calculation compares the projected nose length to the average anthropometric length defined in the lookup table to determine the final angle.

They are thus able to extract the three pose parameters from a single monocular image.

3.6 Expression Analysis

This section provides a classification of approaches that have been adopted for classifying an expression depicted on a subject's face. The remainder of this section introduces other authors' approaches to solving this problem. These range from template matching to rule-based approaches.

3.6.1 Classification Approaches

As will be seen from the discussion in this section, the theme underlying most expression analysis approaches is the decomposition of facial movements into a collection of independent muscle movements that, when combined, result in a specific expression being identified. Additionally, each of the systems presented below proposes to abstract the actual model used from the rules that define the expression.

3.6.1.1 Facial Action Coding System

"The Facial Action Coding System (FACS) notation is meant to describe visible facial expressions" (Pelachaud *et al.* [50]). It is based on anatomical studies. It is assumed that every facial action can be attributed to muscular activity, relaxation, or contraction. FACS describes temporary changes in facial appearance, how a feature is affected by specifying its new location, and the intensity of the changes. An action unit (*AU*) corresponds to an action produced by one or more (related) muscles. Each *AU* describes the direct effect of a muscle by eventual secondary motion due to the propagation of movement, and possible appearance of wrinkles or bulges.

An example of happiness as defined according the FACS system is: a combination of pulling lip corners (*AU* 12 + 13) and/or mouth opening (*AU* 25+27) with upper lip raiser (*AU* 10) and bit of furrow deepening (*AU* 11) (Essa *et al.*[22]).

Essa *et al*. [22] state that the definition of all possible facial expression using FACS is "extremely difficult". They point out that FACS does not include any timing facility in its

specification of an expression, a factor that is a critical parameter to recognising emotions. They go on to develop a muscle-based representation of facial motion, which they then use to recognise facial expressions. This is achieved by building a video database of facial expressions and then probabilistically characterising the facial muscle activation associated with each expression. Additionally, they make use of a physics-based model to generate spatio-temporal motion-energy templates for each expression exhibited by the subject. The templates are then used to classify the expression.

3.6.1.2 Minimal Perceptible Actions

"A *minimum perceptible action* is a basic facial motion parameter. The range of this motion is normalized between 0 and 1 or -1 and 1. MPAs include facial feature movement (eyebrows, jaw, mouth) as well as non-facial actions such as nods or turns of the head or eye movement."

"An expression is then made up of a set of MPAs, providing a higher level of abstraction. The MPAs can combine to show natural expressions such as anger, fear, or surprise, or unnatural and idiosyncratic expressions. Expressions also have associated intensities to reflect stronger or more feeble expressions." Kalra *et al.* [32].

3.6.1.3 MPEG-4 Face Animation Parameter Set

This section is a summary of the discussion afforded by Koenen [34] regarding Face Animation Parameters. The MPEG-4 Face Animation Parameter (FAP) set is based on the study of minimal facial actions and are closely related to muscle actions. They represent a complete set of basic facial actions, and facilitate the representation of most natural facial expressions. Exaggerated values permit one to define actions that are normally not possible for humans, but which can be used to give appropriate expressions to cartoon characters.

All parameters involving translational movement are expressed in terms of facial animation parameter units (FAPU). These units refer to measurements between different key facial features, and so provide an independent mapping between the FAPs and model/subject specific FAPUs. The table below illustrates three examples taken from official FAP set entries.

Facial Animation Parameter Unit Sample Entries				
FAP Name	**FAP description**	**Bi-directional / Uni-directional**	**Direction of movement for positive intensities**	**Measurement units**
Move_h_l_eyeball	Horizontal movement of left eyeball	B	Right	Degrees
Depress_chin	Upward and compressing movement of the chin	B	Upward	Intensity (0 –10)
Push_pull_lowerlip	Protruding or sucking movement of the upper lip	U	Forward	Distance from mouth to nose

3.6.2 Face Bunch Graphs

This approach is used primarily in the recognition of facial features. The assumption they employ is that the face can be described using a standardized structure. In order to allow for the many kinds of variations that may occur with faces (beards, glasses, gender etc.) a sample population is grouped together and **labelled graphs** specified for each face. A labelled graph refers to the representation of the models used. It “consists of a set of nodes and a set of edges connecting these nodes. Each node is labelled with a set of featured and the edges are used to code the topography (i.e. where the eyes and mouth in a face are located)” Kruger [36].

The grouped labelled graphs are referred to as a Face Bunch Graph (Kruger *et al.* [35]). It refers to a collection of individual label graphs and "has a stack-like structure and combines labelled graphs of individual sample faces". An example of a face bunch graph is illustrated in **Figure 3.10**.

Each node in the face bunch graph maps to a number of associated labelled graph nodes. The node is referred to as a **jet**. Each jet represents a collection of **fiducial** points. "All *Jets* referring to the same fiducial point, e.g. all left-eye jets, are bundled together in a bunch, from which one can select any jet as an alternative description. The left-eye bunch might contain a male eye, a female eye, both closed or open, etc. Each fiducial point is represented by such a set of alternatives and from each bunch any jet can be selected independently of the jets selected from the other bunches" (Kruger *et al.* [35]).

Maurer *et al.* [43] discuss a system that is able to locate and track facial features in a labelled graph system. They claim robust tracking at 10 fps. The concept of the Bunch Graph is not restricted to faces but can be applied to any class of objects with a comparable standardization of structure, a fact confirmed by Maurer *et al.* [43].

3.6.3 Template Matching

These approaches make use of a predefined set of expressions for each subject. This approach is very similar to the template matching approach to motion analysis as discussed above, but differs

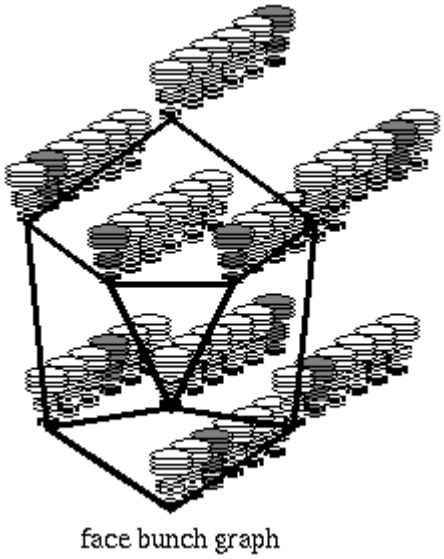


Figure 3.10 A typical Face Bunch Graph. Each node represents a collection of jets.

Source: Kruger *et al.* [35]

with respect to the use of the templates. The input templates represent the specific subject exhibiting different expressions. The expression analysis process attempts to perform classification by comparing the current expression to the list of templates. This approach suffers from the same problems as those mentioned in the discussion on template matching above, namely rotational inconsistencies, increased number of templates to facilitate more possible expressions (thereby leading to a decrease in performance), and a specific database for each subject that will be exposed to the system.

3.6.4 Blob Growing

A tracking mechanism employing blob growing was discussed in section 3.2.2.3. The discussion by Oliver *et al.* [47] also highlights the use of their system to perform real-time expression analysis. Hidden Markov Models are used to evaluate and classify the different shapes of the mouth once the face has been located.

3.6.5 Snake Analysis

"A snake is a conformable contour which seeks to minimize an energy function, **E**, determined by the snake's path through the image (internal deformation energy, $\mathbf{E_s}$) and by image forces acting on the snake (external potentials, **P**). A scalar measure of the total external potentials, ***P***, is the integral sum of **P** over the path of the snake. Depending on the function used for the external potentials, the snake will seek out bright or dim paths, edges, or other features of interest in the image. One can imagine the external potential as a function defined over two dimensions, and whose value corresponds to the height of a relief map or topography of a hilly terrain. Like its biological counterpart, a snake seeking to minimize its potential energy would travel through the valleys and around the hills, and may even wrap around an area to form a closed contour" (Terzopoulos *et al.* [61]).

The human face consists of many feature lines and boundaries. It is very amenable to tracking using snakes. A snake can be defined as a numerical solution of a first order dynamic system (Thalmann *et al.* [63]). It is essentially a dynamic deformable 2D contour in the XY plane. A discrete snake is a set of nodes with time-varying positions. The nodes are coupled by internal forces, making the snake act like a series of springs resisting compression and a thin wire resisting bending. For the purposes of expression analysis, snakes can also be used to estimate muscle contraction parameters in a sequence of facial images.

Tracking approaches that employ snakes are not usually very robust because numerical integration may become unstable over time.

3.6.6 Rule-based Classification

Ekman *et al.* [16] have identified six principal emotions that can be universally associated with distinct facial expressions. These are: happiness, sadness, surprise, fear, anger, and disgust. Ekman and Friesen have identified a number of facial characteristics that, when combined allows the current expression exhibited by the user to be classified. Happiness, according to Ekman *et al.* [16], has the following characteristics:

a) corners of lips are drawn back and up;
b) mouth may or may not be parted with teeth exposed or not;
c) a wrinkle runs down from the nose to the outer edge beyond lip corners;
d) cheeks are raised;
e) lower eyelid shows wrinkled below it, and may be raised but not tense;
f) crow's-feet wrinkles go outwards from the outer corners of the eyes.

Similarly, a face can be classified as sad if the following constraints are adhered to:

a) inner corners of the eyebrows are drawn up;
b) skin below the eyebrow is triangulated, with inner corner up;
c) upper lid inner corner is raised;
d) corners of the lips are drawn or lip is trembling.

Yacoob *et al.* [7] employ a rule-based method to perform expression analysis. They base their expression analysis system on work done by Ekman *et al.* [16]. They subdivide the expression analysis process up into three temporal segments, namely: the beginning, apex and end. Taken together, these three segments form what is called a temporal model. Therefore, each possible expression has a temporal model associated with it. They have taken the idealised expression rules specified by Ekman *et al.* [16] and simplified them. A smile is classified as follows:

a) [Beginning] - mouth curves upwards AND expansion of mouth;
b) [End] – mouth curves downwards AND mouth narrows.

Similarly, sadness has the following rules:

a) [Beginning] – mouth curves downwards AND brow moves upward and inwards;
b) [End] – mouth curves upwards AND brow moves downwards and outwards.

A similar rule-based classification is also employed by Goto *et al.* [28]. Their aim is the extraction of MPEG-4 FAP data from a live video stream. To that end, they develop an MPEG-4

animation system based on the tracking of image edges in real time. An initialisation stage is performed whereby areas of interest, particularly the mouth and eyes in this case, are demarcated. This facilitates the tracking process without the use of fiducial markers. The system they use involves a head mounted camera. As such, the camera's position remains constant with respect to the head. They only track a small part of the subject's nose to determine the face position.

The actual tracking process is decomposed into two parts, namely mouth tracking and eye tracking. For mouth tracking, they make 3 assumptions, namely:

A) The positions of upper teeth remains constant;
B) The movement of the lower teeth is dependent on jaw rotations;
C) Basic mouth shape is dependent on bone movement.

The results of the mouth tracking are thus dependent on whether the teeth are visible or not. During initialisation, the position of each pupil is specified. Information regarding the strength of the edge pupil is extracted from this, and used for later tracking.

The mouth tracking algorithm is able to detect whether the mouth is open or closed. The distinction is drawn by evaluating the continuity of a vertical line drawn from the nose tip to the chin tip. All edges intersecting the vertical line are identified. Subsequent to this, an energy maximisation process is performed to determine which of the edge intersections is the best possible candidate for a mouth classification. The actual classification of whether a mouth is open or closed is performed as follows:

- "*Closed Mouth*: The centre edges appear strong, the outer two edges appear weak, and the teeth are hidden inside;
- *Opened Mouth***:** As shown in the figure, when teeth are present, the edges are stronger than the edge on the outside lips, or between a lip and the teeth or between the lip and the inside of the mouth. If the teeth are hidden by the lips (upper or lower) then of course the edge of the teeth is not detected" (Goto *et al.* [28]).

Having determined the positions of the different edges, the information is compared to data in a generic shape database, and the appropriate mouth shape information selected.

Eye tracking is decomposed into three different tasks, namely:

A) The calculation of pupil positions – An approach similar to the mouth tracking approach is employed.
B) The calculation of eyebrow positions – a box surrounding each eyelid is employed and a process similar to the mouth tracking is used.

C) The calculation of eyelid positions – This classification is dependent on the amount of the eyelid determined to be visible in the calculation of the pupil position. Additionally, if the eyebrow is lower than its original position, the eyelid is assumed to be closed.

They (Goto *et al.* [28]) claim to achieve between 10 and 15 fps on a 300 MHz Intel Pentium II.

3.7 Expression Generation

The process of expression generation is achieved by deforming the avatar representation in some predefined way. The use of the FACS or FAP systems introduced in the previous section maps to a paradigm of collective deformations; expressions are composed of a collection of smaller facial movements. The generation of an expression is facilitated through a collection of individual deformations that, when applied to the avatar simultaneously, result in the desired expression being generated.

This section firstly describes an MPEG-4 based expression generation system. Although this part of the discussion is specific to MPEG-4, it nonetheless illustrates how expression generation is typically approached. Following this, a number of issues relating to the field of traditional free form deformation (FFD) are introduced. Subsequently, a classification of deformation approaches is provided. This section is concluded with in-depth discussion on vertex interpolation-based deformation for the purposes of real-time deformation.

3.7.1 MPEG-4 Expression Generator

Lee *et al.* [39] discuss a system that has been developed to facilitate “real-time interactive animation system based on FAPs”. They have implemented a multi-layered system that uses high-level actions to interface with the MPEG-4 FAP-FDP specifications, according to a multi-layered approach. The hierarchy consists of four layers, with each layer having an increased level of abstraction over the previous layer. They are:

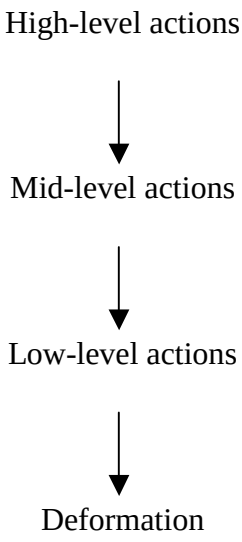
A) High–level actions – the user specifies how the intensity of the expression changes over time. This maps to an intensity graph;

B) Mid-level actions – a snapshot of the face exhibiting a specific expression. The positions/expressions are defined by the user. Each expression has a collection of expressions. This layer defined the movements of the face during an expression change;

C) Low-level actions – defines the positions of the FAPs on the face geometry when it exhibits a neutral expression;

D) Deformation – the layer responsible for the final deformation of the avatar. They make use of an enhanced FFD approach called Rational Free Form Deformation (RFFD);

The breakdown of hierarchy is as follows:



The high-level actions layer is the only layer visible to the animator. This layer allows the user to specify **basic emotions**, **visemes**, and **user-predefined expressions**, thus providing a high level of control and abstraction over the expression generation process. The basic emotions correspond to one of the six emotions that can be specified in the first field of the FAP definition, namely: joy, sadness, anger, fear, disgust and surprise. The visemes (visual phonemes) correspond to the second field of the MPEG-4 FAP specification. Duration information is specified as part of each viseme.

Additionally, they also describe weighted approach to performing action blending, according to the following equation:

$$FAP_{Final} = \frac{\sum_{i=1}^{n} Weight_{i,t} \bullet Intensity_i \bullet FAPValue_{i,t}}{\sum_{i=1}^{n} Weight_{i,t}} \quad (9)$$

where:

- *n* refers to the number of high-level actions that modify the FAP value;
- *Intensity* is the global intensity of the high-level action *i*; and
- *FAPValue* is the FAP movement in the high-level action *i* at time *t*.

The inclusion of *Intensity* allows them to specify the same expression, but at varying degrees. The example they provide is of a small smile as opposed to a big smile.

Additionally they also discuss the animation of the mouth area based on a speech co-articulation system. The system is composed of four conversion components, namely:

- Speech-to-text;
- Text-to-phonemes: a speech synthesis system is applied to the text to generate phonemes;
- Phonemes-to-speech: the phonemes are finally converted into a synthetic voice;
- Phonemes-to-visemes: use is made of a correspondence table to determine mouth movement according to a specific phoneme.

Finally, they (Lee *et al.* [39]) discuss the development of an application tying the above components together. Escher *et al.* [20] describe the development of a similar system.

3.7.2 Deformation Classifications

This section provides a brief overview of the theory underlying FFD and then goes on to classify and discuss two broad deformation approaches, namely:

- **Free Form Deformation** algorithms, and
- **Vertex Interpolation** approaches.

3.7.2.1 Traditional FFD Issues

Gain [27] provides an in-depth discussion about deformation. He discusses the development of an optimised deformation system specifically geared toward arbitrary polygon-based 3D representations. He provides a robust deformation implementation with regards to polyhedral meshes, but Gain's discussion highlights the inherent inefficiencies that plague typical deformation systems. Topology and correctness of the polygonal mesh must be maintained. Gain mentions a number of factors that must be considered when attempting to deform polygons. These are:

- **Self-intersection** issues: responsible for maintaining integrity of the mesh;
- **Refinement** and **decimation** issues – here attempts are made to maintain the topology/surface curvature of the polygonal mesh. The idea of decimation has been discussed in section 4.2.5. Refinement is the opposite of decimation. Whereas decimation allows for the number of vertices in a polyhedral mesh to be reduced, the process of refinement involves the addition of more vertices to a polygonal mesh as the mesh is

deformed. This process ensures the maintenance of C^1 and C^2 surface continuity ([27], pp 79). planar calculations for every vertex have to be performed to ensure surface continuity. This implies that, for every vertex in the polyhedral structure, the vertices connected to that vertex (as specified in the edge graph) have to be evaluated, a process that elevates traditional deformation algorithms to complexity greater than $O(n^2)$ in most cases. Gain provides a table of efficiency versus extra space for numerous least square solution methods (see [27], pp 47). The most efficient algorithm here has complexity $O(m^2n-(1/3)m^3 -1/2m^2p + 2mnp)$, where n and m refer to an n x m deformation matrix, while p represents the size of the solution matrix.

3.7.2.2 Free Form Deformation Approaches

These approaches allow an object's topology to be altered by moving certain control points surrounding the object to affect its shape. The maintenance of C^1 and C^2 surface continuity is important. To this end vertices are added to and removed from the deforming object to guarantee a realistic deformation, as well as surface continuity guarantees. This process tends to be computationally expensive, and has been avoided. For a more complete discussion on traditional FFD issues, refer to Gain [27].

With traditional FFD, the polyhedral representation of an object is converted internally, into a B-Spline representation, which is subsequently deformed, and for rendering purposes converted back into a polyhedral representation.

Eisert *et al.* [15] make use of such an initial B-Spline representation instead of a polyhedral representation. Control points are used to deform an object. The object is composed of B-Spline patches. They claim the following benefits:

- Smoothness of deformation: C^{n-1} surface continuity is maintained, where n refers to the order of the B-Spline representation;
- Local control: deformations are localised, which is appropriate for expression modelling;
- Affine Invariance: the same result is obtained if a surface is deformed per vertex or per control point. Deformations are defined as manipulation of a small number of control points instead of applying transformations to each individual vertex.

This approach, while easier to control and deform is computationally more expensive, because the B-Spline representation must be converted into a polygonal representation;

3.7.2.3 Vertex Interpolation System

These approaches simulate the complexities of traditional FFD algorithms by moving vertices that are part of the object. This is done without the introduction of or removal of vertices from the object structure. This facilitates real-time deformation of an object at the expense of the surface continuity and self-intersection guarantees.

The equation below illustrates a typical vertex interpolation mechanism. It is discussed by Hung *et al.* [31].

$$P' = P + K \times \left[\frac{1 + \cos\left(\frac{P_o - P}{R} \right) \Pi}{2} \right] \times (P - P_d) \times t \qquad (10)$$

As one can see from Equation 10, the complexity of this algorithm is approximately *O(n)*. The components of the equation represent the following values:

- P = current vertex in the polyhedral mesh;
- K = direction of muscle contraction (`1` denotes a contraction while `-1` represents an expansion);
- P_o = source force; this indicates the position where the deformation force originates;
- P_d = point of target force; this value represents the 3D position of where the deformation force ends;
- R = radius of influence; the authors recommend that the radius cover approximately the same area as a bounding box surrounding the vertices being deformed;
- t = time value varying between 0 (no deformation) and 1 (complete deformation);

The radius of influence constrains the effect of the deformation on the topology. One constraint with this system is that singularities occur as R approaches 0. An animation using this system can be achieved by cycling the value of the time parameter between 0 and 1. This algorithm allows deformations to be repeated, with the same results being achieved for specific deformation parameters every time.

Another advantage of this deformation algorithm is that it allows deformations to be scripted very easily, which is beneficial and suits the requirements of an expression manager. As Hung *et al.* [31] explain, the idea is to subdivide the object to be deformed (in this case, a face) into several independent bounding boxes. Each bounding box has its own deformation parameters that control the deformation for that part of the object. The vertices representing the left cheek are therefore defined as belonging to a **left cheek bounding volume**, while those vertices

representing the right cheek belongs to the **right cheek bounding volume**. Consequently, each bounding volume possesses independent deformation parameters. The left cheek's source and target forces will for example specify that the cheek must expand, while the right cheek must contract. Hung *et al.* [31] make use of this deformation approach for scripting facial expressions using a low-polygon count model.

3.8 Summary

This chapter provides broad overview of the theoretical underpinnings of the field of model-based coding. The theory underlying each of the core components in the design introduced in the previous chapter, is discussed.

Normalised Cross Correlation and Background Removal techniques are classified as principal concepts in section 3.2, since they arise more than once in the system design detailed in the previous chapter.

Model acquisition is decomposed into general reconstruction techniques and the more specific approaches to avatar reconstruction based around the deformation of generic head models through profile fitting in section 3.3. Section 3.3.1.5 discusses a quantitative evaluation of existing general reconstruction approaches to choose the best approach for the purposes of avatar reconstruction. The criterion that dictates the choice of reconstruction approach, namely visual hull reconstruction, is one of cost. Visual hull techniques and Normalised Cross Correlation approaches share the advantage of being low-cost solutions, but NCC approaches are computationally more expensive. Other more expensive approaches evaluated were Laser Range finder techniques (too specialised and expensive) and laser illuminated triangulation (final reconstructions dependent on surface colour, and too expensive).

Image-based tracking is discussed in section 3.4 by firstly classifying typical approaches to the problem and then discussing specific implementations by various authors. Four classifications are identified in section 3.4.1, namely: feature tracking, pattern tracking, optical flow analysis and model-based tracking. Correlation-based tracking is discussed in section 3.4.2 as a way of tracking objects given perspective projections. These approaches use NCC to perform tracking. They are susceptible to the problems of high motion in a scene and to the problem of occlusions. An enhanced NCC algorithm is introduced in section 3.4.2.1 for speed up the traditional brute-force NCC evaluation. Additionally, hierarchical approaches are discussed in section 3.4.2.2 to overcome the problems of fast motion. Background elimination is introduced in section 3.4.3 as an approach to reducing the tracking search space by removing the background from a scene and

preserving only the object of interest. Blob Growth techniques make use of *a priori* knowledge of skin colour to locate regions in an image representing skin, as described in section 3.4.4. With fiducial point tracking (section 3.4.5), a number of brightly (possibly differently) coloured dots are attached to a subject's face. The image containing the subject is then filtered out and the location of the points determined very quickly.

Pose estimation is introduced in section 3.5.1 by classifying traditional approaches into three very distinct categories, namely: analytical perspective solutions, affine solutions, and numerical perspective solutions. Two approaches by various authors, namely model-based approaches and monocular approaches are discussed in sections 3.5.2 and 3.5.3 respectively. Model-based approaches attempt to model the pose of a subject by constraining the estimation process. The monocular approach discusses the use of anthropometric and face symmetry assumptions to extract the roll, pitch and yaw of a subject's head given the evaluation of a single image containing the subject.

The FACS, MPA and MPEG-4 FAP expression classification systems have discussed in section 3.6.1 as an introduction to expression analysis (section 3.6). All of these classification systems work on the assumption that each expression can be decomposed into a collection of individual facial movements that, when combined, constitutes the expression. Section 3.6.2 discusses the use of face bunch graphs to provide robust face location, tracking and expression analysis. Template matching approaches (section 3.6.3) involve the use of a predefined collection of expressions of a specific subject to perform expression classification. Subsequent to the location of a subject's face using a blob-growing algorithm, as described in section 3.6.4, the face is analysed and classified appropriately. Snake analysis (section 3.6.5) has been used to determine the shape of the mouth primarily. The two rule-based classification systems discussed in section 3.6.6 are based on specific projected measurements (such as the projected mouth length). Expressions are classified as being active if certain combinations of projected lengths are measured.

An MPEG-4 expression generation system is used in section 3.7.1 to introduce the typical approaches that underlie expression generation (section 3.7). Traditional FFD algorithms are discussed in section 3.7.2, with the focus being on the inherent inefficiency of traditional FFD algorithms because of surface continuity and self-intersection guarantees that they attempt to address during deformation. Deformation approaches are subsequently classified into two broad categories, namely: traditional, and vertex interpolation. Vertex interpolation deformation, as discussed in section 3.7.2.3, avoids the guarantees provided by traditional FFD approaches and is thus faster. A sample vertex interpolation deformation algorithm is discussed.

Chapter 4 - Model Acquisition

4.1 Introduction

This chapter describes the acquisition of facial models. This process involves the generation of an avatar that represents a subject as accurately as possible. The generation of an avatar, particularly a 3D head model of a person is a subset of modelling any real-world object. Unfortunately, the modelling of real-world objects is a hard problem, typically leading to expensive solutions that are impractical in most situations.

The goal of the research presented in this chapter has been to develop a general and low cost 3D model reconstruction system, well suited to facial modelling. It must also be cheap, requiring minimal investment to achieve reasonable results.

4.1.1 Model Acquisition

A quantitative analysis of general reconstruction techniques has been provided in the previous chapter. The choice of the visual hull reconstruction approach as a viable and cost-effective reconstruction solution has been motivated.

The remainder of this chapter is devoted to a model acquisition algorithm similar to the visual hull reconstruction method. As mentioned in section 3.3, this class of algorithms make use of a voxel-based cubic volume that contains the object to be reconstructed. The volume is initialised as a 3D structured grid of voxels.

4.1.2 Description of Algorithm

The object to be reconstructed is placed on the top of a turntable. Using a camera connected to a computer, snapshots of the object are taken at different angles. These images are used as input to the reconstruction process. Associated with each image is the position and orientation of the camera.

Five steps are applied during the visual hull reconstruction process. The following five steps constitute the reconstruction pipeline:

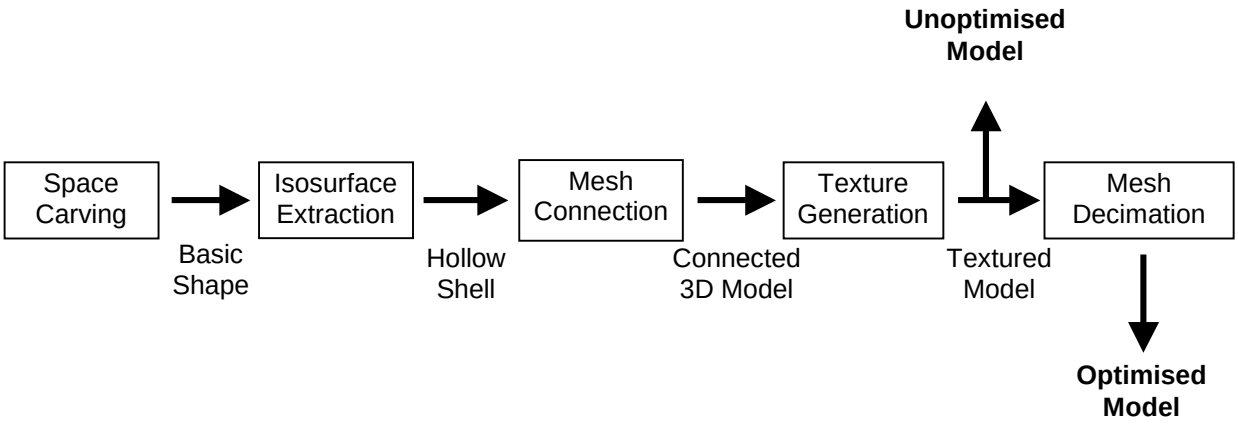


Figure 4.1 The Reconstruction Pipeline.

- **Space Carving** : the basic shape and texture of the object is recovered from the pictures taken. The recovery is done by carving a number of silhouettes out of the bounding volume. The shape is represented as a collection of voxels;
- **Isosurface Extraction** : the voxels not representing the surface of the object are removed. The result is a hollow shell of voxels;
- **Mesh Connection** : the hollow shell is converted into a closed, connected polyhedral mesh;
- **Texture Generation** : the polyhedral mesh is textured to simulate the real-world object;
- **Mesh Decimation :** the polyhedral mesh is optimised to generate a low-polygon count model.

The reconstruction pipeline is illustrated in **Figure 4.1**. Given a number of images as input, the components constituting the pipeline generate a closed connected, and textured 3D polyhedral mesh. The caption below each arrow describes the result that component generates. The decimation of the Textured Model is optional in the design, thus leading to an exit point in the pipeline before execution is transferred to the Mesh Decimation component.

All components up to the Texture Generation component form an integral part of the pipeline. At this junction, however, the pipeline allows for either:

- a completely unoptimised model to be generated as output, or
- for mesh decimation to be applied to the polyhedral mesh.

One simplifying assumption that is made at initialisation of the reconstruction pipeline is that that the bounding volume is constrained to have the same dimensions as the images containing the object, and used during the reconstruction process below. The reconstruction process thus affords a conversion from a voxel-based representation to a closed, connected polyhedral mesh representation.

4.1.3 Motivation

The reconstruction system outlined in this chapter represents a merging of the different approaches mentioned in the previous chapter. This results in a system that takes advantage of the benefits offered by each reconstruction approach. This means that a visual hull reconstruction mechanism is employed to obtain the shape of the object using an inexpensive CCD camera. To facilitate real-time rendering speeds, the visual hull approach is extended. This extension involves the removal of voxels not representing the surface of the reconstructed shape. A 3D textured polyhedral representation is then generated from the shell of vertices. This is similar to results that can be obtained from laser scanning systems, with the added advantage that the use of a camera allows a texture to be generated. This is in contrast to the HLS system from Polhemus (see **Figure 3.1**).

The quality of the final reconstruction is very important. Because the assumption is made that every voxel is part of the final reconstruction, the resulting polyhedral shape does not have any holes. Typical reconstruction methods, especially the laser scanner approaches, are unable to reconstruct areas that are occluded to them. The occluded parts of the surface result in surface holes in the final reconstruction. The HLS scanner is able to handle this to a certain degree, because it can scan any given orientation. The majority of these systems, however, perform hole filling as a standard post-processing technique. The proposed model acquisition system is able to handle the reconstruction of occluded areas to a certain degree, requiring no hole-filling post-processing. The way the pipeline handles occluded voxels is that that they are not removed by the space carving component, and thus included in the final reconstruction.

To summarise, the choice of this algorithm motivated by the following benefits:

- it is cheap when compared to other systems, requiring only the use of a CCD desktop camera;
- reconstructions are general, and not constrained to faces only;
- the resulting reconstructions are efficient, closed, textured polyhedral representations of the real-world object;
- post-processing requirements are minimal;
- The final reconstruction does also not depend on surface colour. As mentioned in section 3.3.1.5, Light Illuminated Triangulation techniques (that typically use red laser light), suffer from this problem when trying to reconstruct black or red surfaces.

The development of an implementation based on this approach needs only a computer with and CCD desktop camera connected to it to be deployed. This adheres to the design criteria that the system be low cost, and so is very attractive.

4.2 Reconstruction Pipeline

This section discusses each of the components of the reconstruction pipeline illustrated in **Figure 4.1** in more detail. The reconstruction resulting from this reconstruction pipeline is a closed connected textured polyhedral mesh.

4.2.1 Space Carving

The input to the pipeline is the bounding volume of voxels and the images of the object taken from different angles. The space carving component removes voxels not representing the object.

4.2.1.1 Design

The algorithm forming the basis of the space carving component is illustrated in **Figure 4.2**. The cylinder represents the object that is to be reconstructed, while the cube represents the bounding volume of voxels in which the object is assumed to lie. Initially, the background is removed from all the images taken of the object, leaving only a silhouetted outline.

At initialisation, all voxels in the bounding volume are initialised as **active**. For each of the images taken, the bounding volume is rotated by the inverse of the angle of rotation associated with that image. Subsequently, an orthographic projection of the bounding volume onto the image plane is performed. Any active voxel that projects onto a background pixel in the image becomes inactive. If an active voxel maps to a pixel not having a background colour, the pixel's colour is recorded for that voxel. This is performed for later texture generation. (This is a very basic description of the texture generation process; a more complete summary of the process is provided in section 4.2.4).

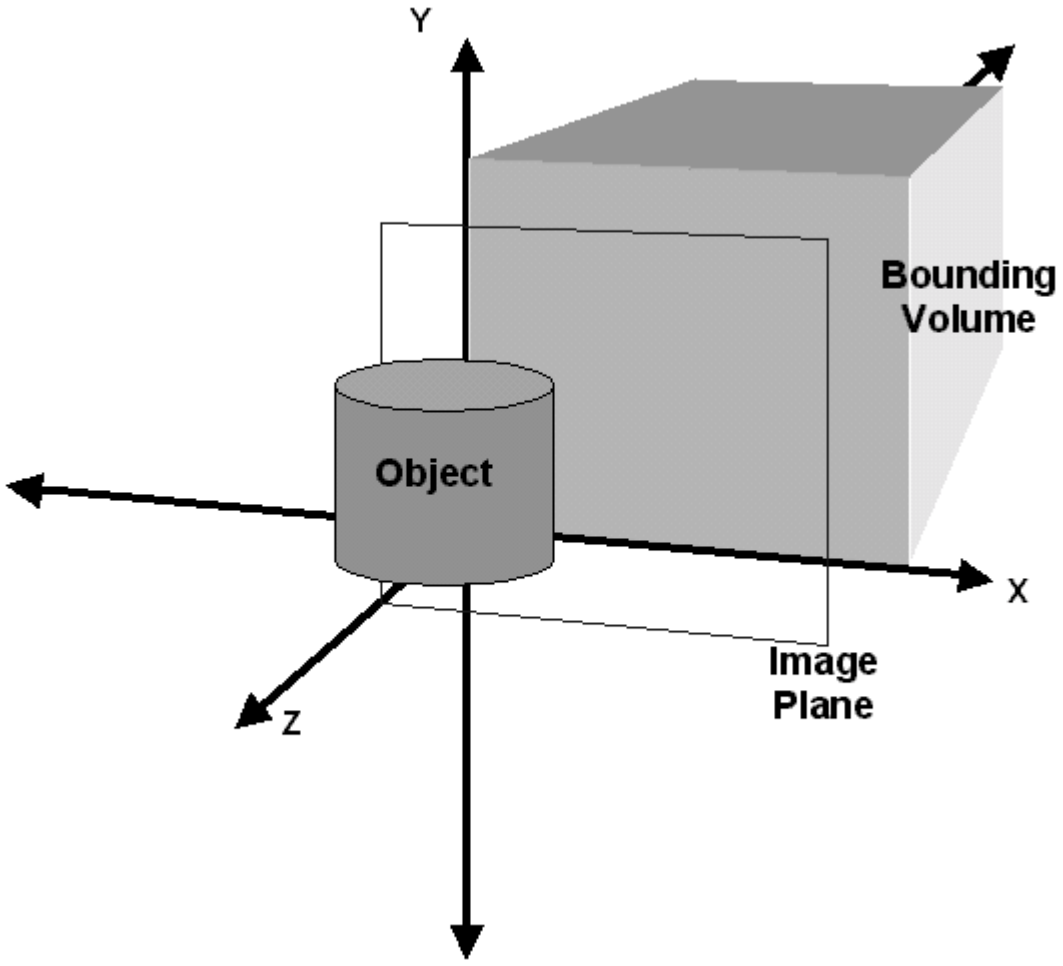


Figure 4.2 The conceptual layout of the Reconstruction Pipeline.

4.2.1.2 Implementation

The bounding volume is constrained to have the same dimensions as the images used as input e.g. a 100x100 image will map to a 100x100x100 bounding volume of voxels. This facilitates scanline-alignment, as mentioned in section 3.3.1.4.

Pixels in the input images are classified as background pixels if their colour matches a pre-defined colour, for example, white. Before running the algorithm, it is assumed that a background elimination algorithm has been applied to the input images. **Figure 5.14** illustrates results of a sample background elimination algorithm that has been used.

The algorithm for the space carving process is as follows:

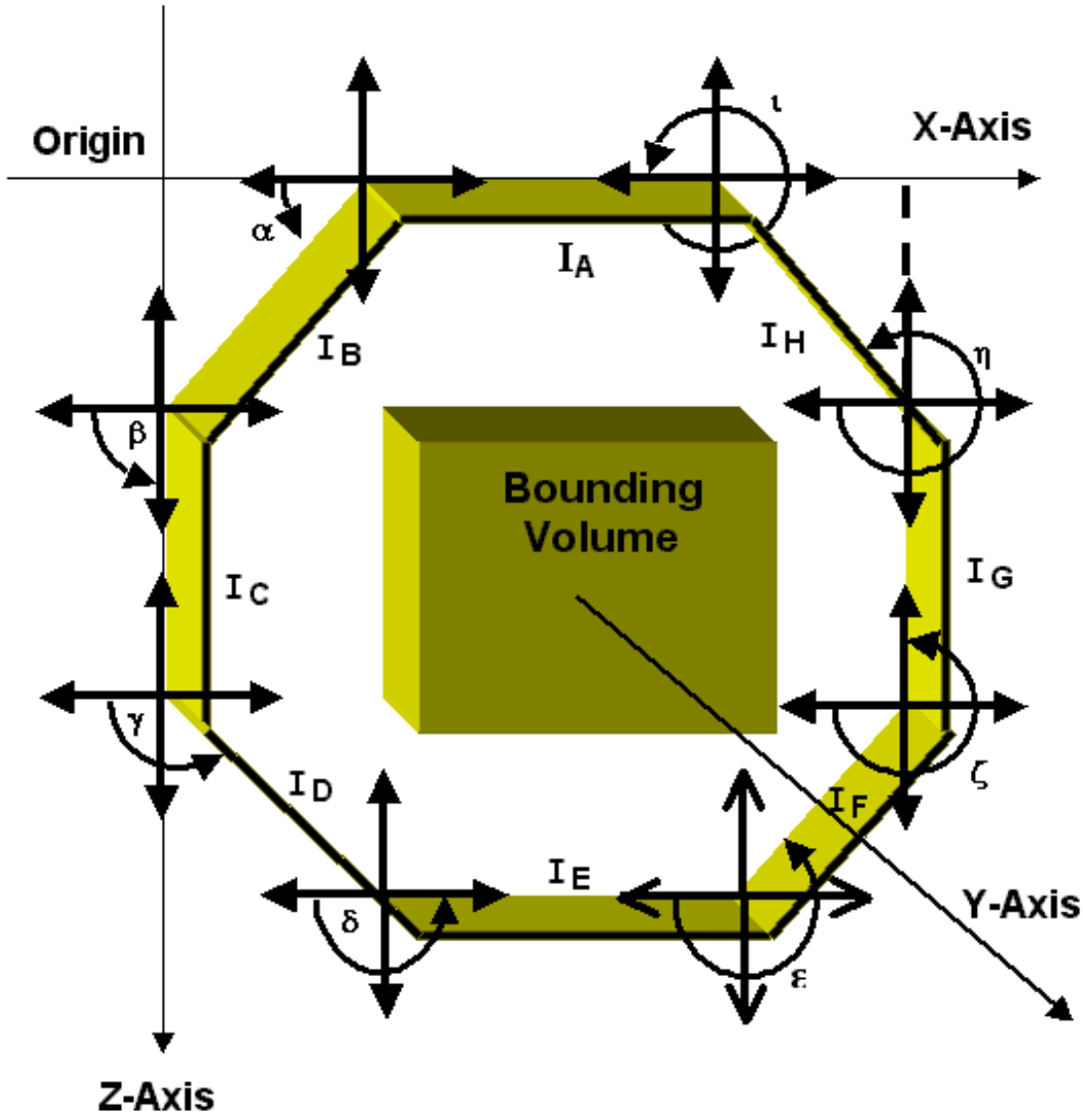


Figure 4.3 Top view of the space carving process.

```
FOR each image at angle θ DO
BEGIN
   FOR each active voxel DO
   BEGIN
      Rotate by angle θ
      Project onto the image plane
      IF voxel maps to background pixel THEN
         deactivate/disable voxel
      ELSE
         voxel's rgb value ← pixel RGB value
      END IF
   END FOR
END FOR
```

where θ represents the angle at which the specific image was taken. The algorithm is illustrated in **Figure 4.3**. In this example, there are eight input images (I_{1-9}). The associated angles α–ι represent the angle of rotation of the object when the specific image was taken. The bounding volume is rotated about its Y-axis for every image that maps to it. The degree of rotation depends on the angle Θ associated with the image. This translates logically into the placement of the images around the bounding volume.

One last thing that must be clarified regarding the basic algorithm is the rotation of the bounding volume mentioned above. In order to ensure correct orthogonal projection of the bounding volume, the rotation of the bounding volume must be performed relative to bounding volume's centre. The bounding volume's centre is translated to the world co-ordinate's centre; it is then rotated by the relevant angle, following which the translation of the volume is reversed.

One might wonder why the angle of rotation is Θ and not –Θ; surely a rotation depicted in an image will involve the rotation of the bounding volume by the inverse angle before space carving? The reason for this is that the bounding volume is assumed to exist in a Euclidean coordinate system. As such goes from coordinate `[0,0,0]` at the bottom of the bounding volume to `[xmax,ymax,zmax]` at the top. Images are typically represented as having pixel (0,0) at the top left of the image and `[xmax,ymax]` at the bottom right. This means that the space carving will result in a reconstruction that is upside down as it is represented in the image. This further implies that a rotation of the bounding volume by Θ will lead to the correct space carving result. This observaiton is crucial to achieving the correct effect with the space carving component.

4.2.1.3 Results

The shape of the object being reconstructed is obtained using this approach. An example of the entire process is illustrated in **Figure 4.4**. Images were taken of a 3D model of a dolphin at different angles, under orthographic perspective conditions. The result of using these images as input for the reconstruction pipeline is the image of the dolphin shown below the others.

Figure 4.5 illustrates the reconstruction of a cube using the space carving component. The image on the right illustrates a cross-section of the reconstructed cube. The reconstruction is composed completely of voxels, each having a dimension 1x1x1. At this size, the cubes representing the voxels are perfectly aligned next to each other giving the effect of a continuous surface. The illustration on the right depicts a cross-section of the cube. It becomes evident that the object is composed entirely of little voxel cubes.

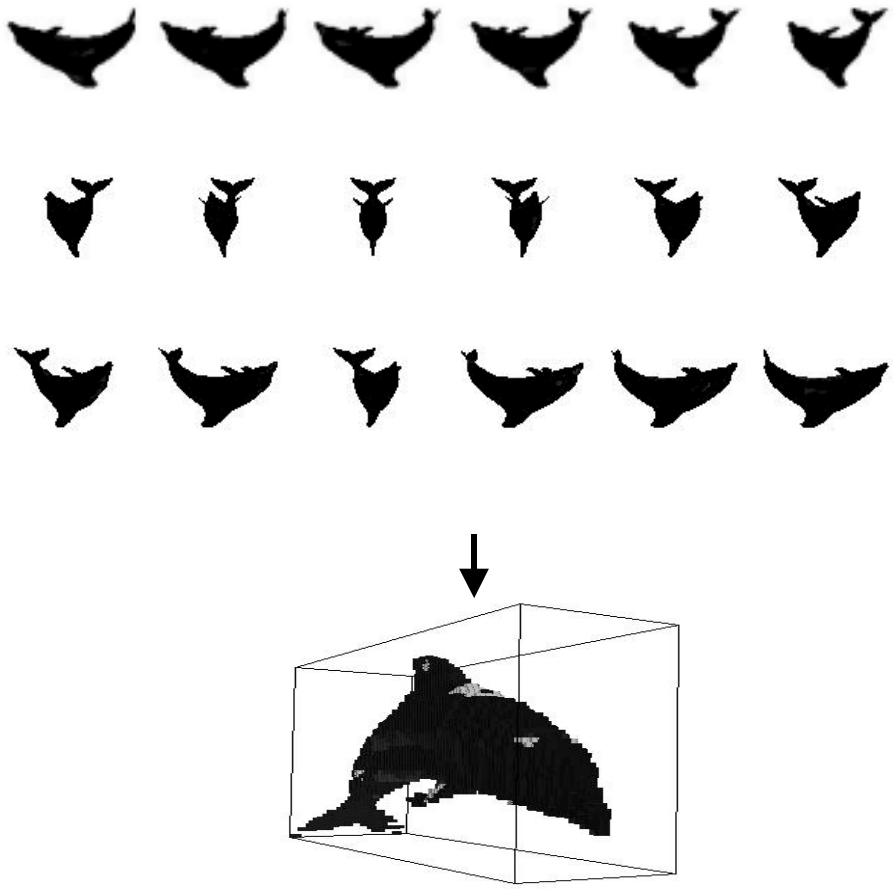


Figure 4.4 The reconstruction of a dolphin.

The space carving algorithm can be thought of as a Boolean intersection algorithm; the resulting reconstruction is the intersection of the parallelepiped extruded by all input images. The outline of the object in every image will ultimately determine the final shape of the reconstruction.

Because the algorithm only evaluates the outline generated by the object in each image, it is unable to reconstruct concavities that form part of the object's surface; the only concavities that can be recovered are those that form part of the object's visual hull.

4.2.2 Isosurface Extraction

Having recovered the basic shape of the object, a hollow shell of voxels representing the surface of the object is recovered through the **isosurface extraction** component.

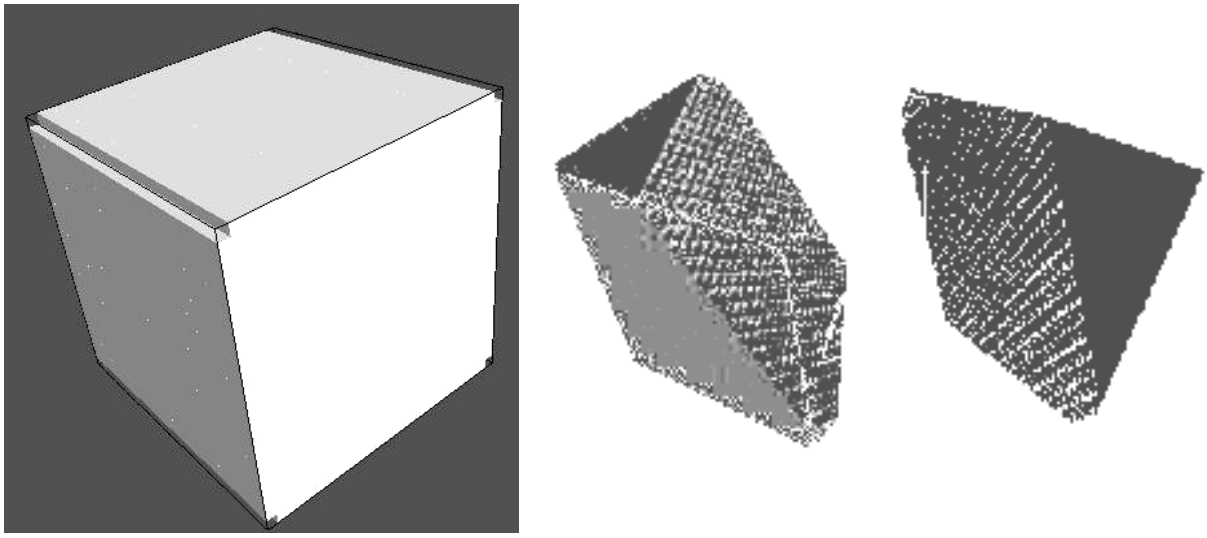


Figure 4.5 The reconstruction of a cube and its cross-section.

4.2.2.1 Design

Many voxels that are part of a reconstructed object can be removed without affecting the shape/structure/appearance of the object, particularly those that can be classified as internal. To remove the internal voxels, an isosurface extraction algorithm is applied to the voxel space. Having evaluated each voxel to determine whether it is internal or not, one is left with a hollow shell of voxels representing the surface of the reconstructed object. The essence of this approach is that of a 3D filter, where the conditional evaluation is based on each voxel's colour.

The result is a reduction of voxels required to represent the reconstruction with no visible degradation in visual quality; the hollow shell reconstruction has the same shape as its space carved counterpart.

Considering the space carving implementation in section 4.2.1.2, it becomes evident that a voxel remains active only if it does not map to a background pixel for the entire space carving process. As such, internal voxels can be identified as those active voxels that are surrounded by active voxels. If a voxel is identified as internal, it becomes inactive. Unfortunately, the comparison of the surrounding active voxels cannot be performed according to their states (active or inactive) because as internal voxels are disabled, they cannot subsequently be used to perform further classifications. As such, the colour the voxel has mapped to during the shape extraction process is used.

If the current voxel is active and none of its neighbours have mapped to a background pixel, then the current voxel can be classified as internal.

4.2.2.2 Implementation

The algorithm is shown below:

```
FOR voxel V at position (X,Y,Z) DO
BEGIN
     IF NEIGHBOURS.colour <> BGCOL AND V = Active THEN
        Disable V
     END IF
END FOR
```

where:

- `NEIGHBOURS` refers to the following list of voxels surrounding `V`:
  - `V[X-1, Y, Z]`
  - `V[X+1, Y, Z]`
  - `V[X, Y-1, Z]`
  - `V[X, Y+1, Z]`
  - `V[X, Y, Z-1]`
  - `V[X, Y, Z+1]`
- `NEIGHBOURS.colour` represents the colour of each of the above voxels; and
- `BGCOL` represents the colour defined to be the background colour.

Figure 4.6 illustrates this algorithm in a two-dimensional example. The voxel represented by

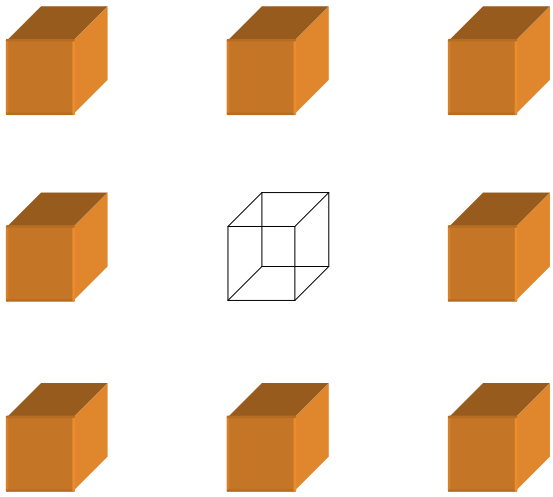


Figure 4.6 The isosurface extraction algorithm in two dimensions.

in wire-frame is removed because it is surrounded by coloured voxels. If a voxel is classified as internal, it is flagged as such. This does not affect the remaining voxels since the comparisons are performed on the voxel colour and not on the status of the voxel (whether it is active or not).

4.2.2.3 Results

Figure 4.7 shows the results of the isosurface extraction algorithm applied to the cube **Figure 4.5**. The isosurface extraction component removes all voxels internal to the carved cube and generates a hollow shell of voxels representing only the surface of the cube. A cross-section of the cube has been performed to show that it is hollow.

4.2.3 Mesh Connection

The next step in the reconstruction pipeline involves by a process of connecting the remaining voxels to generate a closed, connected mesh of the reconstruction. At this point all voxels are conceptually converted to 3D vertices, by taking each voxel centre to be the position of the associated vertex in 3D space.

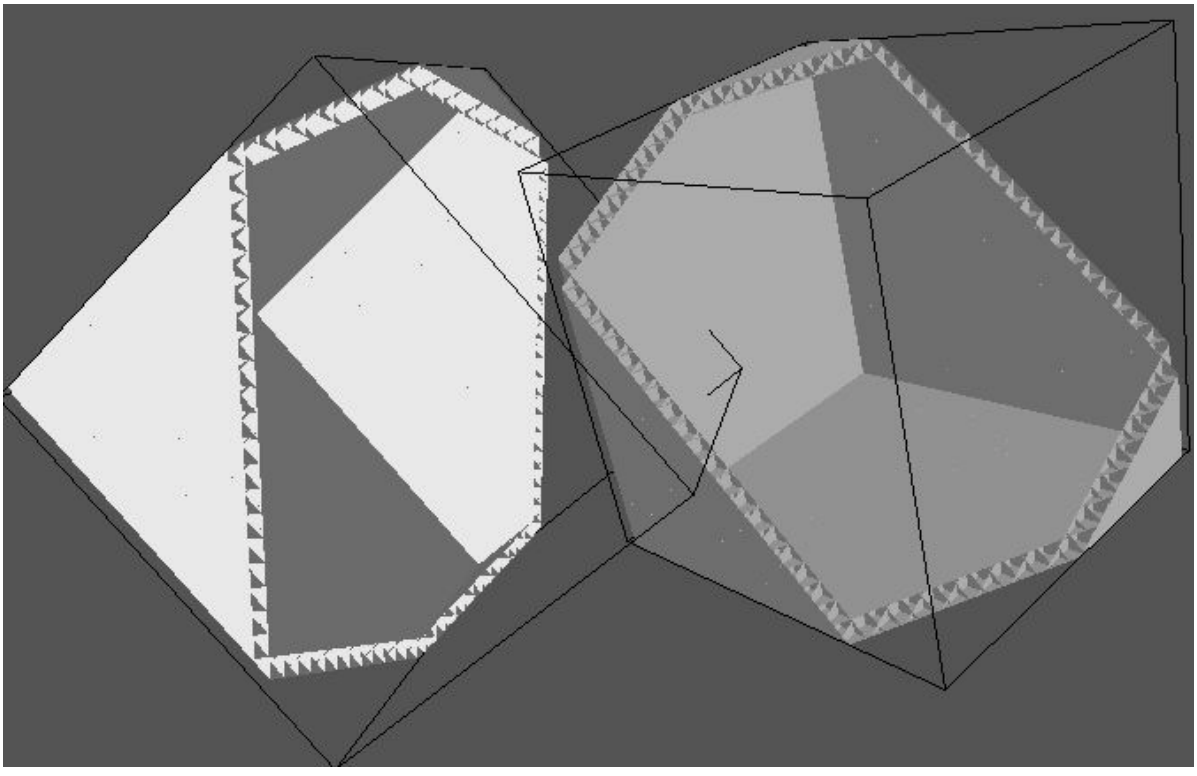


Figure 4.7 Cross-section of a voxel reconstruction after isosurface extraction.

4.2.3.1 Design

Following isosurface extraction, a 3D representation of the remaining voxels must be generated. The algorithm used to perform this generation is based loosely on the marching cubes algorithm (see Schroeder *et al.* [57], pp 146-152 for a full discussion of the marching cubes algorithm), and has the advantages of being a very robust algorithm. This means that if later enhancements are developed, this algorithm remains invariant to these enhancements. An example of this is the mesh decimation component; performing decimation on the reconstruction does not affect the operation of the connection algorithm.

This component is responsible for generating a 3D polyhedral representation of the reconstruction. To that end, an **active vertex** is defined as the centre point of an active voxel. The mesh connection component thus generates a polyhedral mesh by evaluating these vertices.

The algorithm, just like the marching cubes algorithm, evaluates the active vertices two planes at a time. The connectivity algorithm is illustrated in **Figure 4.9**. The vertex (voxel) space is divided into a number of **cells**. A cell is defined as a group of 8 vertices having a cubic structure. Four vertices representing the top of the cube are selected from plane `Z`, and the remaining four vertices are selected from plane `Z+1`. The connectivity algorithm evaluates each of the cells in the vertex space along the Z-axis. The structure of a cell is illustrated in **Figure 4.8**. The cube in

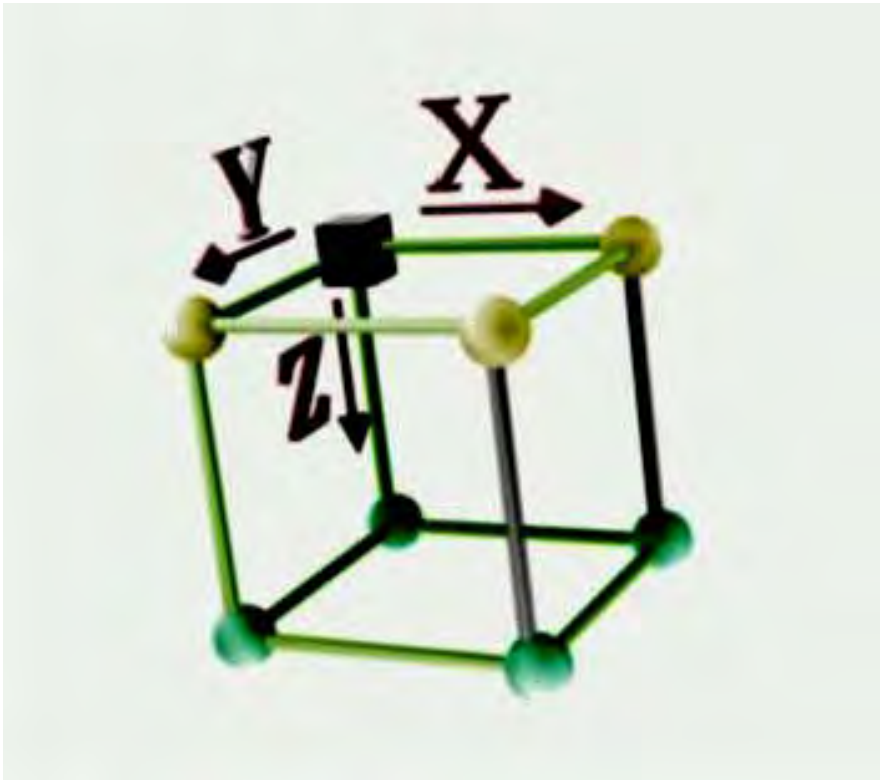


Figure 4.8 The structure of a cell.

the cell structure represents the current vertex being evaluated by the connectivity component. When combined with the remaining vertices (represented as spheres), a cell is formed. The cell illustrated in this image contains the maximum number of vertices allowable in any cell structure (eight). In most situations, there are most likely less than eight vertices in any specific cell.

The final goal of the connection algorithm is the generation of a triangular mesh of the active vertices. Given a cell, a lookup table is used to determine how the vertices present in each cell can be connected to one another to form triangles. Any vertex not present causes certain entries in the lookup table to become invalid. Once each lookup table entry has been evaluated and classified as valid/invalid, a triangle is generated from the information stored in the entry and the lookup table reset for the next cell evaluation.

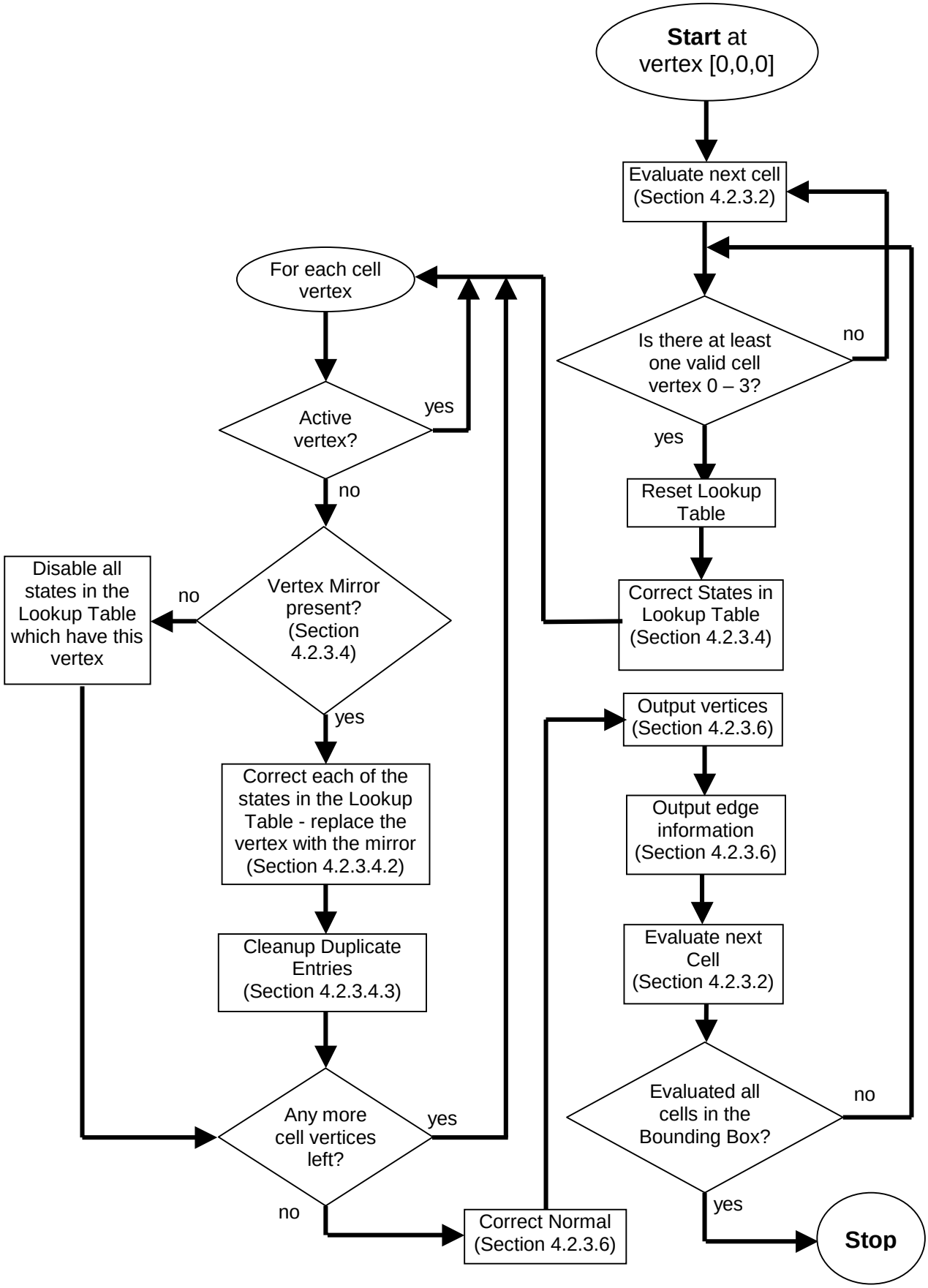


Figure 4.9 A flowchart representing the mesh connection algorithm.

4.2.3.2 Searching for cell vertices

The structure of a cell has been defined. It is important to note that for the rest of the discussion on the connectivity algorithm, **reference assignments** are used. Reference assignments refer to the position of any vertex in the current cell relative to the vertex represented by the cube element that is part of the cell in **Figure 4.8**. Assume that the vertex in **Figure 4.8** represented by a cube exists at position `[10,5,6]`, and all other cell vertices (the spheres) are present. Each present cell vertex has an associated reference assignment, which is used instead of its 3D position. The vertex at position `[10,5,6]` will thus have a reference assignment of 0, with the remaining vertices being given reference assignments as follows:

Cell Reference Assignments					
3D Position	**Example Position**	**Reference Assignment**	**3D Position**	**Example Position**	**Reference Assignment**
`X,Y,Z`	`10,5,6`	`0`	`X+1,Y,Z`	`11,5,6`	`1`
`X,Y+1,Z`	`10,6,6`	`2`	`X+1,Y+1,Z`	`11,6,6`	`3`
`X,Y,Z+1`	`10,5,7`	`4`	`X+1,Y,Z+1`	`11,5,7`	`5`
`X,Y+1,Z+1`	`10,6,7`	`6`	`X+1,Y+1,Z+1`	`11,6,7`	`7`

4.2.3.3 Lookup Table Structure

To put the remainder of this discussion in context it is necessary to realise that the goal is the generation of a closed, connected 3D polyhedral mesh. A polyhedral mesh consists of vertices, edges and faces.

Once the vertices present in the current cell have been identified, a process of determining how they will be connected together must be performed. This is done via a lookup table.

The lookup table consists of 12 entries, each representing a possible valid state. Each state represents a triangular polygon as well as the **side** of the cell to which the triangle belongs. Since a cell has the structure of a cube, with the 8 vertices representing the 8 corners of the cell, each cell also has 6 sides associated with it. The information contained within each state is as follows:

- The 3 vertices that constitute the state, i.e. these are the 3 vertices required for this state to remain valid; if at any point any one of the three vertices is not present, the state becomes invalid;
- The side of the cell cube the triangle represented by this state appears on;
- A flag indicating whether the state is a valid one; this flag is used when the 3D geometry is generated (when the face specification is generated).

A typical entry in this table would thus look as follows:

Lookup Entry	Vertex A	Vertex B	Vertex C	Side Of Cube (P)	State
1	0	2	1	TOP	Active

Vertices A, B, and C refer to the vertices required to be present for the state above to be Active. Additionally, the order of the vertices in the entry (0, 2 and 1) defines the clockwise specification for that face. The importance of the ordering will become apparent later on. The **Side Of Cube** (**P**) identifier defines which side of the cell cube this entry belongs to, and can have one of six values, namely: TOP, BOTTOM, LEFT, RIGHT, FRONT, and BACK, depending on which part of the cube the voxels specified belong to. The lookup entry in the above example requires the presence of voxels 0, 2 and 1. Since the voxels all appear at the top of cell cube, P has a value of TOP.

4.2.3.4 Lookup Table Management

The structure of the lookup table has been defined. Before proceeding, it is necessary to discuss the properties that make the cubic geometry of the cell the most desirable and appropriate structure for the implementation of this connectivity algorithm. This discussion will lead up to an explanation of how the lookup table is managed, i.e. the “Correct States in Lookup Table” node in the flowchart will be explained.

4.2.3.4.1 Properties of a Cube

As mentioned above, each cell is composed of eight voxels. If one inspects the marching cubes algorithm, there are 255 possible cases, i.e. ways in which cell vertices can be connected together. This algorithm reduces the number of cases to 16 due to the inherent symmetrical structure of a cube.

Since the aim is the generation of a closed shape by connecting all active voxels in the bounding volume, this problem can be further reduced to that of connecting all voxels in each of the cells together in every possible way. The maximum number of possible lookup entries that can fully describe all possible ways of connecting the voxels in a cell together is 12. The table below illustrates the entries in the lookup table.

Lookup Table Entries					
Lookup Entry	**Voxel A**	**Voxel B**	**Voxel C**	**Side Of Cube (P)**	**State**
1	0	2	1	TOP	Active
2	2	3	1	TOP	Active
3	0	3	1	TOP	Active
4	2	3	0	TOP	Active
5	4	2	0	LEFT	Active
6	4	6	2	LEFT	Active
7	6	2	0	LEFT	Active
8	4	6	0	LEFT	Active
9	5	0	1	BACK	Active
10	5	4	0	BACK	Active
11	4	0	1	BACK	Active
12	5	4	1	BACK	Active

Two simplifications are made to arrive at this list of cases. These are that:

- there are no states mentioned for the bottom, right and front of the cell, and
- there are no diagonal states.

The lack of states for the bottom of the cube arises from the assumption that the object lies completely within the bounding volume of vertices. Once a cell has been evaluated according to the lookup table, the cell evaluation process proceeds by moving one unit along the X axis. Subsequently, the voxels previously identified as being on the front, become classified as BACK voxels. They are then evaluated accordingly. Similarly, RIGHT and BOTTOM voxels eventually become classified as LEFT and BACK. This is illustrated in **Figure 4.10** that depicts two cells next to each other. The current cell being evaluated is the one on the left. The cell evaluation (with respect to the number of lookup table entries) checks only the back, left and top sides of the current cell for triangle connectivity information. The remaining sides are evaluated as

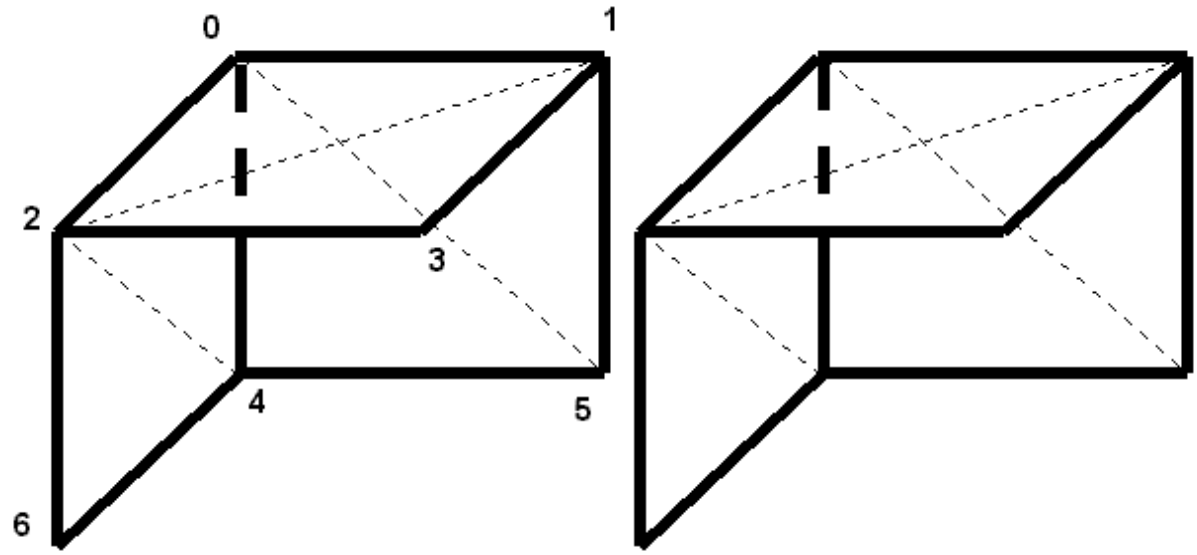


Figure 4.10 The cell evaluation process.

subsequent cells are evaluated. The current FRONT side will become the BACK of the next cell in the evaluation process.

A lack of diagonal state support is described below.

4.2.3.4.2 Diagonal State Support (Mirror Substitution)

Given that a cell vertex is not present, a mirror is searched for and if found, all lookup entries containing that vertex are updated by replacing the vertex with its mirror. Because each lookup table entry has a Side of Cube value associated with it, the mirror to any cell vertex is defined as a cell vertex that lies on the opposite side of the cell to a given vertex's Side of Cube value. The search for a mirror vertex thus involves determining whether an active vertex is present on the orthographically opposite side of the cell.

The definition of a diagonal state is one that specifies the connection of a vertex on one side of the cell cube to one or more vertices on the diagonally opposite side of the cube. The lack of diagonal states arises directly because of the inherent symmetry of a cube. This symmetry means that it is possible to derive every possible diagonal permutation that may arise from the 12 states mentioned above.

The process of mirror substitution refers to the process of trying to find a substitute for a vertex that is not present and then performing the substitution into the relevant states. Mirror substitution is only performed if the substitute vertex is present. This process is illustrated in

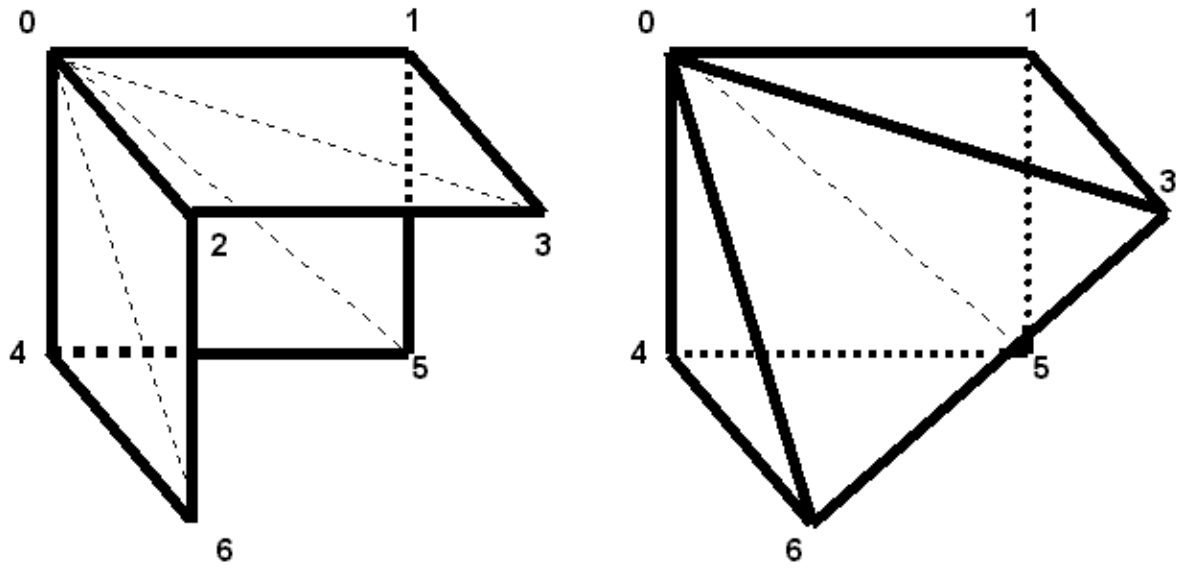


Figure 4.11 The process of Mirror Substitution in action.

Figure 4.11. In this example, vertex 2 is not present in the current cell. Instead of simply disabling each of the entries that require vertex 2, mirror substitution is invoked. In the case of vertex 2, the following are the possible substitutes:

Cube Side	Possible Vertex Substitute
TOP	6
LEFT	3

Mirror substitution is thus dependent on two factors, namely: the position (TOP etc.) of the state relative to the cube and the actual voxel to be substituted. So, with vertex 2 missing from the cell, the following lookup entry alterations will occur:

Cube Side	Original Lookup Entry	New Lookup Entry
TOP	2,3,0	6,3,0
LEFT	6,2,0	6,3,0

A search for substitutes on the opposite side of the cell cube is performed. A vertex that is part of a triangle state at the TOP (vertex 2) of the cube will thus have a substitute at the BOTTOM (vertex 6). Likewise, the following mapping pairs will be enforced during mirror substitution:

```
TOP → BOTTOM
LEFT → RIGHT
FRONT → BACK
```

Therefore, a substitute for a vertex that is part of a triangle on the left hand side of the cube will exist on the right hand side of the cube. All substitutes are searched for on the opposite side of the cube the triangle resides on.

Mirror substitution continues until all the relevant states in the lookup table have been appropriately altered. States are only disabled if a substitute is not present in the cell.

Assume that both vertices 2 and 6 are missing. Mirror substitution will fail in this case; the first state illustrated above (`[2,3,0] TOP → [6,3,0] TOP`) becomes invalid and is removed from the lookup table (the state becomes inactive).

Each of the state entries in the lookup table is dependent on three vertices. The situation may arise that no vertices for a given state are present. The normal process would thus be to apply mirror substitution to all three of these voxels, leading to the possibility of duplicate polygon generation. An original lookup table entry would be as follows:

Lookup Entry	Vertex A	Vertex B	Vertex C	Side Of Cube (P)	State
5	4	2	0	LEFT	Active

With mirror substitution, the entry would look like this:

Lookup Entry	Vertex A	Vertex B	Vertex C	Side Of Cube (P)	State
5	5	3	1	LEFT	Active

The newly altered state in the current cell thus matches the unaltered state 4 in the next cell. This type of situation can be avoided by simply disabling a state if no three voxels for the relevant state are present in the current cell.

4.2.3.4.3 Table Cleanup (Duplicate Removal)

There are two areas where entries in the lookup table may be removed. The one is because of mirror substitution, while the other is inherently part of the connection algorithm.

It may occur that after mirror substitution has been performed, there are various repeating states. To prevent this, the table is checked for duplicate states and if found, they are disabled. An example is provided below to illustrate this situation.

Assume that vertices 2 and 4 are not present. With mirror substitution, the following state changes occur:

Lookup Entry	Vertex A	Vertex B	Vertex C	Side Of Cube (P)
1	0	2	1	TOP
11	4	0	1	BACK

to:

Lookup Entry	Vertex A	Vertex B	Vertex C	Side Of Cube (P)
1	0	6	1	TOP
11	6	0	1	BACK

The final altered states are thus identical. After all the states have been altered (mirror substitution has been applied), duplicates are removed. In the example above, `State 11` would be removed.

The second possible source of duplicates is inherently part of the connection algorithm. If one looks at the structure of a virgin lookup table, as illustrated in section 4.2.3.4.1, the following becomes evident: by dividing the lookup table into groups of 4, one can see that of the entries in each group, the last 2 connect the same group of cell vertices as the first two, but in a different order. To illustrate, the first four entries in the lookup table are repeated here:

Lookup Entry	Vertex A	Vertex B	Vertex C	Side Of Cube (P)	State
1	0	2	1	TOP	Active
2	2	3	1	TOP	Active
3	0	3	1	TOP	Active
4	2	3	0	TOP	Active

The above example is illustrated in **Figure 4.12**. If rendered, these triangles will overlap one another.

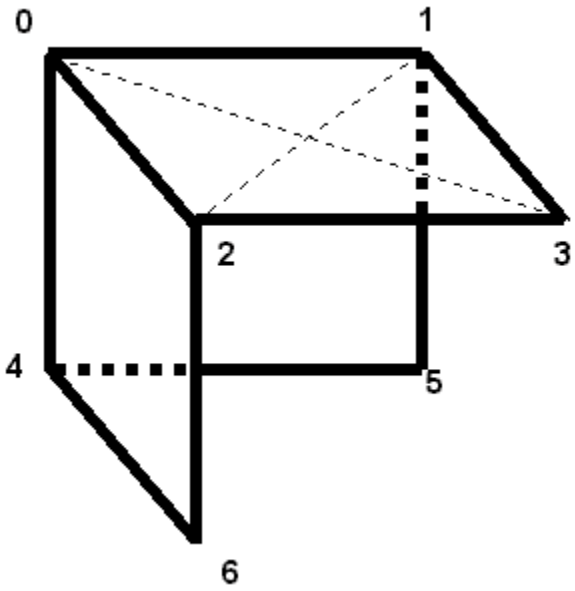


Figure 4.12 Overlapping face specifications are included in the lookup table.

An example illustrating the need for these extraneous state entries in the lookup table is illustrated below.

Assume the following:

- States 3 and 4 above are not included in the lookup table; only states 1 and 2 are present;
- Vertex 2 is not present in the current cell;
- Vertex 2's substitute (vertex 6) is also not present.

Since both states require vertex 2 as part of their specifications, and a substitute for vertex 2 is not present, these states (1 and 2) have to be disabled. This means that even though voxels 0,1 and 3 are present, the lookup table does not support this state. The inclusion of states 3 and 4 in the lookup table would overcome this problem (state 3 directly supports this case).

The inclusion of overlapping cases as described above plays an important role when a relevant vertex's substitute does not exist. If, however, all the vertices of a particular side of the cube are present, it is possible to reduce the number of states by simply disabling the last two entries in every group of four entries. In the above example, if vertices 0,1,2 and 3 are present, states 3 and 4 become unnecessary and are disabled.

4.2.3.5 Polygon Normal Correction

The currently present cell voxels have been identified and the lookup table corrected to accurately reflect the possible triangle pairs that exist for the 8 cell voxels. Before the conversion

of the valid entries in the lookup table to 3D polygon information can occur, it is necessary to perform **polygon normal correction**.

It has been mentioned previously that the order the vertices appear in each of the states in the lookup table is important. The reason for this is a hidden surface removal technique called backface culling. When rendering a 3D model, many graphics subsystems attempt to minimise the number of polygons that must be rendered by evaluating whether a polygon normal is facing the camera; if it is, then the polygon must be rendered; if not, the polygon can be ignored. In order to ensure that all polygons are rendered when they are facing the camera, the vertices representing each polygon must be connected in a clockwise fashion. Each of the states in the lookup table contain references to three voxels which, when connected to one another, will be generated as a polygonal triangle. The important feature of the vertex ordering in each of the states is that it is specified in a clockwise fashion. By pre-specifying the order of the vertex connectivity in this way, it is possible to guarantee clockwise polygon generation.

This pre-ordering is an attempt to automate the polygon generation process. There are situations where the pre-ordering has to be manipulated to achieve the correct ordering. Assume that vertices 4,5,6 and 7 are not present in the current cell i.e. there are no voxels at the bottom of the cell. The cell vertices that are present are used to deal with this situation. Since the final reconstructed object is going to be a closed polyhedral model, it can be assumed that the bottom of the object is being evaluated. As such, the normals of the remaining triangles have to be inverted, i.e. the triangles to be generated from cell vertices at the `TOP` (0,1,2 and 3), must face downwards. The polygon normals must thus reflect this directional change.

A simple way to reverse the polygon normal is to reverse a polygon's vertex ordering. In the case of a triangle (with three vertices), this can be achieved by swapping the first and third vertices. The states for the top triangles would thus undergo the following changes:

Lookup Entry	Original Lookup Entry	New Lookup Entry
1	0,2,1	1,2,0
2	2,3,1	1,3,2

With the ordering of the remaining voxels checked and reversed where necessary, the final step in the connectivity algorithm is to convert the remaining active lookup entry states into a valid polyhedral representation.

4.2.3.6 Polyhedron Generation

The relevant states have been corrected and duplicate states removed. This section describes the process of generating the final polyhedral output.

The polyhedral representation consists of a list of vertices and a list of faces, each face being composed of a list of indices into the list of vertices. Each of the lists is implemented as a linked list. Additionally, each linked list has a number of functions that allow vertices and faces respectively to be added to the linked list, namely:

```
GLOBAL VertList
FUNCTION AddVert RETURNS integer
IN X,Y,Z
```

and

```
GLOBAL FaceList
PROCEDURE AddFace
IN Position[3]
```

The `AddVert()` function adds the vertex with coordinates `(X,Y,Z)` onto the end of the `VertList` linked list and returns its position in the linked list. The `AddFace()` procedure takes as input the list of vertex indexes. Using these interfaces, a face is added onto the end of the FaceList linked list as follows:

```
Position[0] ← AddVert(V[3].x, V[3].y, V[3].z)
Position[1] ← AddVert(V[2].x, V[2].y, V[2].z)
Position[2] ← AddVert(V[1].x, V[1].y, V[1].z)
AddFace (Position)
```

where:

- `V[x]` represents the vertex that must be added to the list; and
- `Position[]` holds the indices for each of the vertices that are added to the linked list.

The is necessary to ensure that the `AddFace` implementation is as efficient as possible, the reason being that the `AddFace` function makes use of the `AddVert` function to make its additions to the `FaceList` linked list. The initial implementation had complexity *O(n!)*, and is replaced by an *O(n)* implementation. The difference arises from the check for duplicates; as a face is added, the entire linked list is checked, thus leading to progressively slower performance as the size of the linked list increases. The assumption that no duplicates are present in the linked

list, speeds up the implementation considerably. The new implementation thus avoids this check completely.

The most important speedup, however, comes about because the mesh connection component generates polyhedral meshes comprised of at least three vertices for each polygon. Using a mechanism whereby the `AddVert` function is called three times for every polygon that is added to the `FaceList` linked list simplifies the addition of the face, but nonetheless slows the implementation down considerably. The enhanced implementation makes use of a caching mechanism; an integer entry (`CACHEPOS`) is added to each voxel's data structure. As vertices are added to `VertList`, so the index values are stored in the relevant `CACHEPOS` entry. Any vertex not having an active associated cache entry essentially means its `CACHEPOS` value has not been modified from its initialisation value. This forces the execution of the `AddVert()` function to determine its index in the linked list. Once returned, the cache entry is updated with the position. If at any time the vertex index is needed again, `CACHEPOS` provides the appropriate information and an `AddVert` call is avoided. The resulting speedup in program execution is remarkable. A 51.6-fold improvement was recorded for the example above.

4.2.3.7 Results

Figure 4.13 illustrates the process of mesh connectivity. The mesh connection component takes as input the hollow shell of vertices (each represented as a voxel cube) generated by the isosurface extraction component and generates a closed connected polyhedral mesh. Illustration Illustrations A (voxel representation) and B (connected reconstruction) depict the cube reconstruction. Illustrations C (voxel representation) and D (connected reconstruction) depict the dolphin reconstruction. In both cases, the connected reconstruction has a more continuous surface, while the voxel representation generates marked discontinuities. This is attributed to the support for diagonal states through mirror substitution. To emphasise this point, illustrations C and D are of the dolphin rendered with smooth shading.

4.2.3.8 Conclusion

A novel mesh connection algorithm has been described. It is robust because it performs an inherent subdivision of the bounding volume space into cells; the connectivity takes place on a per-cell basis. Additionally, it is a very quick algorithm, exhibiting *O(n)* performance. The only constraint on the algorithm is that the bounding volume space be 3D regular.

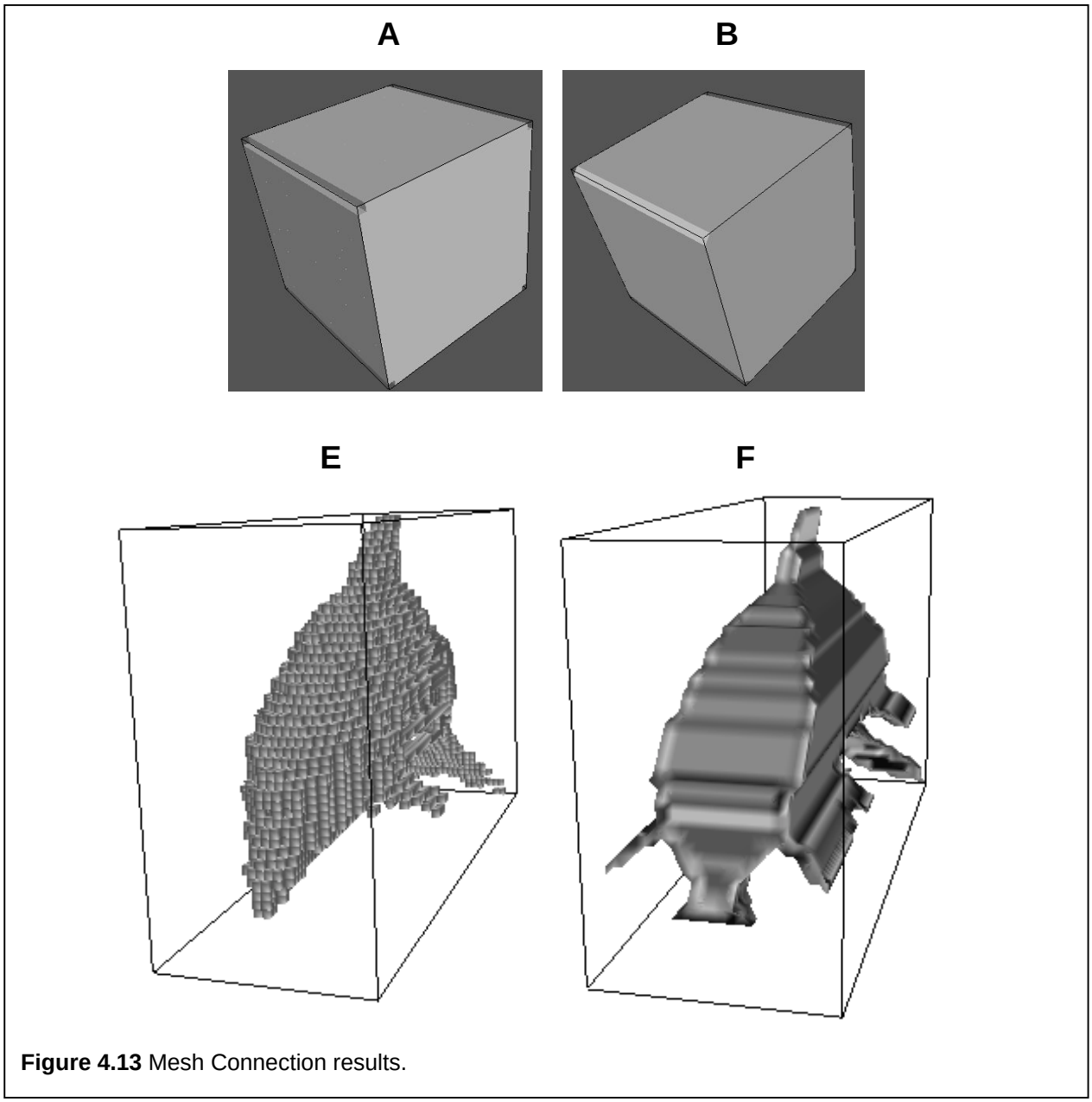


Figure 4.13 Mesh Connection results.

4.2.4 Texture Generation

Once a closed connected mesh has been obtained, it must be textured to generate a realistic 3D reconstruction.

Texture mapping was introduced earlier (section 3.3.3) as a mechanism for generating complex scenes without the explicit need to model them very accurately. Although this notion seems very attractive, it nonetheless suffers from various problems.

The discussion in this section is divided into 3 major parts, namely:

- Explaining the problems associated with traditional texture mapping,
- The approach underlying the texture generation component's implementation; and finally
- A discussion on the benefits and tradeoffs of this component's approach to texture generation.

The texturing technique described here makes use of Gouraud shading to achieve an effect similar to traditional texture mapping. It can be described as a **vertex colouring** method, similar to the voxel colouring method introduced in section 3.3.1.4.

4.2.4.1 Traditional Texture Mapping

The primary benefit traditional texture mapping provides, namely decreased scene complexity, has been mentioned in section 3.3.3. Texture generation with the algorithm presented here is based on a relatively novel approach. It is equivalent to the approach of Seitz *et al.*[58] in that the voxels that remain after the space carving process has been performed are coloured to represent the object's surface. This notion is taken further with the algorithm discussed here. Because the remaining voxels are converted into a closed polyhedral mesh consisting of vertices, use is made of vertex colouring. A vertex colour map of the vertices representing the surface of the reconstructed object is generated. The colour map is then used to Gouraud shade this object. This result is a simulation of traditional texture mapping. The reconstruction pipeline up to this point has resulted in a closed, connected and hollow polyhedral mesh. In addition to the vertices representing the final mesh, the shape carving component ensures that each of the active voxels has an associated RGB value. The RGB value is obtained from the orthogonal projection process it performs. Texture mapping of the reconstructed object can thus be performed without actually resorting to the computational expense of doing this. Experiments by Fourie [25] reveal that texturing causes a drop of 63% in the rendering performance of an SGI Octane Graphics Workstation, or as much as 80% on standard PC workstations that, unlike the SGI machine, do not normally have hardware texture mapping capabilities.

4.2.4.2 Design

The generation of a vertex colour map is at the core of achieving a good textured result. This implies that the algorithm responsible for the storage of RGB values at each voxel needs to be correctly implemented if good texturing results are required. This section describes the different

approaches that have been attempted, and culminates with the development of technique referred to as the Closest Direct-Mapped approach.

The vertex colouring algorithm works in conjunction with the reconstruction process, i.e. as each of the input images are evaluated, voxels are mapped orthogonally to pixels in the images. Each voxel is able to keep track of the RGB value of the pixel to which it has mapped. The basic space carving algorithm discussed previously, also includes the naïve texturing algorithm, as follows:

```
FOR each image at angle θ DO
BEGIN
   FOR each active voxel DO
   BEGIN
     Rotate by angle θ
     Project onto the image plane
     IF voxel maps to white pixel THEN
        deactivate/disable voxel
     ELSE
        voxel's rgb value ← pixel RGB value
     END IF
   END FOR
END FOR
```

The underlined section indicates the texture gathering process. The above algorithm thus means that it is applied as it stands to the bounding volume, the last image used in the space carving algorithm will override any previously stored values, which is undesirable. It is thus necessary to start looking at alternative approaches to overcome this problem.

The solution is an extension on the above algorithm and comes about because of the observation that each voxel in the bounding volume will be closest to only one image during the space carving algorithm.

The bounding volume is rotated about its Y-axis in front of the image plane (as illustrated in **Figure 4.2**). As a result, each voxel being rotated by θ will be closest to only one image. The term closest in this context refers to the distance from the image plane (assumed to lie at position Z=0) to the Z coordinate of the rotated voxel.

Below are descriptions of the different implementations attempted, and the shortcomings of each.

4.2.4.2.1 Direct Mapped

This is the name given to the above algorithm. As mentioned above, this is a crude and unacceptable solution; the last image overrides all subsequent stored RGB values. The mapping of the colours to the voxels happens as the shape is extracted.

4.2.4.2.2 Z-Buffer Mapped

This is an extension of the **direct mapped** algorithm, although colour mapping is performed only after the shape carving process has been performed. Instead of only storing one RGB value at each voxel, all the RGB values that are assigned to each voxel for the duration of the space carving algorithm are recorded; a history of RGB mappings is recorded for each voxel. In order to minimise the memory overhead of doing this, the colour mapping process is separated from the actual visual hull reconstruction process. As such, the colour mapping process only occurs after the isosurface extraction has been performed on the bounding volume. Since the images used to perform the visual hull reconstruction are re-used during the colour mapping process, these images are stored in a texture manager. The texture manager is essentially a lookup table containing the images as well as the angle of rotation associated with each of the images.

At the start of the vertex-colour generation process, the data structure storing the mappings and representing the history is initialised; each active voxel thus has what is called a mapping history. The input to the process is the bounding volume of active voxels only, so the calculation of the size of the data structure can be predetermined. This is where the **z-buffer** part of this algorithm begins. The term z-buffer refers to a 2D array of elements, the dimensions of the array being equivalent to those of the input images. The essential difference between the z-buffer and the image structure is that whereas each item in the image represents a RGB triplet, each element in the z-buffer contains 3D coordinate information, i.e. each element in the z-buffer refers to the position of an active voxel in the bounding volume. Another way of describing this process is that each pixel in the image now has a number of associated vertices, the number of vertices being proportional to the **depth** of the z-buffer.

The algorithm works as follows:

```
FOR each image at angle θ DO
     Initialise the z-buffer to a depth X
     Rotate the bounding volume by θ
```

```
*    Project Bounding Volume to image plane storing the X
     voxels closest to the image plane in the z-buffer

     Record the RGB value for each pixel in the voxels
     specified by the associated z-buffer entry

END FOR
```

where X in the above algorithm refers to the depth specified by the user.

The entry marked with a * in the above algorithm requires more explanation. Each voxel in the rotated bounding volume is mapped to a pixel in the current image. A check is performed to determine if any previous voxel has been mapped to this pixel. If a mapping has previously occurred, the Euclidean distance between the current voxel and the image plane (N) is compared to the distance of the previously mapped voxel (P) to the image plane. If the distance N–P falls within a user-specified threshold, this means that the voxels are close together when viewed from the image plane. This means that only the X closest voxels that map to a pixel are stored in each buffer entry.

The result is that it is added to the z-buffer. A FIFO mechanism is employed to facilitate the filling of the z-buffer for that entry. If the z-buffer is full, the first voxel mapping added to the z-buffer entry is removed to facilitate addition of the latest entry.

The execution of the above algorithm results in a z-buffer that holds crucial colour mapping information. Each entry in the z-buffer maps directly to a pixel, and contains a list of the closest voxels in the rotated bounding volume that map to the pixel. Once this had been done, the z-buffer can be used to determine which voxels in the bounding volume require additional colour entries. As such, the RGB mapping information is stored in mapping history of each voxel listed in the z-buffer for that entry.

The above process is repeated for each image that is part of the texture map. Once the process has been completed, each voxel has a number of colour entries in its mapping history. Because a number of different colours can map to the same voxel from different angles, the process of selecting the final colour needs to be performed. Two statistical approaches that have been attempted are:

- Simple Averaging: traverse the mapping history and determine an average RGB value;
- Standard Deviation: this process is based on the assumption that 66% of a statistically normal population falls within 1 standard deviation of its mean. The standard deviation is calculated for each of R, G and B components separately, (since they are independent of one another), and ignores RGB values whose standard deviations exceed 1. The entries

falling within the standard deviation are then added up and an averaging technique used to determine the final RGB value for the voxel.

4.2.4.2.3 Closest Direct Mapped

This algorithm was finally selected for the generation of the vertex colour map. It works on the same assumption underlying the z-buffer approach, namely that, during space carving, each voxel can be closest to only one image.

The primary problem with the z-buffered approach is one of mapping quantity. The use of a z-buffer arises from the fact that many voxels map to a single pixel in an image, thus the use of a depth buffer during the construction of the z-buffer. Unfortunately, in some situations, many voxels may actually map to the same pixel, but be ignored if the depth of the z-buffer is exceeded. This situation arises particularly if the user-specified threshold is too low or the degree of surface curvature is so high that a large number of voxels map to the same pixel. This many-to-1 relationship highlights the problem with the previous method.

The Closest Direct Mapped approach makes use of the fact that during the evaluation of a single image, only one pixel can map to a single voxel. It attempts to reverse the many–to-1 mapping which exists between vertex and pixel. The result is an intelligent extension to the Direct Mapped approach. The algorithm is listed below.

```
FOR each image at angle θ DO
BEGIN
   FOR each active voxel DO
   BEGIN
     Rotate by angle θ
     Project onto the image plane
     IF voxel maps to background pixel THEN
        deactivate/disable voxel
     ELSE
        IF (voxel has not yet been mapped to a pixel) THEN
        BEGIN
          record current distance from voxel to plane
          voxel's rgb value ← pixel RGB value
        END
        ELSE
           IF (current distance < distance to voxel) THEN
           BEGIN
               record current distance from voxel to plane
               voxel's rgb value ← pixel RGB value
           END IF
        END IF
```

```
        END IF
      END FOR
    END FOR
```

The underlined code segment identifies the addition to the shape extraction algorithm by the inclusion of the Closest Direct Mapped model.

During the extraction of the shape, each voxel is rotated and the orthogonal mapping performed. If this is the first time a mapping is being performed for this voxel, the associated pixel's RGB value and Euclidean distance is recorded. The measurement of distance is based on the following assumptions, namely: the image plane lies in the plane `Z=0`, and the bounding volume is rotated about its local Y-Axis (as illustrated in **Figure 4.3**). A measure of a voxel's distance to the image plane is the Z-component of its centre position. During bounding volume rotation, each voxel moves closer to and further away from the image plane.

If, on the other hand, this voxel has been assigned a colour beforehand, and the distance currently stored in the voxel is greater than the distance of the rotated voxel to the image plane, the RGB value and distance assignment replaces the previous data stored in the voxel entry. Thus, as images are found that are closer to the voxel, not only is the distance recorded, but also the RGB value.

This algorithm thus allows texture information to be gathered as the shape of the object is extracted. The lack of statistical averaging as with the z-buffer approach above arises because of the assumption that the voxel will be closest to only one pixel.

4.2.4.3 Implementation

The process of texture generation follows the same approach as that of polyhedral generation above. As the space carving algorithm proceeds, so a linked list of colours is generated. Additionally, a linked list of mappings from the colours in the linked list to the actual vertices is generated. At the point where the polyhedral data structure output is saved, so the texture generation data structure is saved as well.

4.2.4.4 Result

Figure 4.14 depicts results from the texturing process. Illustration A represents the original object used as input to the reconstruction pipeline. Images of the cube were taken of the object at 20^0 intervals and provided as input. The cube was placed horizontally with respect to the camera.

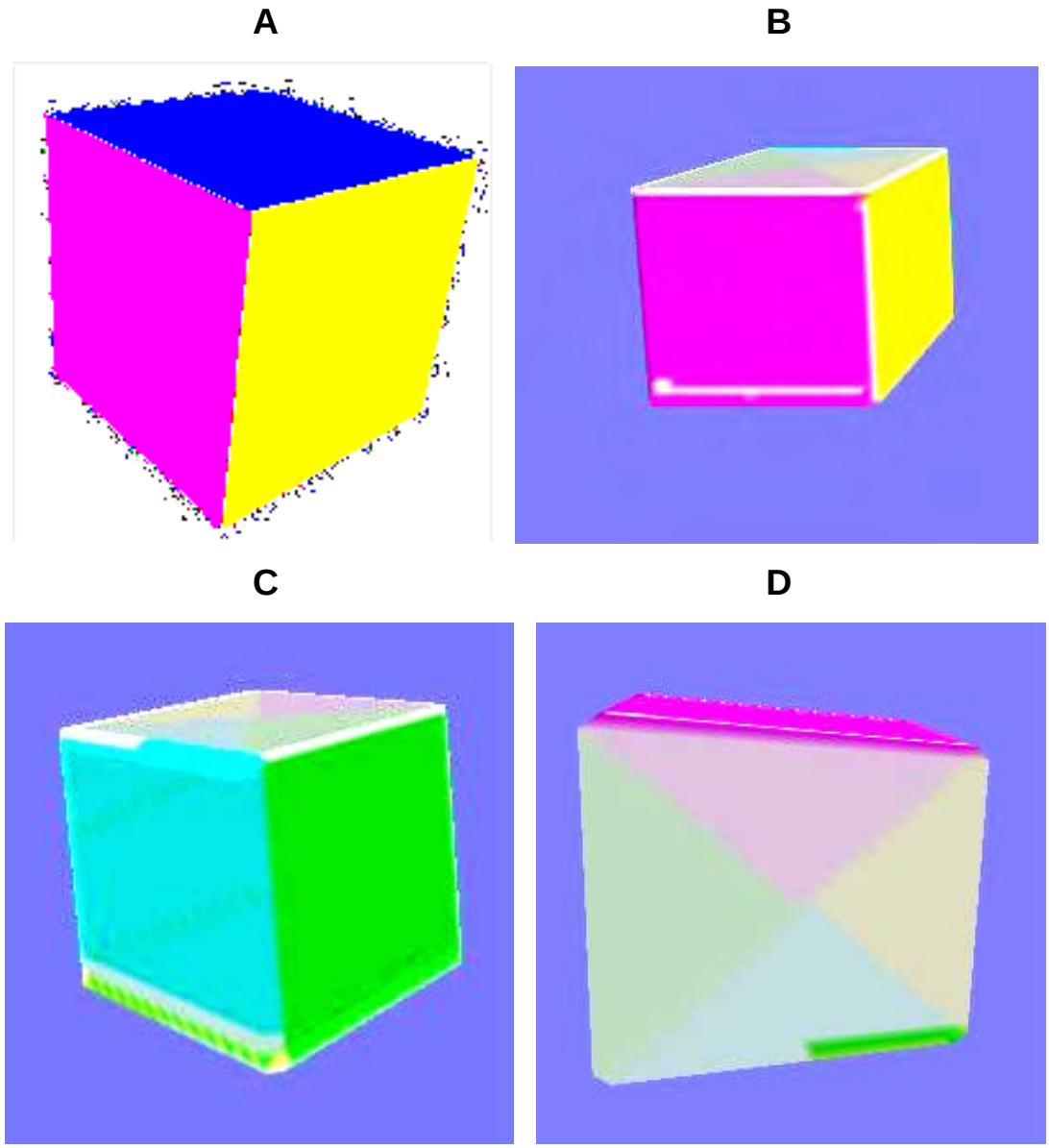


Figure 4.14 Gouraud shaded texturing applied to the cube.

As such, the input images do not depict any of the top part of the cube. Illustrations B, C and D depict the reconstructed cube after the texturing component has generated the associated vertex colour map. As can be seen from B and D, the algorithm is not able to reconstruct the top of the cube at all; the blue colour is missing. The advantage of using a cube allows the texturing algorithm to be tested.

4.2.4.5 Benefits and Trade-offs

This section briefly describes the perceived benefits and problems with the Gouraud shaded texturing. The discussion is based around visual acceptance.

Texture mapping is becoming almost a trivial task. Most IHVs (independent hardware vendors) and manufacturers of 3D accelerator cards provide ever-improving support for traditional texture mapping algorithms in hardware. Most of these implementations are so sophisticated that they are able to perform multi-texturing (applying 2 or more textures to the same polygonal patch and blending the results) in single execution cycles. Visually, texture mapping generates acceptable results most of the time.

There are a number of advantages afforded by the Gouraud shaded approach over traditional implementations. The first is the removal of inverse-perspective calculations. This is traditionally done to ensure that textures do not appear warped when applied to objects. Gouraud shading avoids this because polygon filling occurs at a per-polygon level, thus avoiding the need to maintain visual continuity at a higher level. Traditional texturing approaches have to perform anti-aliasing to ensure surface coherence of the final rendered polygon. Finally, the texturing of curved surfaces is trivial with this approach. Texturing 2D image planes onto 3D curved surfaces is a non-trivial process.

Traditional texture mapping is computationally expensive, while the proposed Gouraud shaded technique requires minimal computational overhead when compared to the traditional texturing approach.. The Gouraud shaded approach is invariant to texture warping and does not suffer from the inherent problems associated with trying to project a 2D plane (the image) onto a 3D surface. Additionally, the approach scales and translates very well, generating none of the artifacts with which traditional algorithms are fraught, particularly due to scaling.

Although the proposed texturing method offers many benefits over traditional approaches, there are a number of problems that remain unsolved, the first being a problem of memory. Many vertices are required to achieve a detailed texture effect. An RGB triplet is required for each vertex in the mesh, thus translating to large memory/bandwidth. The quality of the final rendered scene thus depends greatly on the vertex density (how far the vertices are from one another); a higher vertex density means a superior textured result at the expense of more vertices. The situation imposes processing overhead in terms of vertex processing. This refers to, amongst other things, geometric operations (such as rotation/translation of the polyhedral mesh) and lighting operations.

Secondly, this approach makes it difficult to perform optimisations, such as the decimation approach mentioned above. Special-case implementations have to be added to the base case as soon as any algorithm alters the topology of the model by either removal or addition of vertices; in most situations, however, the biggest problem is the removal of vertices caused by decimation operations.

4.2.4.6 Current Problems

The most prevalent problem with the Closest Direct Mapped texturing solution is that it assumes that the object being reconstructed is situated in the exact centre of the image, and thus in the centre of the bounding volume. Assuming the situation illustrated in **Figure 4.15** exists, where the object lies in the bottom left hand of the bounding volume when viewed from the top, the Closest Direct Mapped texture generation approach will not work, since distance calculations as

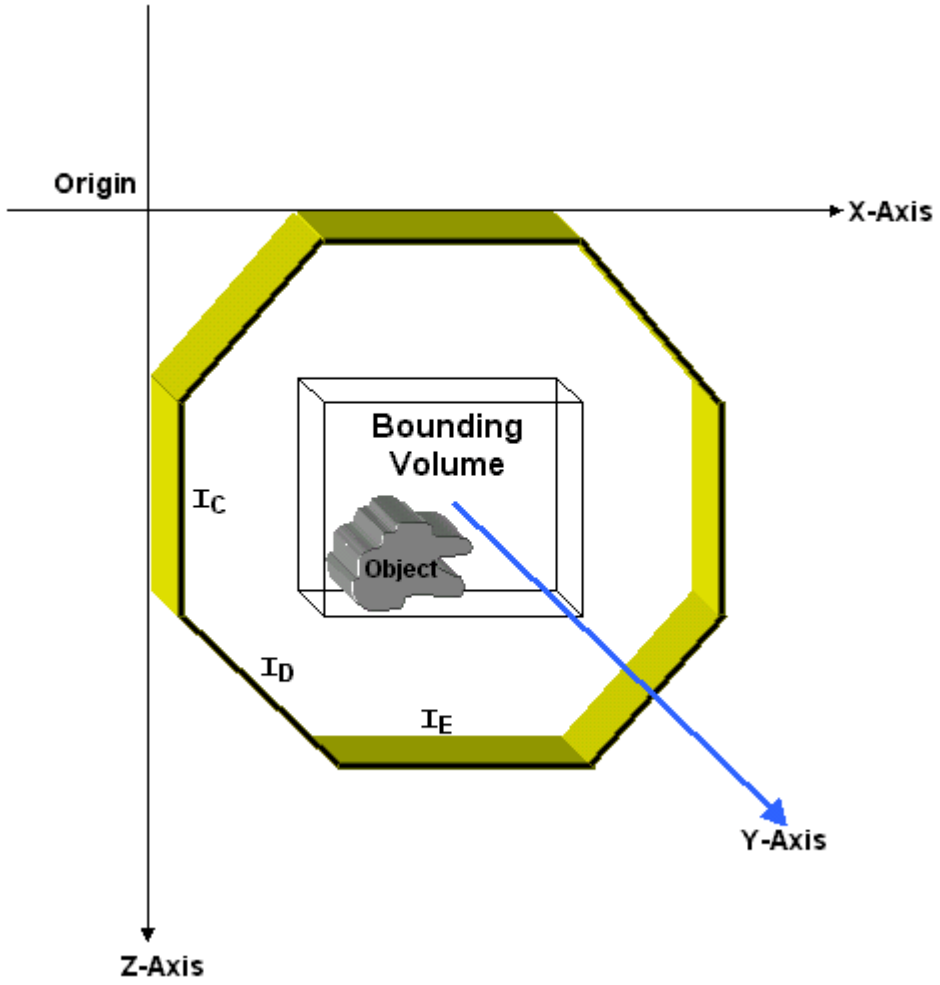


Figure 4.15 Off Y-Axis Object Reconstruction.

performed by the algorithm will generate incorrect results since the object will be closest to image I_C, I_D and I_E. This problem is highlighted by the reconstruction of the cube **Figure 4.14**, where the texture generation is very good. The images used as input to the reconstruction pipeline were taken when the object was in the exact centre of each image. The poor texturing results obtained with the face of the Avatar Austin reconstruction (see **Figure 4.2**) are attributed to this problem.

An example that further illustrates the failure of this algorithm becomes evident if one evaluates the budgie in **Figure 4.16**. The rotation of the budgie occurs relative to the base on which it is perched. The tail is very thin and as such, the situation illustrated in **Figure 4.15** applies to the texture generation of the tail. The entire tail will be closest to a select group of images, namely those containing the black outer side of the tail. The final reconstruction will result in the tail's underside exhibiting the same texture as the black outer side.

4.2.4.7 Summary

A unique and original approach to texture mapping has been developed that unfortunately, given the current state of hardware is unable to compete with traditional texture mapping. The biggest advantage of this method is that, unlike texturing, information is not discarded for every

Figure 4.16 Problematic Texture Generation Subject.

frame that is rendered. This leads to significant speed enhancements.

The advantages offered by this approach are numerous, and a possible dual coexistence with traditional texturing approaches should be investigated. It is envisaged that in areas of high surface curvature, Gouraud shading be employed, while in very planar areas which the decimation algorithm below affects, traditional texturing be utilised. The hope is that a balance between the two techniques will result in faster rendering performance.

4.2.5 Mesh Decimation

Should an optimised reconstruction be required, the **mesh decimation** component reduces the polyhedral representation of the reconstruction.

The introduction of a specialised decimation component that facilitates real-time rendering performance is discussed here. Decimation allows polyhedral meshes to be simplified without much loss of shape. The mesh connection component (in conjunction with the isosurface extraction component) above ensures the generation of a closed shape at the expense of geometrical size. This means that the objects generated by these components tend to be very large; objects with 200 000 polygons are frequent. This might seem like an excessive number of polygons, but this is typical of this class of algorithm; algorithms utilising laser range data also generate such excessive numbers of polygons. The decimation of objects generated by the connectivity algorithm presented above is a necessary step towards generating an object that provides real-time rendering performance on low-end platforms.

The procedure fore decimation here differs slightly from the traditional approach to decimation discussed in section 3.3.4. This approach takes advantage yet again of the fact that vertices are regularly spaced. As mentioned above in the definition of decimation (see section 3.3.4), traditional algorithms have to perform planar calculations for every vertex that is evaluated. These planar calculations may be avoided altogether, since the regularity of the structure forces the vertices in the bounding volume to be in certain fixed positions. This provides a platform for a decimation algorithm, which when implemented, is very quick. This assertion will be explained and expanded on further below.

The mesh decimation approach can be classified as an inverse quadtree algorithm. “The basic principle of a quadtree is to cover a planar region of interest by a square, then recursively partition squares into smaller squares until each square contains a suitably uniform subset of the input” (Eppstein [18]). This definition describes the process of the decimation algorithm, bar the subdivision into smaller squares.

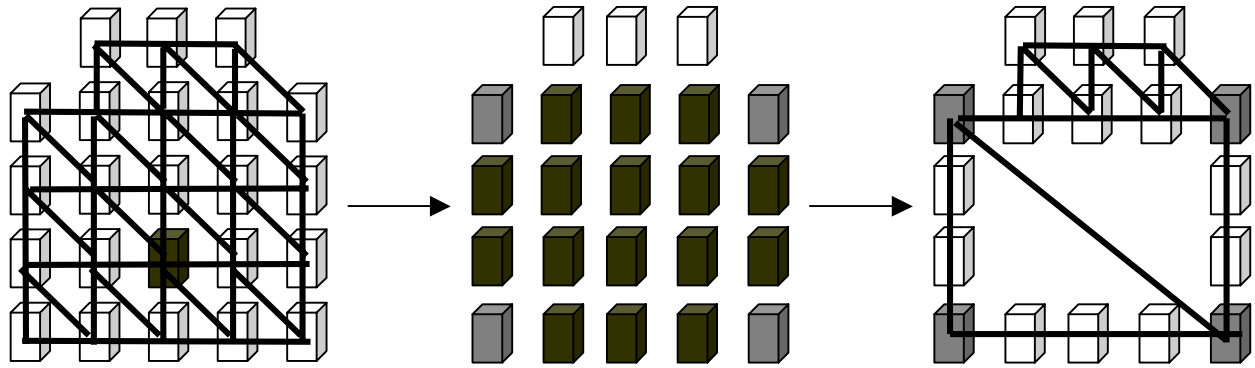


Figure 4.17 The decimation process in two dimensions.

4.2.5.1 Design

The result of the isosurface extraction algorithm is a bounding shell of active vertices representing the surface of the object. The decimation algorithm is applied to these remaining vertices before the mesh connectivity algorithm is performed. For simplicity, the explanation below will only describe decimation in one plane, namely two dimensions..

For every active vertex, determine the largest contiguous rectangle it lies in; remove those active vertices lying within this largest **bounding rectangle**. Repeat this process until all possible bounding rectangles have been evaluated. The progressive removal of active vertices from each bounding rectangle thus prevents a previously classified bounding rectangle from being so classified again. The speed of this algorithm is due in part to this fact.

The algorithm can be described as a bounding rectangle decimation approach and is illustrated in **Figure 4.17**. Beginning with the black seed vertex, the rectangle surrounding it is grown outwards until it has the same dimensions as the rectangle represented by connecting the grey vertices together. The vertices internal to the perimeter are disabled (become invalid) and the result of this is the generation of two triangles representing that rectangle. It must be noted the vertices representing the perimeter of each rectangle remain active; they are not disabled. By doing this, the remaining active vertices are not affected at all.

4.2.5.2 Vertex Classification

The first step in the algorithm is to determine which of the active vertices are **valid active vertices**. The definition of a **valid active vertex** is an active vertex whose neighbours lie in the

same plane as the current vertex. The neighbours are constrained to be active as well. The perimeter formed by these vertices is thus a contiguous collection of active vertices, and is aptly described as a **valid perimeter**. An active vertex at position [A,B,C] is thus classified as a valid active vertex if its neighbours [A-1,B-1,C] to [A+1,B+1,C] are all active.

As can be seen from this example, the evaluation occurs in the XY plane. The generalisation of the algorithm to three dimensions is discussed below. For simplicity of discussion, however, the algorithm is initially only described for one dimension. The decimation algorithm below is thus discussed with respect to decimation of all X-Y patches. The final decimation implementation performs planar evaluations in all three axes, namely X-Y, X-Z, and Y-Z planes. The result is that all possible decimations are applied to the object.

4.2.5.3 Planar Evaluation

Once vertex classification has been performed and the coordinates for the bounding rectangle identified, the next step is to determine the dimensions of the largest bounding rectangle in which the valid active vertex lies. Solving this problem involves the use of a recursive algorithm that has the following structure:

```
FUNCTION checkFunc RETURNS BOOLEAN
INOUT V
BEGIN
   IF perimeterExists (V) THEN
   BEGIN
      T ← V
      IF getMinBoundingRectangle(T) THEN
      BEGIN
         IF checkFunc(T) THEN
         BEGIN
            V ← T
            RETURN TRUE
         END
         ELSE
            RETURN FALSE
         END IF
      END IF
   END IF
   RETURN FALSE
END FUNCTION
```

where:

- V contains the coordinates of the current bounding rectangle;

- `perimeterExists(V)` determines whether the perimeter of the `V` is comprised of active vertices only; if it is, the perimeter is valid;
- `getMinBoundingRectangle(T)` attempts to enlarge the perimeter; if the size has been updated, it returns `TRUE`.

The recursion is dependent on `perimeterExists(V)`. This function checks whether the current bounding rectangle (`V`) has a contiguous perimeter of active vertices; the recursion fails if this function fails. The result of this function is that the parameter (`V`) contains the co-ordinates of the largest bounding rectangle that can be associated to the current active vertex.

Descriptions of the `perimeterExists()` and `getMinBoundingRectangle()` functions are provided below.

4.2.5.3.1 Does a Valid Perimeter Exist? (`perimeterExists`)

The basis for the recursive nature of the decimation algorithm is that successively larger perimeters are evaluated as the recursion deepens; if a perimeter is valid, the recursion proceeds; if not, the recursion stops. In this situation, the search for a larger valid perimeter has failed.

The co-ordinates of the possible bounding rectangle (passed to the `perimeterExists(V)`) are stored in `V` as follows:

Bounding Rectangle Coordinates	
Perimeter Coorinate	**3D Position**
`V[0]`	`minX,minY,Z`
`V[1]`	`maxX,maxY,Z`
`V[2]`	`maxX,maxY,Z`
`V[3]`	`minX,maxY,Z`

with the associated perimeter being defined by the following vertex sequences:

Bounding Rectangle Perimeter Composition		
Side of Perimeter	**Start Position**	**End Position**
AB	V[0]	V[1]
BC	V[1]	V[2]
CD	V[2]	V[3]
DA	V[3]	V[0]

The current bounding rectangle has a valid perimeter if the above vertex sequences only contain valid vertices. The term vertex sequence as used here is defined as: A line segment consisting of only valid active vertices and having a length AB. In the above example, A corresponds to V[0] and B corresponds to V[1].

4.2.5.3.2 Expand the Current Bounding Rectangle (`getMinBoundingRectangle`)

An integral part of the Bounding Rectangle Decimation algorithm is to determine what way the dimensions of the bounding rectangle will alter as the recursion proceeds.

With each call to `getMinBoundingRectangle(t)`, a best-case scenario is assumed at the outset: T (the valid perimeter) will expand diagonally outwards in all directions. Thus:

```
minX ← T[0].x-1
minY ← T[0].y-1
maxX ← T[2].x-1
maxY ← T[2].y-1
```

Since the best-case scenario is just that, a best case, it is necessary to implement an algorithm that supports all the possible situations that may arise. A bounding rectangle may not be able to expand diagonally left. These types of situations have to be catered for. Luckily, the support for all possible cases translates into altering the new minima/maxima (`minX, maxX` etc.) in some way.

Each of the minima/maxima is dependent on certain vertices in the new perimeter defined by the list of minima/maxima below. Upon closer inspection of **Figure 4.1**, the following minima/maxima dependencies can be identified:

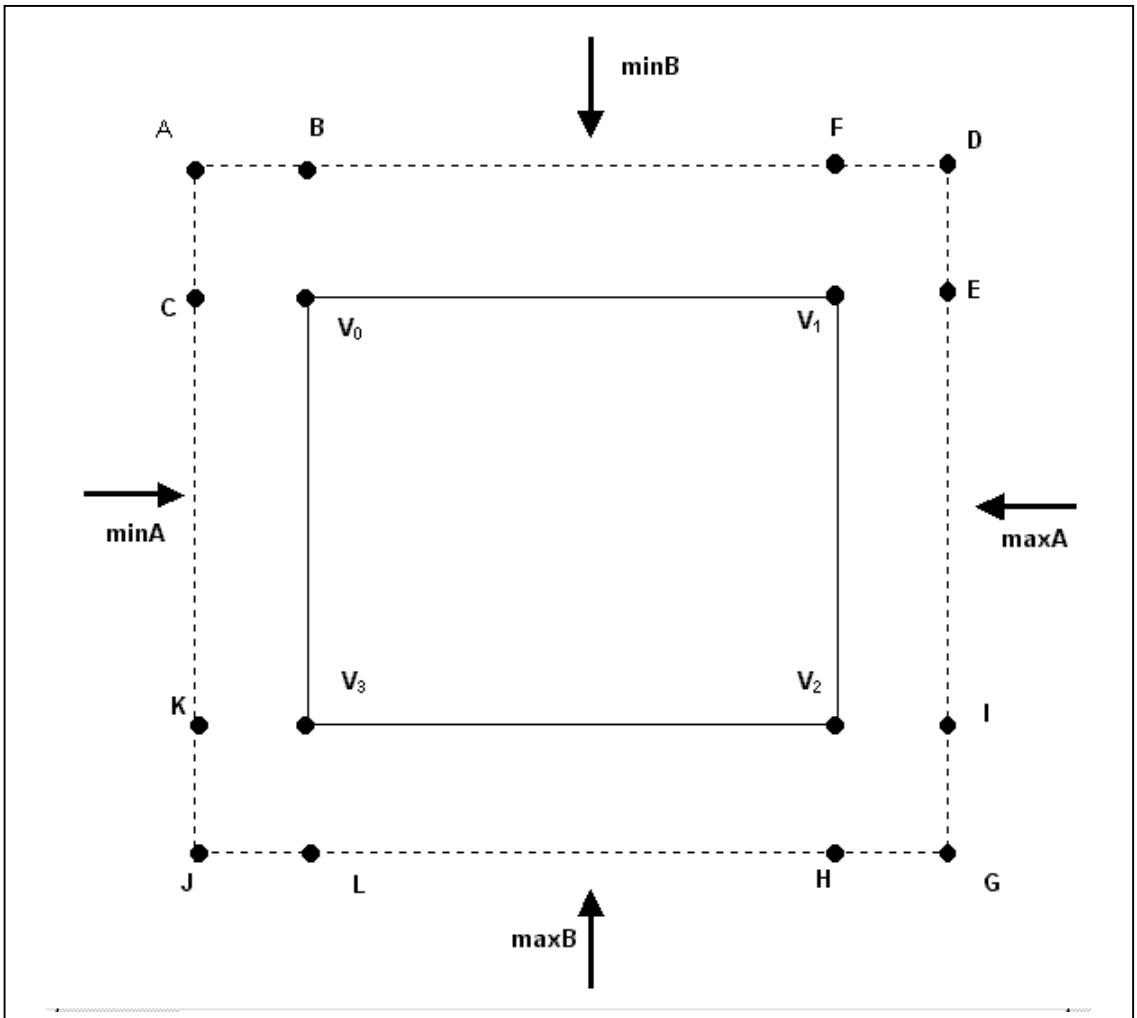


Figure 4.18 Vertices evaluated during the bounding rectangle expansion algorithm.

Bounding Rectangle Dependencies		
Classification	**Group 1**	**Group 2**
`minA (minX)`	`A,C`	`J,K`
`minB (minY)`	`A,B`	`F,D`
`maxA (maxX)`	`D,E`	`I,G`
`maxB (maxY)`	`J,L`	`H,G`

The algorithm can be described by evaluating the expansion for the line segment `minA (minX)`. For the vertical line segment `AJ` (`minA`) to be valid, both vertices `C` and `K` have to be present. If either of these two vertices is not present, then the naïve `minA/minX` defined

previously becomes invalid. The dotted lines in the figure indicate the possible dimensions of the expanded bounding rectangle.

The complete algorithm for the bounding rectangle expansion is provided below:

```
FUNCTION GetMinBoundingRectangle RETURN BOOLEAN
INOUT T

   {Section 1 - naïvely expand the rectangle}

       minA ← CurrMinA ← T[0].x - 1
       minB ← CurrMinB ← T[0].y – 1
       maxA ← CurrMaxA ← T[2].x + 1
       maxB ← CurrMaxB ← T[2].y + 1

   {End Section 1}

   {Section 2 - check base cases for irregularity. The
   rectangle can either get larger or remain the same size.}

       IF B=Inactive OR F=Inactive THEN
          minB ← minB + 1
       END IF
       IF C=Inactive OR K=Inactive THEN
          minA ← minA + 1
       END IF
       IF E=Inactive OR I=Inactive THEN
          maxA ← maxA + 1
       END IF
       IF L=Inactive OR H=Inactive THEN
          maxB ← maxB + 1
       END IF

   {End Section 2}

   {Section 3 - check if each of the new sides is valid. If a
   new side is    not valid, the old coordinates are used.
   This implies that the bounding rectangle will stop growing
   at a side where it is unable to expand any further}

       IF NOT checkSide(minA,minB,minA,maxB) THEN
          minA ← CurrMinA
       END IF
       IF NOT checkSide(minA,minB,maxA,minB) THEN
          minB ← CurrMinB
       END IF
       IF NOT checkSide(maxA,minB,maxA,maxB) THEN
          maxA ← CurrMaxB
       END IF
       IF NOT checkSide(minA,maxB,maxA,maxB) THEN
          maxB ← CurrMaxB
```

```
        END IF

    {End Section 3}

    {Section 4 - did the rectangle get any bigger? If the
    rectangle does get bigger return success }

        IF minA <> CurrMinA OR minB <> CurrMinB OR maxA <>
        CurrMaxA OR maxB <> CurrMaxB THEN
        BEGIN
           T[0] ← minA,minB,Z
           T[1] ← maxA,minB,Z
           T[2] ← maxA,maxB,Z
           T[3] ← minA,maxB,Z
           RETURN TRUE
        END IF

    {End Section 4)

    {Last Resort – Failure}
     RETURN FALSE

  END FUNCTION
```

The algorithm above depicts the bounding rectangle decimation in the XY plane. This can be seen in sections 1 and 4 of the algorithm. `Inactive` evaluations in the above algorithm refer to the state of the vertex, whether it is active or not. The extension to all three planes, namely the inclusion of the XZ and YZ planes is trivial, and will not be discussed further. The bounding rectangle does not grow in areas where it cannot. Given two coordinates, the Boolean function `checkSide()` in the algorithm checks that the line segment formed by these two coordinates is composed entirely of active vertices. If so, the evaluation succeeds and it returns `TRUE`.

If at any point the above function returns `FALSE` because it is unable to determine a bigger rectangle, the failure will bubble up to the `checkFunc()` function, leading to a halt in the recursion. The function will fail if it is unable to determine a new expansion, i.e. if the new bounding rectangle has the same dimensions as the previous one.

4.2.5.3.3 Vertex Removal

The result of running `checkFunc()` is that the parameter passed to it (`V[4]`) contains the four co-ordinates of the bounding rectangle when the function (`checkFunc`) finally terminates.

With this information available, it is possible to decimate the current bounding rectangle using V. The algorithm employed to do this is listed here:

```
FUNCTION disableVertices RETURNS INTEGER
INOUT → V
   K ← V[0].z
   Count ← 0
   FOR I FROM V[0].x+1 TO V[2].x-1 BY 1 DO
   BEGIN
      FOR J FROM V[0].y+1 TO V[2].y-1 BY 1 DO
      BEGIN
         Count ← Count + 1
         BBOX[I][J][K] ← INACTIVE
      END FOR
      RETURN count
   END FOR
END FUNCTION
```

where BBOX[I][J][K] represents the voxel at position (I,J,K) in the bounding volume. The function returns the number of vertices that are removed from the final reconstruction for this rectangle.

The result is that all vertices lying within the perimeter of the bounding rectangle are removed.

4.2.5.4 Geometry Restoration

It is very well to indiscriminately remove vertices from the bounding volume used for reconstruction, but as mentioned in the decimation introduction in section 4.2.5, traditional decimation algorithms also fill the holes generated by the removal of vertices. Holes typically arise when vertices forming part of polygons are removed; the edge information becomes corrupted. Hole filling is performed by re-triangulation of the remaining vertices surrounding the removed vertex; the faces that contain any references to any vertices that have been removed, are themselves removed and re-triangulation is performed. The bounding rectangle decimation algorithm is no different. The re-triangulation in this case is even easier than the traditional re-triangulation, since the bounding rectangle simply translates into two triangles, the coordinates of which can very quickly be determined from the bounding rectangle's coordinates. The following algorithm ensures that the geometry of the original shape is restored:

```
PROCEDURE fillHoles
IN V
IN LinkedList
```

```
    Position[0] ← addVert(LinkedList, V[3].x, V[3].y, V[3].z)
    Position[1] ← addVert(LinkedList, V[2].x, V[2].y, V[2].z)
    Position[2] ← addVert(LinkedList, V[1].x, V[1].y, V[1].z)
    addFace(LinkedList, Position)
    Position[1] ← Position[0]
    Position[0] ← addVert(LinkedList, V[0].x, V[0].y, V[0].z)
    addFace(LinkedList, Position)
END PROCEDURE
```

where the `LinkedList` parameter refers to the linked list containing the 3D polyhedral information of the final reconstruction.

The robustness of the connectivity algorithm has been frequently mentioned previously, specifically in section 4.2.3.8. This robustness becomes evident during bounding rectangle decimation. Applying the decimation without performing geometry restoration has no effect on the connectivity algorithm; it simply proceeds to connect the voxels remaining, albeit with the holes remaining from the removed vertices. The fact that active vertices representing the perimeter are not removed by the decimation algorithm adds an extra level of robustness to the reconstruction process.

4.2.5.5 Results

Figure 4.19 illustrates the mesh decimation component's functionality when applied to a cube and a vase. As can be seen, the voxels representing each side of the cube in illustration A, are replaced by 2 triangles (illustration C). This results in a considerable reduction in the complexity of the model (see the table in the next section for a numerical breakdown of the final polyhedral meshes and savings involved). Additionally it can be seen from illustrations A and B that the cube's surface is represented by a collection of voxels. Illustrations E and F depict voxel representations of the vase, while illustrations G and H show the holes left after decimation has been applied to the object.

Figure 4.20 illustrates the effects of reduction on texturing. Illustrations B and D clearly depict the removal of texture from the sides of the cube, while the areas not affected by the decimation implementation remain unaffected. This highlights the most fundamental problem with the Gouraud shaded texture mapping algorithm, namely that any alteration to the object structure will affect the texturing of that object.

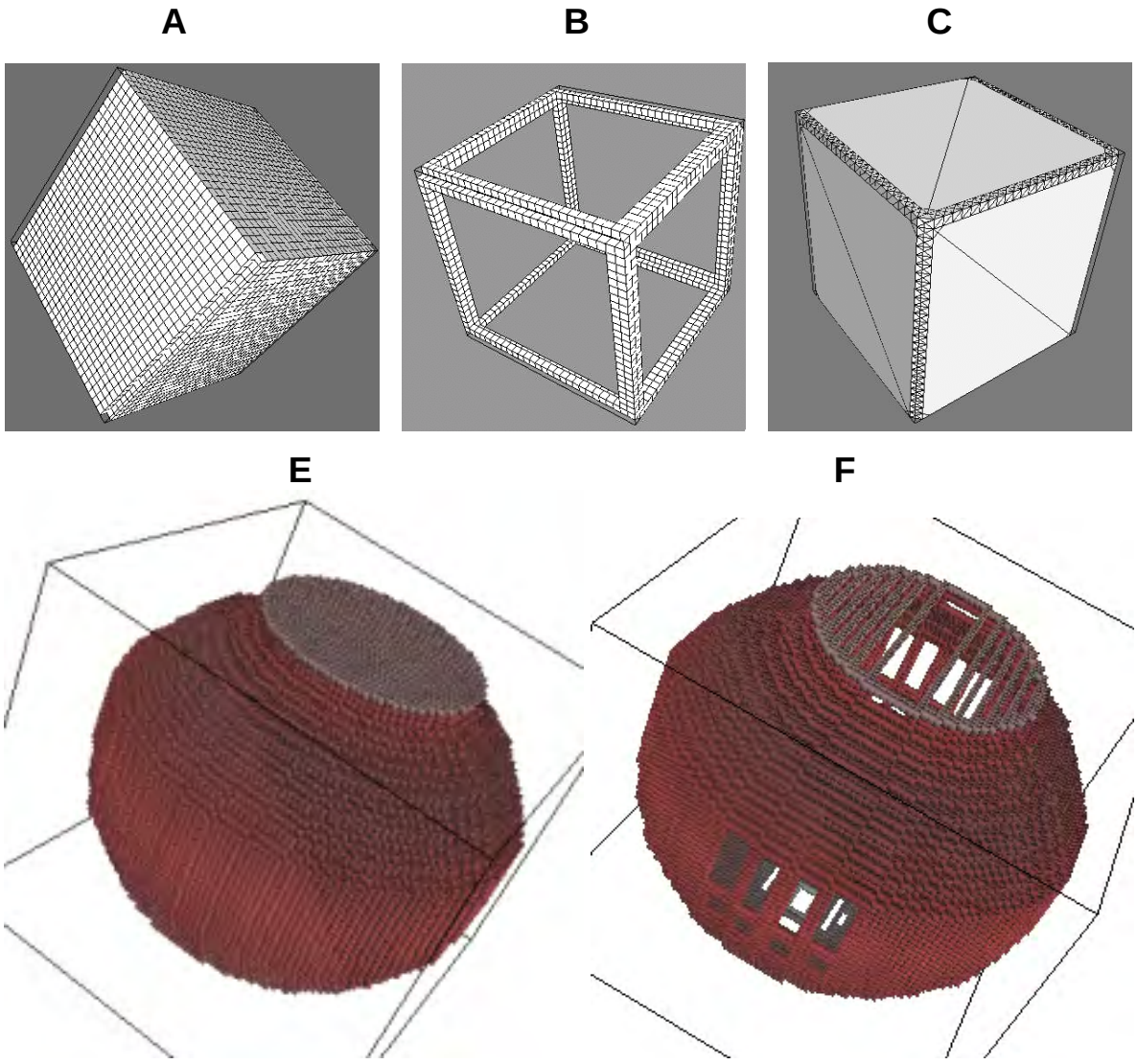


Figure 4.19 Mesh reduction applied to a cube and a vase.

4.2.5.6 Important Issues

The introduction of decimation into this discussion introduces various problems and other considerations; an object is decimated, but at the expense of certain other factors. These factors are briefly highlighted and discussed below. They are listed here as issues to be considered in future improvements of the reconstruction algorithm.

The first problem area that must be highlighted is that of texture generation and how the removal of vertices from the final reconstruction affects the texturing results.

Clearly, the removal of vertices by the decimation algorithm is problematic for this kind of texture generation. A possible solution to this problem lies in the combination of Gouraud-

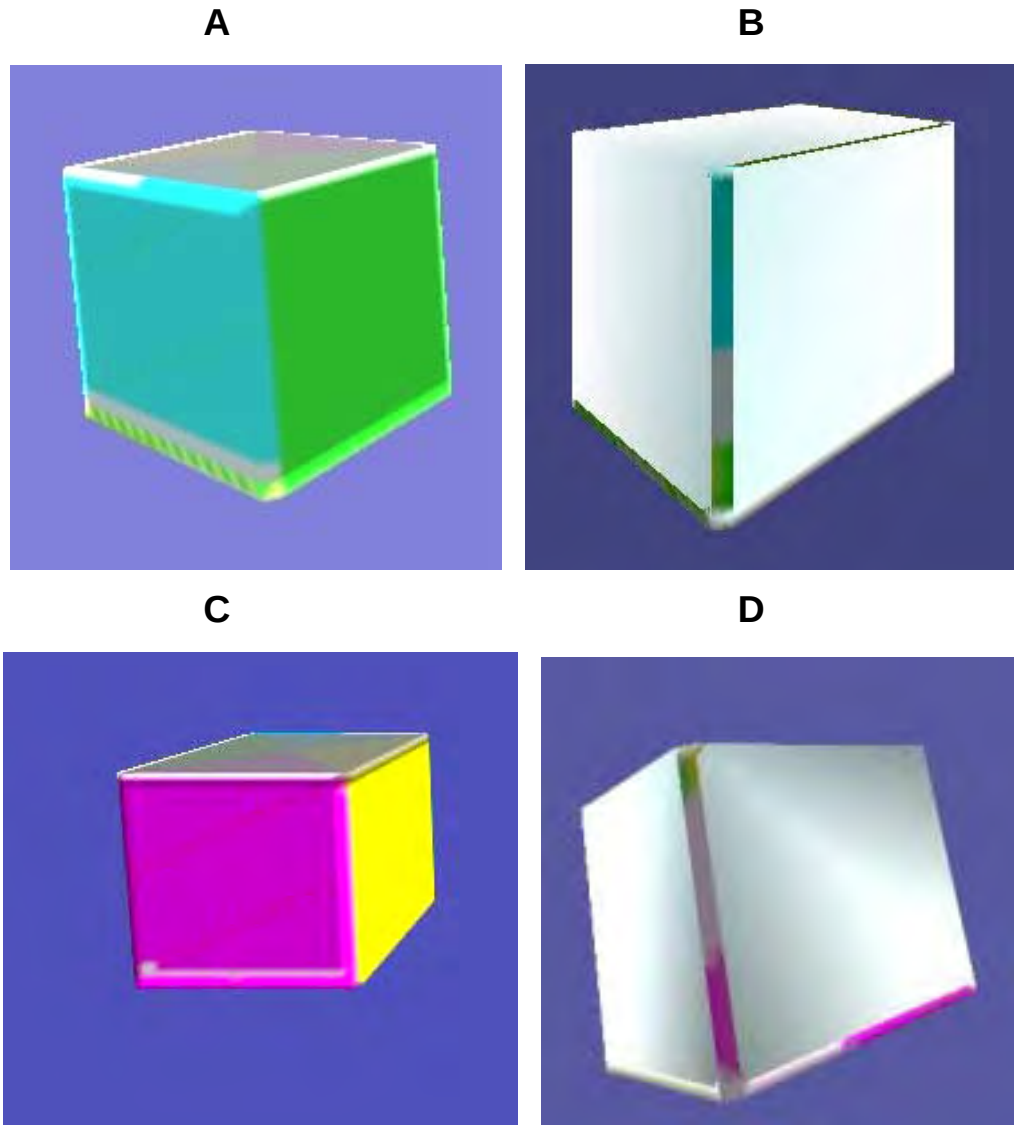


Figure 4.20 The effects of reduction on the Gouraud shaded texturing implementation.

shaded texture mapping and traditional texturing approaches. The idea here is that a traditional texture image can be produced for each of the planes the decimation algorithm affects (attempts to decimate). The texture image would thus be a map of the RGB values stored at each of the vertices within the current plane. This process is repeated for each of the planes identified during the decimation process.

The second factor that has to be considered is the **surface curvature** of the object. The major assumption underpinning the decimation algorithm's implementation is that the bounding volume is a 3D regular volume. The current implementation only attempts to remove voxels that form co-planar rectangular patches, i.e. form rectangular patches in either the XY, XZ, or YZ planes. The effectiveness of this algorithm is obviously dependent on the curvature of the object

being reconstructed. The table below provides an indication of the how surface curvature affects the effectiveness of the decimation algorithm.

Reconstructions with(out) Decimation			
Model	**Vertices Removed**	**Vertex Count**	**Polygon Count**
Austin	-	16382	52456
Austin Decimated	1130	15256	47538
Cube	-	4050	11787
Cube Decimated	3404	646	1639

The timings reported by the decimation application to perform the decimation are as follows:

Decimation Timings (as reported by executable)		
Model	**Bounding Volume Size**	**Reconstruction Time (seconds)**
Austin	100x100x100	9
Cube	50x50x50	0

The timing measurements were obtained using a dual 195 MHz MIPS R10000 processor

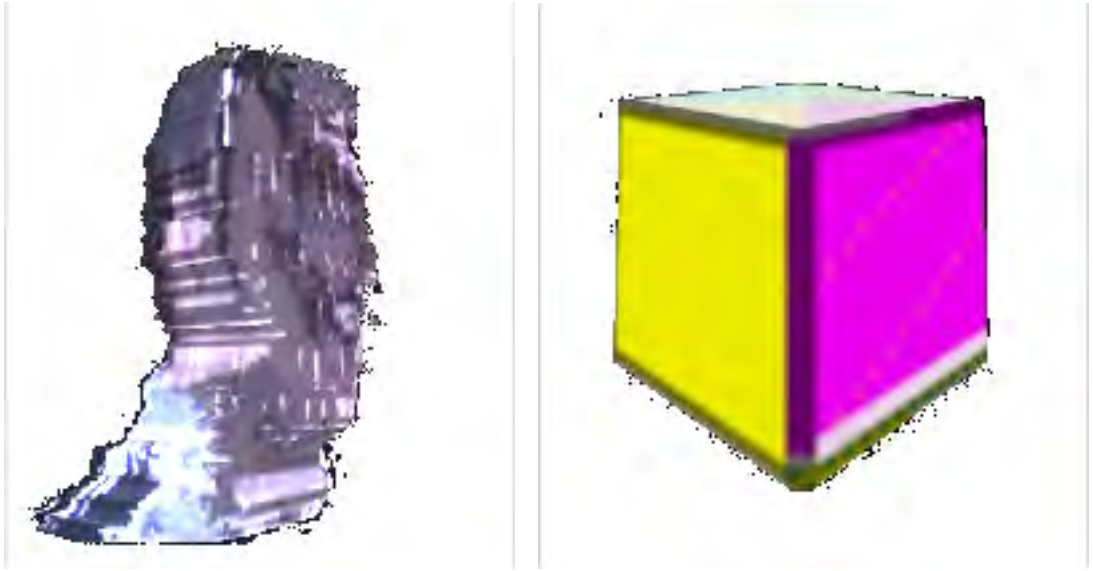


Figure 4.21 Decimation results based on surface curvature.

subsystem running the IRIX 6.4 Operating System. **Figure 4.2** illustrates the two objects used to compose the above table. The objects depicted in the figure have been chosen explicitly to highlight the effect surface curvature has on the decimation algorithm. This is further emphasised by the table, where the decimation algorithm manages to remove more than 84% of the vertices in the original cube, but only approximately 8% of the vertices representing Avatar Austin's head. One last point to mention about the table above is the timing information. The current decimation implementation adds negligible overhead to the overall reconstruction time. This statement is justified, since there was no reported time increase in the generation of the decimated object over the original version in either Avatar Austin or the cube. The Austin and Austin Decimated reconstructions both took 9 seconds. The results were obtained using the same configuration as the one mentioned for the table above.

4.2.6 Reconstruction Results

4.2.6.1 Avatar Model Acquisition

Figure 4.22 depicts the results of the reconstruction pipeline for the purposes of avatar generation. The bounding volume used had dimensions 150x150x150. The reconstruction is very good, except for the face as depicted in illustration C. It is suspected that because the space carving algorithm is based on a visual hull–type approach and concave surfaces are not supported, voxels that are not actually part of the surface are classified so nonetheless. The result is a shrink-wrapped effect, whereby the concave parts of the face are not reconstructed at all. Additionally, the texture tends to suffer as well.

One other possible reason for the bad texturing results in illustration C could be the granularity of rotations; the images of this person used as input to the reconstruction pipeline were taken roughly at 30^0 intervals. It is possible that smaller angles are required before better results can be achieved.

4.2.6.2 Bounding Volume Resolution

The most important factor affecting the quality of the final reconstruction is the size of the bounding volume. The table below illustrates the results of applying the reconstruction pipeline to the same object, but varying the size of the bounding volume.

Reconstruction Parameters				
Bounding Volume Dimensions	**Vertices**	**Polygons**	**Colour Map Size**	**Rendering Timings (fps)**
50x50x50	1260	3683	624	62
100x100x100	16386	52456	5586	6.8
150x150x150	66705	225081	13054	1.95

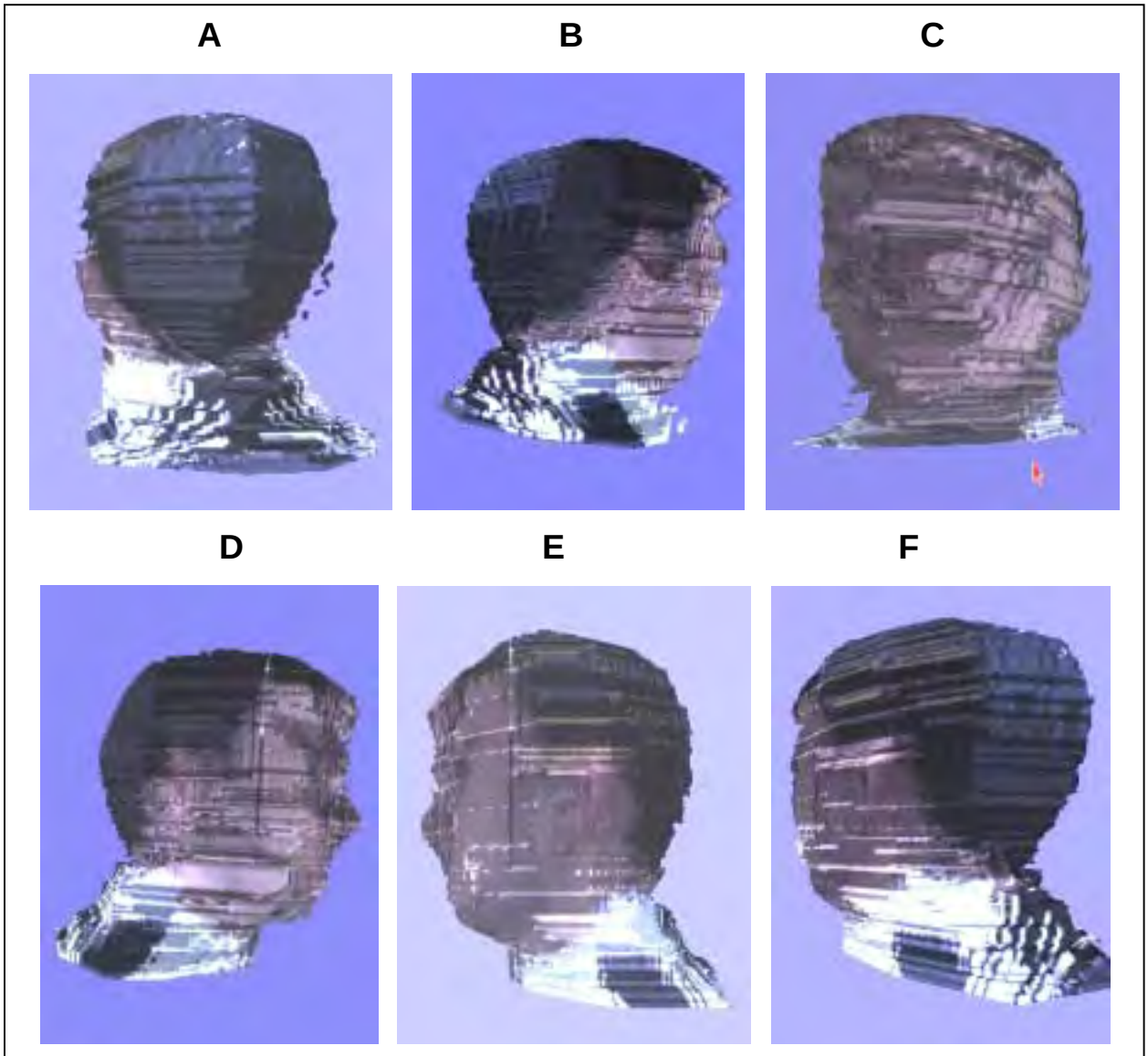


Figure 4.22 The reconstruction pipeline applied to a real-world subject.

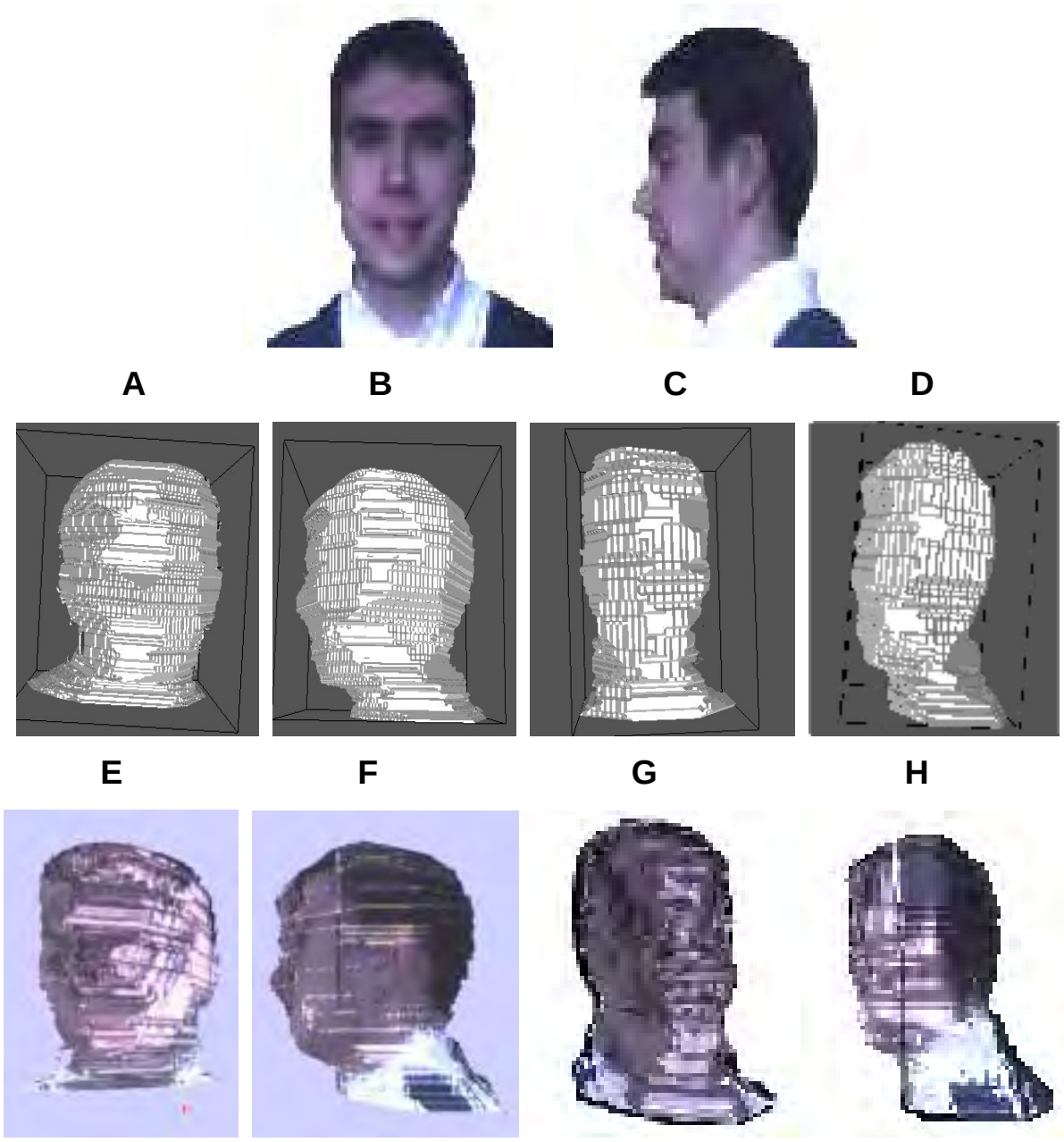


Figure 4.23 Multi-resolution Avatar Reconstruction Results.

As the resolution of the bounding volume decreases, so the resulting reconstruction quality becomes progressively worse. This is illustrated in **Figure 4.23**, representing the avatar reconstruction at a resolution of 150x150x150 (illustrations A, B, E and F) and 100x100x100 (illustrations C, D, G and H). Illustrations A-D depict the structure of the reconstruction without texturing, while illustrations E-F depict the final textured model. The reconstruction of a budgie, illustrated in **Figure 4.24**, also shows how an increase in the resolution of the bounding volume not only affects the structure of the final reconstruction, but also the quality of the final texture.

The budgie consisted of 36 input images containing the budgie, taken at 10^0 intervals. Reconstructions of the budgie at different resolutions are depicted. Each of the three rows

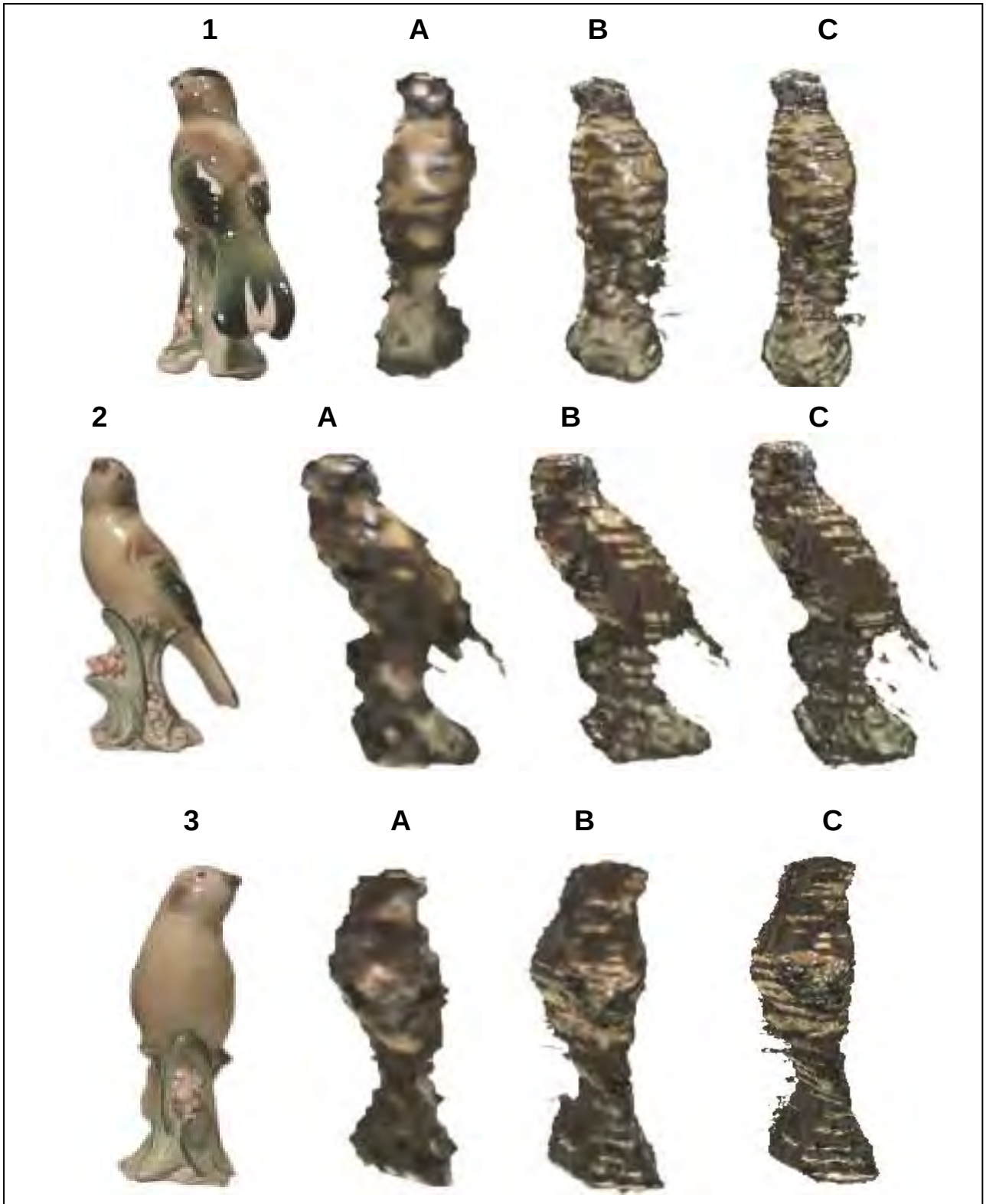


Figure 4.24 Multi-resolution Budgie Reconstruction.

represents: the budgie at a certain angle, a 50x50x50 reconstruction (A), a 100x100x100 reconstruction (B) and a 150x150x150 reconstruction (C), each with the reconstructed budgie in approximately the same position as the real budgie. It becomes evident that as the reconstruction resolution increases, the budgie takes on a better form and the texture becomes sharper and more detailed.

4.2.6.3 Performance Measurements

This section discusses performance issues relating to the reconstruction pipeline and evaluates the overhead, if any, that each component of the pipeline adds to the time taken to generate a reconstruction.

Reconstruction Pipeline Results									
Model	**Res**	**Tex**	**Red**	**Vert**	**Poly**	**Cols**	**VR**	**CT/[s]**	**TT/[s]**
Vase	50	N	N	3034	10491	-	-	0	4
Vase	50	Y	N	3304	10491	142	-	0	4
Vase	50	Y	Y	2607	8828	?	427	0	4
Vase	100	N	N	12982	46221	-	-	3	31
Vase	100	Y	N	12982	46221	404	-	3	32
Vase	100	Y	Y	11334	38939	?	1648	3	32
Vase	150	N	N	12982	46221	-	-	11	107
Vase	150	Y	N	12982	46221	659	-	11	107
Vase	150	Y	Y	11334	38939	659	1648	11	107
Budgie	50	N	N	3766	27246	-	-	0	4
Budgie	50	Y	N	3766	27246	1245	-	0	4
Budgie	50	Y	Y	1384	5202	1062	2382	0	4
Budgie	100	N	N	6589	24421	-	-	2	34
Budgie	100	Y	N	6589	24421	4310	-	4	35
Budgie	100	Y	Y	6267	22689	4175	322	4	35
Budgie	150	N	N	16644	63194	-	-	20	125
Budgie	150	Y	N	16644	63194	9153	-	20	126
Budgie	150	Y	Y	15481	56439	8774	1163	20	124

The table above lists budgie and vase reconstruction timings and model information. The `Budgie` information refers to the budgie reconstruction depicted in **Figure 4.24**, while the `Vase` information refers to the vase reconstruction depicted in **Figure 4.19**. It consists of the following information:

- Res: The bounding volume **resolution** used; a value of 50, for example, implies that the bounding volume resolution is 50x50x50;
- Tex: specifies whether a **texture** was generated (**Y**es) or not (**N**o);
- Red: specified whether **reduction** was enabled (**Y**es/**N**o);
- Vert/Poly/Cols: the number of **vertices**, **polygons**, and **colours** constituting the final reconstruction;
- VR: the number of **vertices removed** by the decimation component;
- CT: the **time taken** by the **mesh connection** component (times are in seconds);
- TT: the **total time** taken to generate the final reconstruction.

A "-" in any cell above implies that no measurement exists in that context. The TT column does not include the time taken to allocate the appropriate memory to the bounding volume initially. The results depicted have been obtained using a 333 MHz Intel Pentium II with 256 MB of RAM. The operating system used was RedHat Linux 6.1.

As can be seen from the table, the difference in reconstruction time between an unreduced untextured model and a reduced textured model is often negligible. In some cases, such as the budgie reconstruction with 150x150x150 bounding volume resolution, the total time taken fell by 2 seconds. This is attributed to the reduction that has been applied to the reconstruction in this case. Additionally, one can see that texture generation has a negligible effect on all cases.

The complexity of the algorithm appears to be linear with the number of voxels, as illustrated in **Chart 4.1**. This chart depicts the textured, unreduced times recorded for the reconstruction of the budgie object in the above table (at resolutions of 50x50x50, 75x75x75, 100x100x100, 125x125x125, 150x150x150 and 175x175x175 respectively).

4.2.6.4 General Comments

Because it is assumed that every voxel is initially part of the object, hole filling as a post-processing task is avoided. Furthermore, the final reconstruction does not depend on surface colour. Light Illuminated Triangulation techniques that typically use red laser light suffer from this problem when trying to reconstruct black or red surfaces.

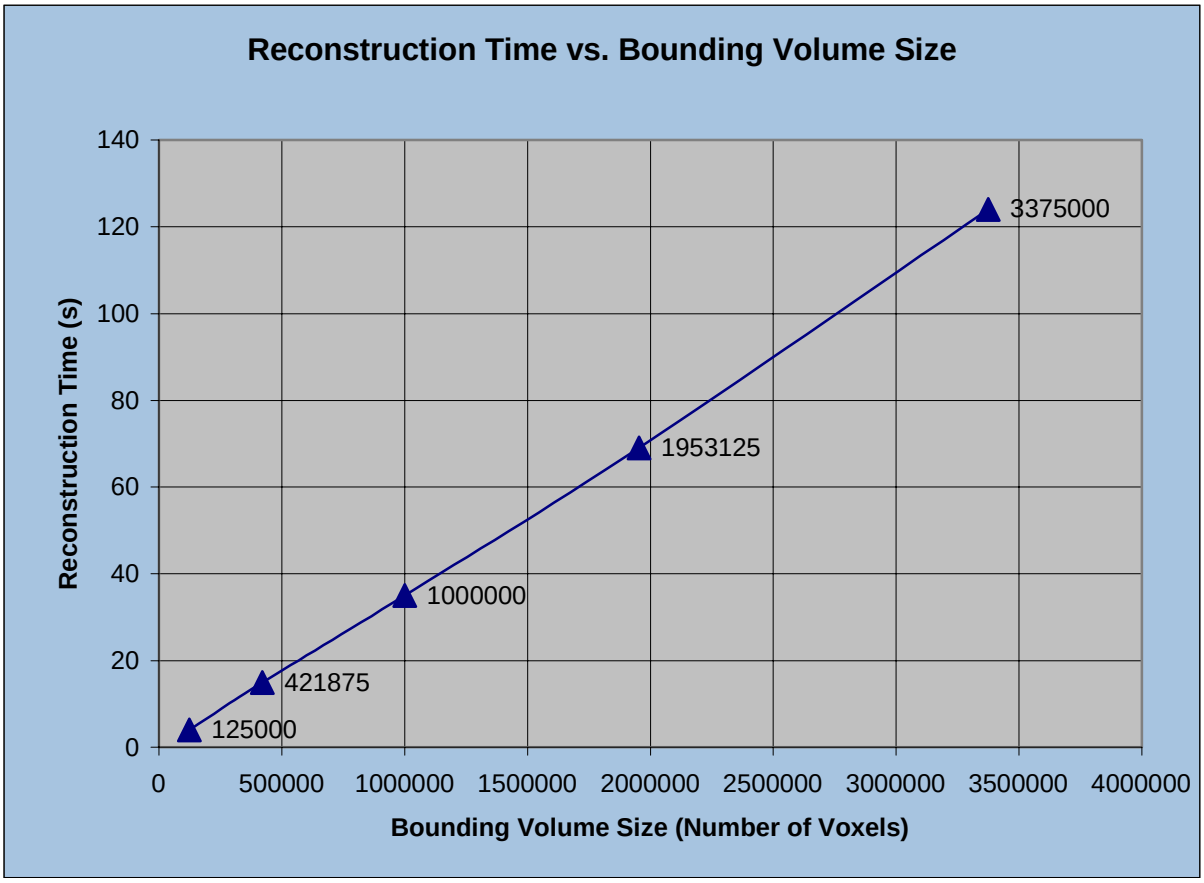


Chart 4.1 Reconstruction Time vs. Bounding Volume Size.

The reconstruction of the budgie in **Figure 4.24** is suffers in areas such as the tail. This problem is attributed to perspective effects of the CCD camera used as the object is rotated. As the tail moves further away from the camera during rotation, so it becomes smaller, thus leading to reconstruction errors. One possible solution would be to take pictures of the object using a telephoto lens to minimise perspective distortions. Additionally, Thalmann *et al.* [64] suggest placing the object relatively far away from the camera; the distance between the camera and the object should be at least four times the maximum width of the object.

This concludes the discussion on the reconstruction pipeline. The selection of this algorithm over traditional facial modelling approaches, such as the profile fitting methods, is well suited to the reconstruction of human heads, partly because of their ellipsoid shape. The final pipeline has an added aspect of generality in that it is able to handle the reconstruction of any object it is presented with, and not only human heads, using an inexpensive CCD camera.

4.3 Depth estimation/recovery

This section describes a novel approach to overcome the most fundamental flaw with the reconstruction pipeline, namely the accurate reconstruction of surfaces containing concave patches, i.e. concavities that do not form part of the object's visual hull. The dolphin reconstruction in **Figure 4.4** illustrates the ability of the algorithm to reconstruct visual hull concavities.

4.3.1 Design

The following observation facilitates the motivation of this algorithm: looking at any point in 3D space from different angles can result in two possible outcomes:

- The colours of the point differ as the view of that object changes (as we look at that point from different angles); the point can thus be classified as a point in empty space;
- The colour of the point remains the same (with the exclusion of colour changes due to the illumination cone) from different angles; this point is part of the surface of an object.

The algorithm makes use of the NCC algorithm's inherent robustness; it is invariant to illumination changes due to rotation of an object.

During space carving, a history of projection matches is recorded for each voxel. The projection matches consist the pixel position information and the associated image used to perform the mapping. The algorithm is somewhat like the Closest Direct Mapped texture generation algorithm in that a number of closest pixel matches are recorded for each voxel. The result can be viewed in two ways, namely:

- The closest pixels that map to the voxel have been determined; or
- The closest pixels are matching pixels, i.e. in the sense of depth recovery, the pixels represent the same part of the surface in different images.

Given the outcomes mentioned at the beginning of this section, it is possible to determine whether a voxel is actually part of the object's surface. NCC can be used to compare each of the pixel matches for every voxel to one another. If a voxel is indeed part of the surface of the object, the NCC matching score for the comparisons will be very high. The result is that, for each voxel, matches have been determined by the space carving component's projection of each voxel onto a number of images, thereby facilitating a single NCC check.

The texture generation component maps only one pixel colour to every vertex in the bounding volume. In order to support concave surface recovery, more colours need to be associated to each

vertex. This involves the extension of the Closest Direct Mapped algorithm to associate the RGB values of a number of pixels with each of the voxels. The storage of the RGB value is also replaced with the image projection information, namely the pixel position and associated image reference instead. The closest projections associated with each vertex can be equated to the way NCC performs its matches. Instead of trying to find the best match using a 2D search window, the proposed algorithm implicitly performs this process. The search part of the correlation algorithm (that performs the difference minimisation) can thus be removed completely.

4.3.2 Implementation

The algorithm, which extends the Closest Direct Mapped algorithm, is as follows:

```
FOR each image I DO
BEGIN
   FOR each active voxel V DO
   BEGIN
     D ← Rotate by angle Θ associated with I
     A ← Projection(D) onto Image Plane
     IF A = Background Pixel THEN
        Deactivate / Disable V
     ELSE
        IF (V has not yet been mapped to a pixel) THEN
        BEGIN

{ Store Projection Information For First Match }
          V.MinDist ← A.Z
          V.Pos[0] ← A
          V.Image[0] ← I
        END
        ELSE IF (V.MinDist < A.Z) THEN
        BEGIN
           V.MinDist ← A.Z

{ Back Up Current Matching Information }

           FOR J FROM N-1 TO 0 DO
           BEGIN
              V.Pos[J+1] ← V.Pos[J]
              V.Image[J+1] ← V.Image[J]
           END

{ Store Current Projection Information }
          V.Pos[0] ← A
          V.Image[0] ← I
        END IF
```

```
        END IF
      END FOR
    END FOR
```

where:

- `N` represents the number of projection matches that can be recorded per vertex;
- `A` represents the projection of `V` onto the current image plane after the voxel has been rotated;
- `V.Image` represents an array of `N` image references;
- `V.Pos` represents an array of `N` pixel position references. Each pixel position refers to the position of `A` in the current image `I`.

The result of this algorithm is that the `N` closest projections (and associated pixel position information for each projection) for each vertex in the bounding volume are recorded. Once this has been done, they are compared to one another using the standard NCC evaluation to determine the correlation between the matches. The pixel positions are used to do this. Assuming that only three pixel projections (`A,B,C`) are associated with each vertex, an NCC matching score is obtained for each combination `AB` and `BC`. If either `AB` or `BC` falls below a minimum matching score or threshold, the voxel is assumed not to be part of the surface of the object and it is disabled. To maximise accuracy, the threshold score should be set close to one.

4.3.3 Results

The results depicted in **Figure 4.25** illustrate the use of a space carving algorithm enhanced to use NCC in its evaluation of the bounding volume. Illustrations A and B show how the enhanced algorithm has managed to identify points on the surface of the face based on how interesting it is. The areas around the eyes, for example, have been classified as interesting. Other areas identified as interesting include some parts of the mouth and parts of the nose. The rest of the face (the areas that are not black) represents the areas that the algorithm found to be uninteresting. The reason for this seems to be that the NCC algorithm has problems performing matches between areas that are bland, such as the forehead. This illustrates the NCC algorithm's inability to accurately determine the appropriate matching scores for bland surfaces, i.e. surfaces that consist of a single colour. Illustrations C and D depict the robustness of the connectivity algorithm. Just as with mesh decimation, the removal of voxels representing the surface of the object has not affected the connection component's ability to generate a connected mesh from the remaining voxels.

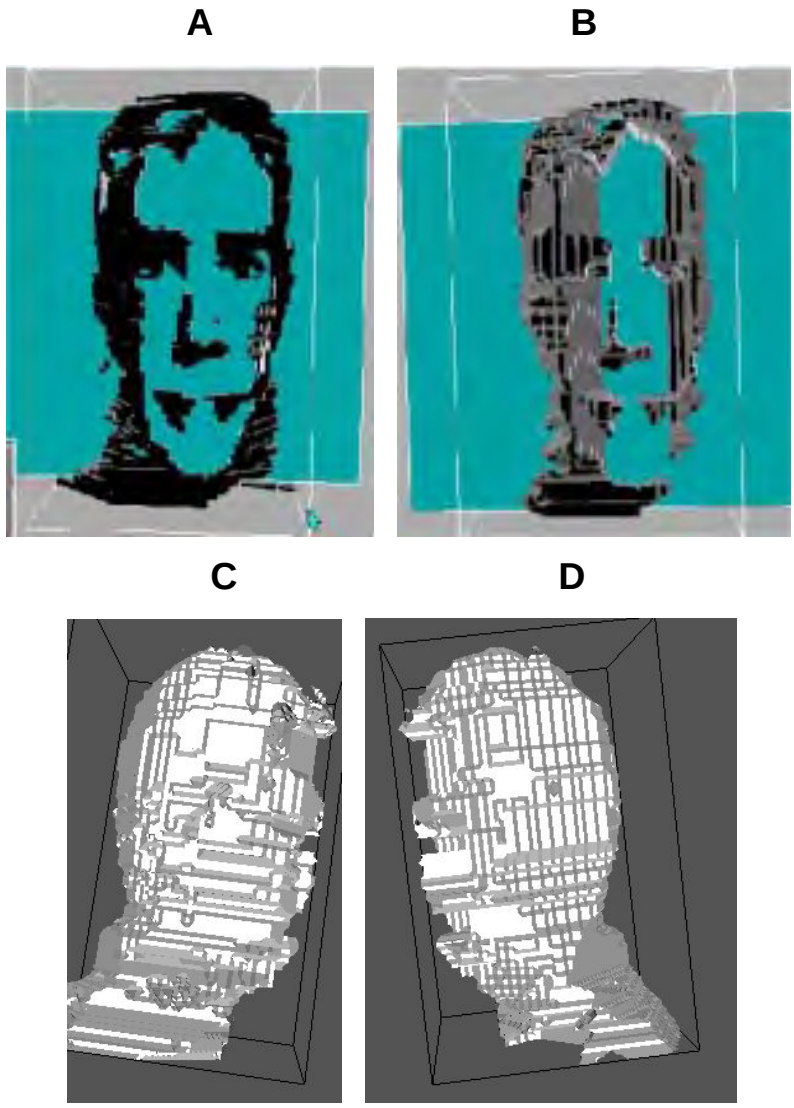


Figure 4.25 The NCC-enhanced Reconstruction Pipeline.

4.3.4 General Comments

The current implementation of the enhanced NCC space carving algorithm is not very good. A great deal of improvement is needed to achieve the desired effect, namely the reconstruction of concave surfaces. However, the use of an enhanced space carving algorithm holds promise for improvements with respect to 2½D scene reconstructions, particularly in speeding up the process.

The use of the reconstruction algorithm in conjunction with the NCC algorithm introduces the possibility of overcoming the sluggishness of the NCC algorithm. The slowest part of a traditional NCC evaluation is in locating matches. The space carving algorithm does this as well

by utilising a more brute-force approach. As reported in the section 4.2.6.3, the complete reconstruction of a vase using a 100x100x100 resolution bounding volume took only 31 seconds. During the reconstruction, each voxel has an associated collection of pixel matches, a result of the texturing approach. The traditional role of NCC in this sense could then be used to only determine only good matches reported by the reconstruction algorithm are, and not to actually find the best matches via the computationally expensive 2D search process.

4.4 Applications and Improvements

This section describes a number of applications of the reconstruction technique detailed in this chapter. Additionally, it also briefly highlights some of the areas in the current implementation requiring improvements.

4.4.1 Level of Detail

The reconstruction of the budgie as depicted in **Figure 4.24** introduces the notion of a possible Level of Detail (LOD) extension. Level of Detail is typically used to simplify the rendering of complex scenes. One way this is facilitated is with multiple copies of an object, leading to progressively simpler polyhedral representation of the object. These simpler representations sacrifice accuracy for speed. When rendering the object, a LOD-enabled renderer evaluates the distance of the viewer to the object and chooses a representation that has an appropriate amount of detail. As a viewer moves closer to the object, so a more detailed representation is used, while a simpler representation is adopted as the distance between the object and the viewer increases. This process thus facilitates improved rendering speeds. The performance enhancement offered by this approach has been hinted at in section 4.2.6.2. Further timings that support this notion are illustrated in the table below.

LOD Rendering Performance		
Model	**Resolution**	**Frames Per Second**
Vase	50^3	20.4
Vase	100^3	6.8
Vase	150^3	3.1
Budgie	50^3	24
Budgie	100^3	10.0
Budgie	150^3	1.2

Simply altering the bounding volume resolution and the associated input images, the improvements in rendering speed are remarkable. The timings were measured using a dual 195MHz SGI Octane Workstation. **Appendix B** illustrates the budgie reconstruction at each of the above-mentioned resolutions, without texturing.

The use of the reconstruction pipeline for LOD reconstructions is a very promising application of this technology.

4.4.2 Texture Mapping

The texture generation component suffers from a number of problems and is, at the moment, not able to generate vertex colour maps of a very good quality. A solution to this problem must be found, as it is imperative that textures of a high visual quality are generated to ensure visually accurate reconstructions. Given this situation, the problem of concavities could also be minimised.

The combination of traditional texture mapping with the Gouraud shaded vertex colouring method must also be investigated. If there is a possible balance between the two methods, it must be found. Additionally, this approach will facilitate the use of the mesh decimation algorithm.

4.4.3 NCC Matching Strategy

The use of an NCC evaluation for depth recovery has been discussed in section 4.3. This section proposes an extension based on that idea to considerably enhance the performance of the NCC

algorithm, while at the same time providing a mechanism for the recovery of dense depth maps with sub-pixel accuracy.

Traditionally, NCC has been used to perform 2½D reconstruction of scenes from stereoscopic image pairs. Typically, NCC is enhanced using the epipolar constraint. As mentioned previously in section 3.3.1.3.2, this constraint narrows the NCC search space from a 2D problem down to a single line. It was also mentioned that the recovery of the fundamental matrix is achieved by finding at least eight matches using the traditional 2D search approach, and then solving simultaneous equations based on these matches to determine the rotation and translation between the two images. Adopting this approach, the information generated by the fundamental matrix could be used to place the different images around the bounding volume according to the rotation and translation information extracted. The projection of each of these images into the volume would then result in the situation mentioned above, namely a number of pixel matches for every voxel. The NCC evaluation could then be used to determine whether a voxel is indeed part of the surface of the object being reconstructed.

The use of a bounding volume thus reduces the NCC search space from a 2D problem down to a single check, and in theory should speed up the 2½D reconstruction method and in the process generate dense depth maps of a scene very quickly without the need for hardware-based NCC implementations. The use of a bounding volume that has a higher resolution than the images e.g. voxels at 0.5 unit intervals as opposed to the pixel unit intervals used throughout the discussion in this chapter, means that sub-pixel depth recovery can be performed.

4.5 Summary

This chapter details the development of a model acquisition system using only a standard desktop CCD camera that is well suited for the reconstruction of avatar head. The idea is that a person wanting to use a Virtual Conferencing system has a method available for the low cost reconstruction of his/her head. The system is based on a component-based reconstruction pipeline that is classified as a visual hull reconstruction algorithm. It is composed of the following core components:

- **Space Carving** (section 4.2.1): The object is assumed to lie in a bounding volume of voxels. Several pictures are taken from different angles and the backgrounds are stripped from these images. Voxels projecting to background pixels in each of the image are disabled. The result of doing this for all the images leaves a set of voxels representing the object that is to be reconstructed. During the extraction process track is kept of the image

(as well as the position in that image) to which each voxel is closest. The result is a collection of voxels in the bounding volume representing the object;

- **Isosurface Extraction** (section 4.2.2): Voxels not representing the surface of the object are removed. This is achieved using a 3D filter approach, whereby a voxel is removed if it is surrounded by active voxels. The result is a hollow shell of voxels representing the object, as well as a large decrease in object size;
- **Mesh Connectivity** (section 4.2.3): The voxels remaining after isosurface extraction are converted into a closed, connected polyhedral representation. The connectivity algorithm works on the assumption that the bounding volume of voxels has a 3D regular structure. A coarse-to-fine subdivision of the bounding volume into cells facilitates the robust connection of the hollow shell of voxels. A lookup table containing 12 entries defines the connectivity of all vertices in each cell (section 4.2.3.3). Each lookup entry defines a triangular polygon by specifying which cell vertices combine to form the triangle, and which side of the cell on which the triangle. A process of diagonal state support is discussed in section 4.2.3.4 whereby, given that a cell vertex is not present, a mirror is searched for and if found, all lookup entries containing that vertex are updated by replacing the vertex with its mirror. The search for a mirror vertex thus involves determining whether an active vertex is present on the orthogonally opposite side of the cell.
- **Texture Generation** (section 4.2.4): Texturing is simulated through Gouraud shading. A vertex colour map is generated, that allows the connected mesh to be Gouraud shaded. The result is a cheap texturing mechanism that avoids the overhead associated with traditional texture mapping approaches. Unfortunately, the need for a high number of vertices packed closely together to obtain a realistic result affects rendering performance adversely;
- **Mesh Decimation** (section 4.2.5): This is an optional component with respect to the framework. A bounding rectangle approach is employed to decimate axis-aligned rectangular patches. The decimation process replaces any suitable rectangle with two triangles.

The result of the reconstruction implementation is a textured, closed, connected polyhedral representation of the real-world object. It is shown in section 4.2.6.3 that the reconstruction pipeline has complexity *O(n)*, and that the performance is dependent on the size of the bounding volume used.

To overcome the pipeline's inability to reconstruct surface concavities that do not form part of the object's visual hull, a novel algorithm based on the Normalised Cross Correlation is developed in section 4.3. The hypothesis on which this algorithm is based is that if the surface of an object is inspected from a number of different angles, the colour at the point on the surface remains the same, bar illumination changes. Fortunately, the NCC evaluation ignores illumination changes. Given the space carving component's ability to generate the shape of an object and to capture RGB values at each of the voxels, an NCC evaluation is used to determine whether a voxel has the same colour in different images i.e. from different angles. If the NCC evaluation determines that similarity is strong, it is assumed that the voxel is part of the surface of the object. Otherwise, it is removed/disabled. In this case, the NCC algorithm is not used to determine the best possible match, but rather to determine whether a number of pre-defined matches are correct. This means that the shape extraction algorithm has performed a matching process that very quickly determines matches.

An optimisation relating to the NCC algorithm's matching performance is proposed in section 4.3.3. It is based on the same principle as the concave surface recovery enhancement, namely that the space carving component returns brute force matches. The enhancement proposes the use of a bounding volume with a stereoscopic image pair to facilitate the recovery of dense depth maps. Sub-pixel accuracy can be achieved by increasing the resolution of the bounding volume.

A LOD extension is discussed in section 4.4.1. It involves the reconstruction of object using different bounding volume resolutions.

A shortcoming that is addressed in section 4.4.2 is the problem with the texture generation component, namely the algorithm's inability to generate visually correct textures.

4.5.1 Contributions

The most significant contributions of this chapter are as follows, namely:

- The development of a cheap generic model acquisition system that is able to generate fully textured representations of any real-world object, with a final implementation shown to exhibit complexity *O(n)*. Although the reconstruction pipeline underlying the model acquisition system is geared towards facial modelling, it has an added aspect of generality in that it is able to handle the reconstruction of any object it is presented with, and not only human heads, using an inexpensive CCD camera.
- A novel connection algorithm based very loosely on the marching cubes algorithm. It is able to generate completely closed 3D models;

- The use of NCC to perform concave surface reconstructions; this is facilitated by the space carving component's mapping process;
- The introduction of a method to reduce NCC search space from a 2D problem down to a single evaluation, and allow for much improved performance enhancement for the recovery of dense depth maps with sub-pixel accuracy.

Chapter 5 - Encoding

5.1 Introduction

This chapter discusses the encoding process that is responsible for converting a live video stream into avatar control data. This encoding involves the evaluation of a video stream, firstly to determine the position and orientation of the subject's head (**pose estimation**), and then to analyse his/her facial mannerisms (**expression analysis**).

The pose of a subject's head can be determined easily using electromagnetic trackers. The problem with this approach is that, although VR systems rely on this equipment for accurate 6 DOF position information in real time, it tends to be very expensive. The field of augmented reality offers cheaper image-based alternatives to pose estimation, and research in this field is used as a basis for the implementation discussed here. Expression analysis is performed using a rule-based threshold system such as the one used by Yacoob *et al.* [7], where an expression becomes valid if a specific threshold is exceeded.

Having obtained the pose information and performed expression analysis, an avatar can be controlled remotely to simulate the user's movements and expressions. To facilitate the conversion process from the expression analysis phase to the expression generation phase, an **Expression Parameter Lookup Table** (EPLT) is used. Its function is to perform a mapping between the two phases. Just as the MPEG-4 FAP mechanism provides an independent mapping between the subject independent FAPs and the model specific FAPUs (see section 3.6.1.3), the EPLT specifies a collection of high-level generic expressions, the subject-specific expression analysis rules, and the avatar-specific parameters required to generate the expression. Any subject with appropriate expression analysis rules defined can thus be used in conjunction with any avatar having the appropriate deformation parameters defined for each expression.

Although the pose estimation and expression analysis problem domains fall into two different categories, they can both be classified as image processing or image analysis problems, and as such are discussed here in tandem. The technology underlying both the head tracking and expression analysis implementations is based on an NCC matching strategy that has been applied to a live video stream.

The discussion presented here firstly focuses on the high level design of the encoder, and then goes on to discuss the concept of fiducial graphs. This is followed by an explanation of the motion tracking implementation that is based around an NCC evaluation. The approach to pose

estimation is subsequently discussed. Details about the implementation of the expression analysis component are then provided. Finally, experimental results are provided and discussed.

5.2 Design

Given an input video stream, a number of points on the subject's face are chosen and tracked. These **fiducial points** are tracked by the system using a hybrid feature-based/correlation-based tracking algorithm. The tracking algorithm's implementation is based on a Normalised Cross Correlation approach. The choice of this algorithm is motivated by the following factors, namely:

- It is a simple algorithm;
- It is able to generate robust tracking results. It will be shown that the algorithm is invariant to scaling and rotations of the subject being tracker. It is also invariant to changes in illumination, thus avoiding the need to convert the incoming RGB signal to an equivalent YUV/YIQ signal. This is typically performed because the YUV colour space is able to separate illumination effects (Y) from colour information (U and V). Rekimoto [54] performs this conversion to effect background elimination for this reason;
- It does not require pre-defined templates of the subject being tracked, as with the pattern tracking approaches (see section 3.4.1.2);
- It provides a mechanism for unobtrusive face tracking. This is in contrast to the approach discussed in section 3.4.5, which involves the explicit marking of face positions using brightly coloured fiducials to simplify fiducial point classification and tracking.

At the core of the encoder's design is the tracking algorithm. The requirements are that it should be robust and fast. As will be seen in this discussion, the Normalised Cross Correlation algorithm fulfils these requirements. The result is a tracking algorithm that is able to achieve robust image-based tracking with near-real time speeds (real time here referring to the 15 fps minimum).

The invariance of the tracking algorithm to rotations and scaling of the subject arises directly because of the proposed robustness of the NCC tracking. Unlike template-based tracking mechanisms, the NCC-based tracker discussed here uses a feedback loop: as a point is tracked and located in the current image, its new position is used as input for the next iteration of the tracking of that point; tracking occurs on the new position of the point. This approach has the advantage that the search criteria provided as input to the tracking algorithm is completely dynamic. To counter the fact that an error in tracking may occur, possibly due to occlusions, the

feedback loop is user-assisted. This means that if errors in tracking do occur, the user is able to force corrective measures to be executed by the tracking algorithm.

The automatic location of fiducial points on the face is not discussed. This is deemed beyond the scope of this dissertation. As such, a user-assisted location mechanism is employed; it is assumed that the user specifies a collection of fiducial points in advance. Six fiducial points are manually selected around the face: three are used to determine the pose of the head, while four fiducials (including one of the pose fiducials) are used during the expression classification process. As the tracking is performed, so the positions of these points change relative to each other.

Pose estimation and expression analysis is performed by modelling the face as a plane, an approach similar to the method discussed by Black *et al.* [7]. The motivation for using an approach similar to theirs comes about because of their discussion on the accuracy of the pose estimation: whereas more enhanced models (such as ellipsoids) could be used to obtain more accurate tracking results, the approximate results using a plane are acceptable. The process of expression analysis falls into the rule-based category, which is also very similar to Black *et al.*'s [7] methodology, which is constrained by the EPLT. The subject-specific rule-based evaluations defined for each generic expression in the EPLT translate into a collection of threshold measurements. If a combination of thresholds is exceeded, the associated expression is selected from the EPLT.

The result of the encoding system is the conversion of a live video stream into avatar control data, thus leading to a reduction in bandwidth requirements by several orders of magnitude.

5.2.1 Fiducial Graph

A fiducial point can be described as “an interesting point on a person's face that is used for tracking". The term interesting is included in this description to ensure robust NCC tracking. The NCC algorithm does not perform very well with bland areas, as discussed in section 3.3.1.3.1. As such, the fiducials selected for tracking have been chosen specifically because they are typically found around areas of the face exhibiting edges.

Six fiducial points are selected, and grouped into what is termed a **Fiducial Graph**. This concept is very similar to the face bunch graph (see section 3.6.2) in that it is composed of a number of nodes representing fiducials on a subject’s face. Additionally, it defines a standard graph structure to which any subject can be fitted; each of the lengths of the edges of the graph are specific to the subject used, but because of the standard structure of the human face, each

node can be referred to generically. The two fundamental differences between a fiducial graph and a face bunch graph are:

- A fiducial graph is related to a single person, and does not constitute a collection of fiducial information for the sample population used per node;
- Whereas face bunch graphs are used primarily to perform automatic face location, fiducial graphs are used to perform expression analysis and pose estimation.

The points that constitute a typical fiducial graph are:

0) the left eyebrow/eye (L_e);
1) the right eyebrow/eye (R_e);
2) the Nose tip (`Nose`);
3) the Chin tip (`Chin`);
4) the Left Mouth Edge (L_{me});
5) the Right Mouth Edge (R_{me});

The R_e, L_e and `Nose` fiducial points are used to estimate the pose of the user, while the remaining points, along with the `Nose` fiducial, are used to classify expressions. A sample fiducial graph is illustrated in **Figure 5.1**. The choice of these points thus facilitates a subject-independent face specification, a factor that is used to perform pose estimation and expression analysis later. The fiducials that will ultimately be used to perform pose estimation are required to be as invariant as possible to muscle deformations due to expression changes or speech. As such, the L_e, R_e and `Nose` fiducials are used, since they have been found by the author to be least affected by changes in expressions or speech. This is based on the observation that the position of the `Nose` fiducial remains fixed relative to the subject's head, and is not affected by most expression. Furthermore, the L_e and R_e fiducials are subject to upward movement with a surprise expression, or movement towards each other with an anger/frown expression. The use of larger correlation windows to cover more of the eye/eyebrow area can compensate for this motion.

Similarly, the fiducials around the mouth, including the `Nose` fiducial are used to perform expression analysis. The classification process makes used of thresholds to determine which expression is the currently active one.

The fiducial graph representing a subject is thus used in the following context:

- Edge lengths of the mouth fiducials for rule-based expression classification;
- Edge length comparisons between different fiducial graphs for pose estimation.

At initialisation, the user is required to assume a neutral expression and face the camera directly. The image is captured and the fiducial points matching the above list are selected and fiducial graph generated. During the conference, the fiducials are constantly updated as the video feed is evaluated, thereby updating the fiducial graph. After the new position of each fiducial has been determined for the current expression, the current fiducial graph is then compared to the original fiducial graph to determine the head pose and the expression exhibited by the user during the conference.

5.3 Motion Tracking

Given an incoming video stream, and initial fiducial point locations, it is necessary to track these points to perform pose estimation and expression analysis. Although Normalised Cross Correlation is a slow algorithm exhibiting complexity $O(n^2)$, its robustness lends itself very well to this task.

This section discusses the application of NCC to a video stream and then discusses various unrelated issues that affect the robustness of the final tracking process.

5.3.1 Normalised Cross Correlation

Normalised Cross Correlation has been discussed previously in the context of depth recovery for 2½D scene reconstructions from stereo image pairs (see section 3.3.1.3). To that end, it has been introduced as a robust mathematical approach for determining pixel matches in different images.

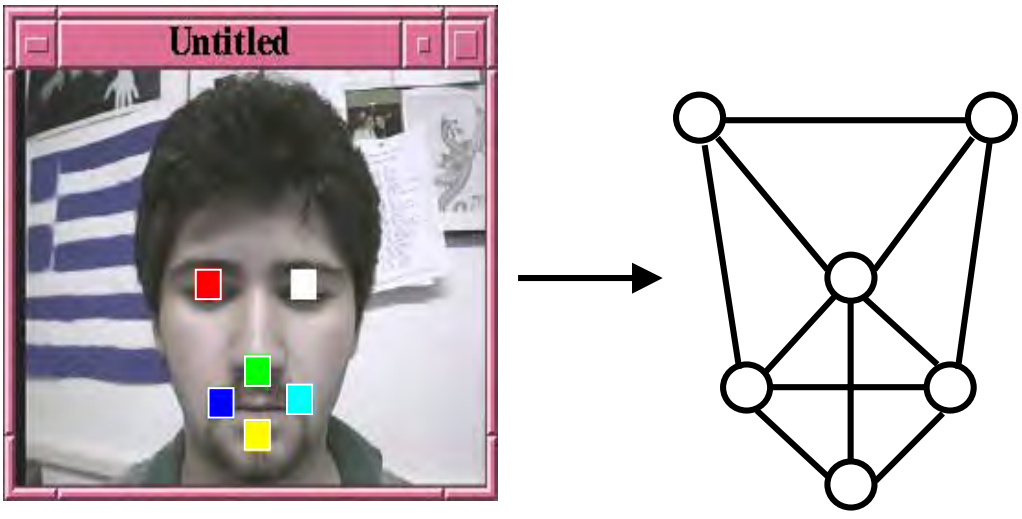


Figure 5.1 The structure of a fiducial graph overlaid on top of a real subject.

This notion extends very well to head tracking. Given an image $\mathbf{I_1}$ from a video sequence **V**, a pixel **p** can be tracked by performing an NCC evaluation using $\mathbf{I_1}$ and the succeeding image $\mathbf{I_2}$. An NCC search evaluation is then performed to find the best match for **p** in $\mathbf{I_2}$. NCC in its traditional form is very computationally expensive ($O(n^2)$). This occurs because of three factors, namely:

- The search window size: as the search window size increases, so the area that is evaluated increases. A 32x32 search window will cause the NCC algorithm to take 4 times longer for the evaluation as a 16x16 window;
- The correlation window size: this is subject to the same performance constraints as the search window size. Any increase in the correlation window will lead to a four-fold decrease in general performance, but will also cause more of the background to be included in the evaluation, thus leading to results that are more robust;
- The frame size: ultimately the size of the incoming video frame dictates the search window and the correlation window sizes, because the object to be tracked becomes smaller, thus leading to reductions in the required search window and correlation window sizes. Furthermore, the frame size is also subject to the performance constraint mentioned above, but because it is larger than the other two factors, any change in its size will also affect performance the most.

These three factors affect the performance and the robustness (to varying degrees) of the NCC tracking algorithm. The goal is thus to find a minimising frame size/search window size and a maximising correlation window size combination that will generate the best tracking results. This optimisation problem is because of the observation that the inter-frame image-coherency of **V** is assumed constant for a short period of time (see section 3.4.1.3, specifically Carceroni *et al.* [10]). This implies that a position in image V_1 will move to a position in V_2 that is very close to V_1. By extension, however, the later V_2 is taken after V_1, the lower the inter-frame image-coherency will be, and thus, the higher the probability of failure. The failure could also occur as a direct result of occlusions, which would result in a false match being returned and then tracked in succeeding frames.

A further advantage of the NCC evaluation is its ability to ignore illumination changes during tracking. During the NCC tracking algorithm, the ratio $I(p)/I_{PW}(p)$ is used as a search criterion. $I(p)$ refers to the intensity of pixel p, while $I_{PW}(p)$ refers the average intensity of the pixel window surrounding p, with each pixel in the pixel window contributing to the average. The use of this ratio causes the effects of changes in illumination to be ignored by the tracking algorithm.

5.3.2 Tracker Calibration

The NCC-based tracker requires calibration. This process attempts to find the smallest correlation window that will return robust results. Any static scene can be used, with an arbitrary point selected, preferably one that is quite busy and therefore not a blank surface such as a white wall; ideally, the area to be used for the conference should be employed. Once the point has been selected, the NCC-based tracker begins tracking the point with an ever-increasing correlation window size. While the correlation window size is still too small, the tracker exhibits very erratic behaviour i.e. with a static scene, the tracker is unable to track the designated point accurately. As the window size becomes bigger, some point is reached where the tracking results obtained are accurate; the erratic behaviour is eliminated and the tracking is accurate i.e. the desired point remains tracked in the static scene. Once this situation arises, an acceptably large correlation window size has been chosen.

Although this process facilitates a minimum correlation window size selection, it by no means implies that the tracker will be able to track the subject robustly. Further evaluation for correct parameter selection, as is discussed in section 5.6.2.4, is required for this.

5.3.3 Head Tracking

Once the most appropriate correlation/search window sizes have been determined, it is necessary to determine the positions of the fiducial points on the user's face. During initialisation, a video frame depicting the subject in an upright position (and with a natural expression) is obtained. The subject is then required to manually select the fiducial points from this image. In addition to repeatedly specifying the positions of the different points, he/she is able to continually select a different video frame. This process is repeated until the user is satisfied with both the frame and fiducial point selection.

Having obtained this calibration data, the next step involves tracking these fiducial points. The NCC tracking algorithm is initialised by making a copy of the data that has been acquired during the calibration step, namely the fiducial graph and the associated frame of video. This copy is used as the initial input data for tracking. Since the NCC tracking algorithm utilises two successive video frames in its evaluation, the current frame from the incoming video stream is married with the calibration data's image to form this pair. The tracking algorithm proceeds as follows:

```
FOR each fiducial A in fiducial graph GT DO
BEGIN
   find CURR.A that best matches GT.A (via NCC tracker)
   GT.A ← CURR.A
END FOR
GT.Image ← CURR.Image
```

where `GT.Image` represents the image acquired during calibration, and `CURR.Image` represents the current image.

This process is repeated for the duration of the tracking. As can be seen, `GT` represents the fiducial that is constantly updated and used as input to the tracking process. Based on the assumption that the inter-frame image coherency between `GT.Image` and `CURR.Image` will be very high, the NCC matching score used is very high, namely 0.95.

5.3.3.1 Fiducial Graph Support

The positions of the fiducial points vary with respect to the subject used. However, the positions remain relatively constant for each subject. These observations imply that the fiducial graph information for each subject introduced into the system can be used at a later stage. Having this facility thus enhances the system considerably and saves a reasonable amount of time during the calibration step of overlaying the fiducials on an arbitrary frame of video containing the subject. In addition to storing the pixel positions, the size of the frame used to obtain the fiducial positions is also saved to the same file containing the position data. This data is used to scale the fiducial positions if a different sized video frame is used during calibration. In that situation, the fiducial positions can be scaled appropriately.

5.3.3.2 Object Size

Maximisation of inter-frame image-coherency is a very important aspect. It directly affects the accuracy of the NCC tracker. As mentioned above (in section 5.3.1), the goal of robust tracking involves finding an optimising combination between the frame size, the search window size and the correlation window size. The maximisation process that is applied to the correlation window size is also required to take into account the size object that is to be tracked. If the selected correlation window size is large relative to the object, it will lead to tracking errors. This is illustrated in **Figure 5.2**. Illustrations A through D depict how a large correlation window causes tracking failure. In this situation, as the head rotates, a number of the correlation windows begin

sticking to the outline of the user. This is due to occlusions of parts of the large correlation windows due to rotation. The result is that the tracking fails with a smaller degree of rotation. Evaluating illustrations E to H, one can see that with a smaller correlation window, the degree of rotation of the head is increased. The rotation of the head in these images is much greater, and the fiducials being tracked have not been compromised. The frame size used is 128x128, the search window size is 18x18, and the larger correlation windows are 14x14, while the smaller ones are 6x6.

5.3.4 User-Assisted Tracking

The implementation of the NCC tracking system is a naïve one. It does a brute-force rectangular search in the current image for the match with the highest value. Unfortunately, this means that any incorrect or false match returning the highest NCC score in the current evaluation will be chosen as the correct one. In order to prevent the system from becoming completely unusable when such a situation arises, the final implementation is based on the idea of **user-assisted tracking**. This implies that any tracking errors that occur are corrected by the user. To that end, the user is provided with a video feed and fiducials overlaid on top of the video feed, as

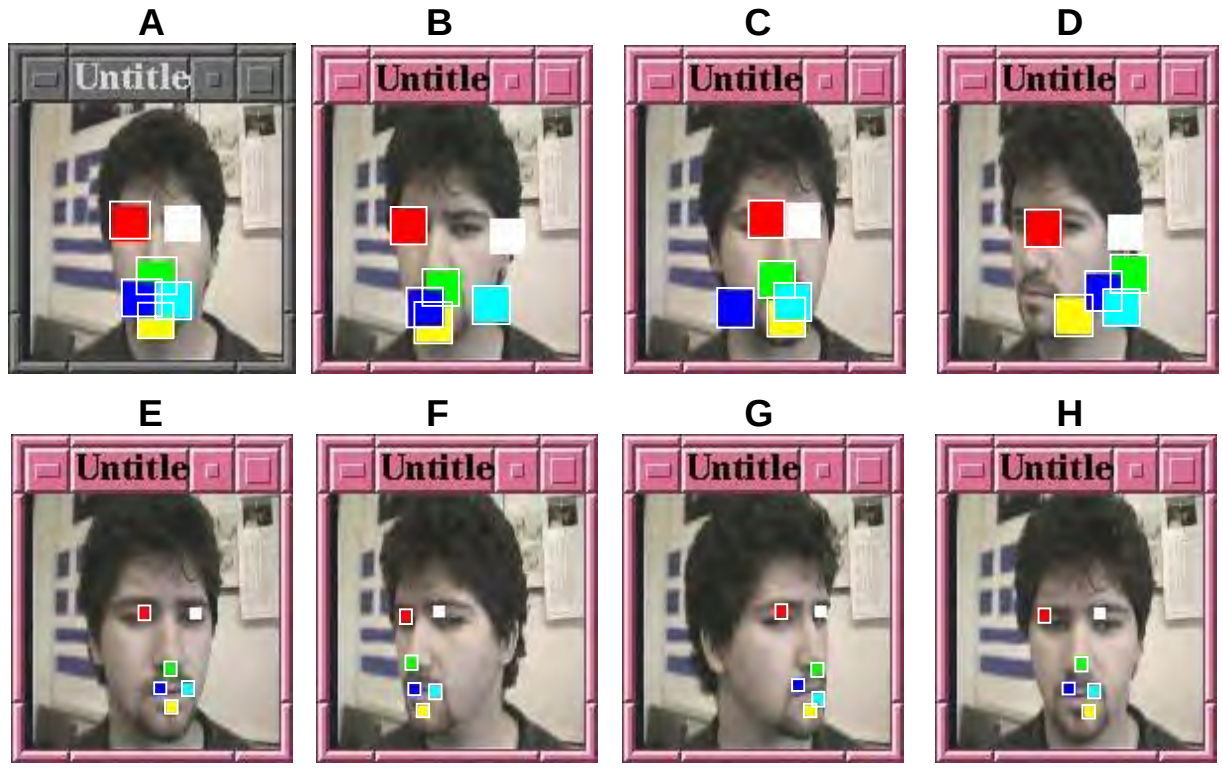


Figure 5.2 Tracking with different pixel window sizes.

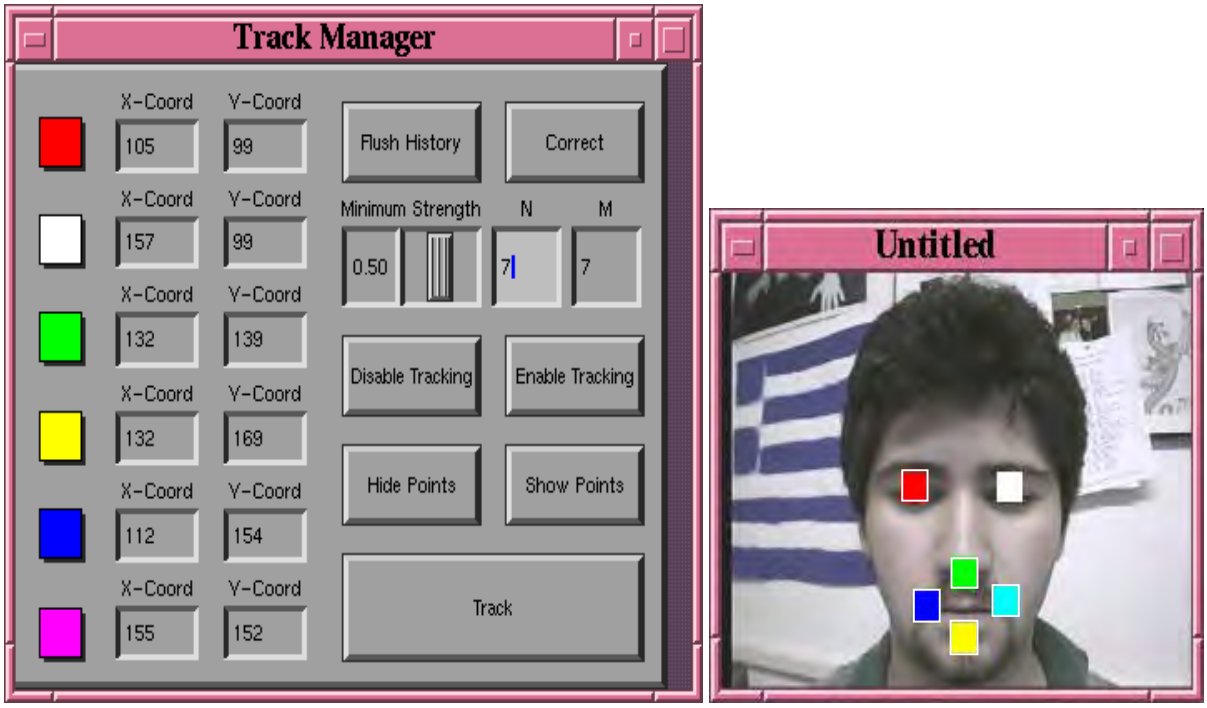


Figure 5.3 The interface of the Encoder Application.

illustrated in **Figure 5.3**.

Initially, the user is able to specify the position of each fiducial point. Additionally, he/she is able to specify the pixel window sizes (**N**/**M**) and the Minimum NCC matching strength (**Minimum Strength**). He/she is also able to initiate the tracking process (**Enable Tracking**). The interface also facilitates calibration as discussed in section 5.3.2, where a minimum acceptable correlation window size is determined, and further, the placement of fiducials relative to the subject's face. The fiducial points are overlaid on top of the frame of video captured from the camera. Additionally, each fiducial point is colour coded, and its pixel position (X,Y) is listed next to the relevant colour on the `TrackManager` window panel.

Once the user is satisfied with the positions of the fiducials relative to the face, the tracking process is initiated. When this occurs, the selected fiducial coordinates and the initial frame of video are used as input to the search process.

As tracking is performed, so the fiducial points are tracked according to the live incoming video stream obtained from the CCD camera. When any false match is classified as being correct, the relevant fiducial point will assume an incorrect position on the subject's face. Given the window displaying the video feed, the user is able to constantly monitor the current frame and the positions of the fiducial points relative to the face in that frame. In this way he/she can decide whether or not the current position of each fiducial is acceptable. If a situation arises

where a tracking error is detected by the user, the system provides him/her with facilities to correct the tracking error(s).

Two possible mechanisms are supported to correct errors. The first error correction mechanism employed involves a recalibration process, whereby the fiducials are overlaid on top of a static frame of video containing the subject. This is similar to the calibration step defined above, but is time-consuming, since it involves the user having to assume the same pose and expression as during the initial calibration step. The user places himself/herself in the frame of video with respect to the positions of the initial fiducial points. Once satisfied with the new fiducial positions, the user can initiate tracking again.

The alternative approach involves the use of a caching mechanism. This is based on the assumption of inter-frame image coherency, namely that at any given time, the user is close to a position he was previously. Therefore, if a tracking error has manifested itself with any of the current fiducials, an earlier correct fiducial graph can be used to continue the tracking. To that end, an array of past fiducial graphs and associated frames of video is employed as the cache. The tracking algorithm employing this approach is as follows:

```
oldnum ← currnum
currnum ← (currnum + 1) MOD MaxHistory
IF (totalnum < MaxHistory) then
     totalnum ← totalnum + 1
END IF
FOR each fiducial point X in HISTORY[currnum].A to be
matched in CURR DO
BEGIN
    FindNCCMatch(CURR.B, GT.A)
    HISTORY[currnum].A ← CURR.B
END FOR
HISTORY[currnum].Image ← CURR.Image
```

where:

- `MaxHistory` represents the maximum possible cache entries;
- `HISTORY` represents the cache of past correct fiducials and images;
- `HISTORY[currnum]` represents the new image/fiducial graph;
- `GT` is the last fiducial graph/image combination that was added to `HISTORY`;
- `CURR` is the new fiducial graph and associated image; it represents the fiducials that will be tracked in the new image;
- `FindNCCMatch` finds the best possible matches for each fiducial point in `GT` for `CURR.`

`HISTORY[]` is a cyclical array, and as such, when `currnum` reaches the `MaxHistory` count, it begins storing fiducial graph/image combinations at the beginning of the array again.

If a tracking error occurs and corrective measures are initiated by the user, `currnum` is updated as follows:

```
delta ← MaxHistory DIV 2
IF totalNum < delta THEN
     delta ← totalNum-1
END IF
currnum ← currnum - delta
IF currnum < 0 THEN
      currnum ← MaxHistory + currnum
END IF
totalNum ← totalNum - delta
```

`currnum` thus points to the Fiducial Graph and image `MaxHistory DIV 2` elements earlier. Additionally, `delta` is subtracted from `totalNum`, thus rendering the remaining images after `currnum` in the cache array invalid. The result of this correction process is that the user is able to make use of past correct tracking results to dictate what the input tracking parameters for the next frame of video are to be. This also means that the previous error correction mechanism can be avoided altogether.

5.4 Pose Estimation

Pose estimation calculations are performed by modelling the face as a flat plane. The plane used for evaluation is the one formed by connecting the left eyebrow (L_e), the right eyebrow (R_e), and the nose (`NOSE`) fiducial points together. Connecting these points together forms a triangle `T`. The major assumption applied with the pose estimation is that, during the placement of the fiducial points at the appropriate positions on the subject's face, he is assumed to face the camera directly. As such, the fiducial points representing `T` are a maximum distance from each other. This assumption facilitates pose extraction.

The pose can be represented by three rotations, namely a rotation around the X-axis, a rotation around the Y-axis, and finally, a rotation around the Z-axis. These three rotations can be represented by a single quaternion, while the position of the head can be represented by a translation vector. Both the quaternion and the translation vector are computed by comparing the current fiducial graph to the graph obtained during the **acquisition phase**, during which the subject is instructed to exhibit a neutral pose and face the camera directly. The quaternion

specifies how the head is rotated with respect to the fiducial graph obtained during the acquisition phase, while the translation vector indicates the magnitude of horizontal and vertical movement. Additionally, the calculation of a scaling factor facilitates the calculation of the Z-component of the translation vector.

It has been mentioned in the previous section that a copy of the acquisition or calibration data is used to perform general tracking; the copied data resides in `GlobalTemplate`. During tracking, the parameters represented within `GlobalTemplate` are constantly updated as the fiducial points are tracked over time. Therefore, at any instant, there exists an untouched copy of the original calibration data as well as the current `GlobalTempate (History[currnum])`. The pose estimation makes use of the original calibration data and the current `GlobalTemplate` to determine the five variables required for pose estimation; the derivation of each is discussed below.

The goal with the recovery approach described here is not accuracy or recovery, but rather an estimate of subject movement. This is acceptable since it means that, while the recovery is not very accurate, the avatar nevertheless mimics subject movement. This inherent inaccuracy arises out of the simplicity underlying the recovery method below. Methods that are more complicated, such as the transformation approach discussed in section 5.7.2, provide mechanisms for improved and more accurate pose recovery.

5.4.1 X-Axis Rotation (Pitch Recovery)

The determination of the angle of rotation about the X-Axis is illustrated in **Figure 5.4**. As the

Figure 5.4 X-Axis Rotation (Pitch Recovery).

head rotates about the X-Axis, so the projected distance M_2Nose decreases accordingly. If M_2Nose is the same distance as M_1Nose, the Theorem of Pythagoras can be used to determine the magnitude of angle Γ. The computation of this angle facilitates the determination of the degree to which the user's head is nodding from his/her original position. From the original acquisition data, the distance from M_1Nose is computed. This distance is denoted by D_1. As the user rotates his head about the X-axis, so the projected distance D_2 decreases. A right-angled triangle $O\text{-}Nose\text{-}M_2$ is then used to compute angle Γ, as follows:

$$\Gamma = \cos^{-1}(D_2 \,/\, D_1)$$

It is also necessary to determine whether the angle of rotation is positive or negative. In this situation, it can be determined by evaluating the distance eL_1eR_1. If eL_1eR_1 is greater than eL_2eR_2, then the head is moving downwards, as is illustrated in the image. This is because both the fiducials eL_2 and eR_2 are now closer to the camera.

5.4.2 Y-Axis Rotation (Yaw Recovery)

The derivation of the rotation angle about the Y-Axis is very similar to the derivation for the X-Axis. It is illustrated in **Figure 5.5**. The degree of rotation about the Y-Axis, involves using line segment eL_1eR_1 to represent the real magnitude of eL_2eR_2. Assume that the real distance eL_1eR_1 is denoted by D_1 and eL_2eR_2 is represented by D_2. A rotation about the Y-axis is signified by a decrease in the size of D_1 to D_2. Given that the real distance of eL_2eR_2 is D_1 and

Figure 5.5 Y-Axis Rotation (Yaw Recovery).

its projected distance is D_2, any change in D_2 over D_1 means that the head has rotated about the Y-axis. The magnitude of rotation is thus:

$$\Theta = \cos^{-1}(D_2/D_1)$$

The computation of the sign of the angle is performed by exploiting the effects of perspective. As the user rotates his/her head about the y-axis rightwards, eL moves closer to the camera and eR moves further away from the camera. As such, if the distance eN_L is greater than eN_R, so Θ is assumed positive, while on the other hand it is negative.

5.4.3 Z-Axis Rotation (Roll Recovery)

A rotation about the Z-Axis implies a rotation of angle Λ. This angle is computed in a very similar manner as the calculations above, and is illustrated in **Figure 5.6**. The angle Λ represents the angle of rotation about the Z-axis. The derivation is based on the observation that a Z-axis rotation occurs in the image plane. Using the Euclidean distance eL_1eR_1 to represent the real magnitude of eL_2eR_2, the rotation can be determined. Once again, eL_1eR_1 is denoted by D_1, while eL_2eR_2 is called D_2. Looking at the resulting rotation from above, eL_1eR_2 represents D_1 having rotated through angle Λ. As such, D_2 represents the projected distance eL_1O. The magnitude of rotation is thus:

$$\Lambda = \cos^{-1}(D_2/D_1)$$

Figure 5.6 Z-Axis Rotation (Roll Recovery).

5.4.4 Translation

The fiducial graph is used to compute the magnitude of the translation vector directly. This is done by using a Centre of Mass (CM) approach. The coordinates of all six of the fiducial points are averaged out. This averaged value is known as the Centre of Mass for that specific fiducial graph. The Centre of Mass for the current `GlobalTemplate` (`CMcurr`) is subtracted from the one for the Acquisition data. The resulting translation vector is derived as follows:

$$TR_X \leftarrow CM_X - CMcurr_X$$
$$TR_Y \leftarrow CM_Y - CMcurr_Y$$

The derivation of a value for TR_Z, the distance of the head from the image plane, is provided below.

5.4.5 Scaling

Scale is indicative of how far the user is from the camera. Use is made of ratios to determine the scaling factor that must be employed to accurately determine the distance of the user to the image plane.

Assuming the distance of the user to the image plane during Acquisition is 1, perspective effects can be used to determine a scaling factor. The derivation is based on the assumption that if the user moves towards the camera, the triangle edges will appropriately increase. It was mentioned above that connecting `eL`, `eR` and `Nose` fiducials gives a triangle `T` (and initial edge lengths `A`, `B` and `C`). Given `Tcurr` for the current fiducials, the normalised scaling factor can be calculated as follows:

$$A_T \leftarrow Acurr / A$$
$$B_T \leftarrow Bcurr / B$$
$$C_T \leftarrow Ccurr / C$$
$$Scale \leftarrow (A_T + B_T + C_T) / 3$$

where `Acurr`, `Bcurr`, and `Ccurr` represent the edge lengths of the current triangle `Tcurr`. Assuming the user is closer to the camera than during the Acquisition phase, `Scale` is greater than 1; if he/she is further away, the value is less than 1. Assigning an arbitrary distance for the initial position of the users head, say 100, `Scale` is multiplied against this value to obtain a depth value for the subject's head. This translates to the Z-component of the translation vector.

5.5 Expression Analysis

Expression analysis is performed according to a rule-based system. Each expression that the system has knowledge about is expressed in terms of a number of rules. If the rules for a specific expression are adhered to then the system ceases in attempting to further classify expressions.

The approach followed below is equivalent to Yacoob *et al.*'s [7] method, bar one point: the analysis loop does not wait for the expression to complete before identifying it. Instead of subdividing the expression analysis process into 3 phases, 2 states are used: an expression is either active or not.

The current fiducial graph is compared with the fiducial graph obtained during the calibration phase. Threshold measurements are determined from the width and height of the fiducials surrounding the mouth. Only if the user exceeds the threshold does the system classify his/her expression accordingly. This is discussed below.

5.5.1 Expression Parameter Lookup Table

The purpose of the EPLT, as mentioned in section 5.2, is to provide a concrete, independent mapping between subject-specific expression analysis and model-specific expression generation. A generic lookup table consists of a list of generic expressions and associated Expression IDs. The generic lookup table is enhanced for a specific subject/avatar combination by adding the following information to each expression in the generic lookup table:

- The subject-specific expression analysis rules that are used for the threshold classification system described in the next section;
- The avatar-specific expression generation parameters required by the decoder for expression replication.

5.5.2 Classifying Expressions

This section describes a mechanism for performing expression analysis using a set of general rules, similar to the method employed by Yacoob *et al.* [7]. As mentioned in section 3.6.6, Ekman *et al.* [16] identifies the six principal emotions that can be universally identified and associated with distinct facial expressions. These are: happiness, sadness, surprise, fear, anger, and disgust. The high level rules relating to each of the six expressions translates to a collection

of threshold rules that apply for a given expression. These rules are general in that they are valid for any subject.

The EPLT mechanism discussed in the previous section provides a mechanism for associating an Expression ID with subject-specific expression analysis rules and model-specific expression analysis rules. The above observation, namely that each of the six expressions is defined in terms of a collection of subject-independent threshold rules implies that there exists a collection of fixed ratio/threshold measurements that apply for a given expression. The result is that, for each of general expression, the associated classification threshold rule is general. If a given subject does exhibit a noticeably unique facial expression that the general rules do not apply for, they can be altered to suit the individual. This idea can be extended to the notion of adding arbitrary expressions with unique threshold rules to the EPLT. It is therefore not constrained to a standard set of expressions, unlike the MPEG-4 standard that defines 84 standard parameters.

5.5.2.1 Sample Classification

The number of expressions that can be classified is a function of the number of fiducial points that are tracked. Tracking only 4 points around a subject's mouth means that the following two expressions be classified reliably, namely: happiness/**smiling** and sadness/**frowning**. Tracking the eyebrows allows for the classification of **surprise**. As more fiducial points are tracked, so the number of expressions that can be identified increases.

The general rules for a `SMILE` or `FROWN` classification is illustrated in **Figure 5.7**. The rules used to classify the expression of a subject are illustrated. Given the initial fiducial positions, a smile can be identified if the mouth is wider and the mouth edges are raised. Alternatively, a frown is detected if the current mouth edges are lower than the original edges. The dashed lines illustrate how the mouth expands when smiling. An additional facial state that is general is the detection of whether the subject's mouth is open or not. The expression classification process is performed by comparing the positions of the fiducial points for the current `GlobalTemplate` to the original acquisition data. The data used to perform the classification are:

- `MouthWidth`: the width of the mouth, i.e. the magnitude of the line segment R_m-L_m;
- `MouthHeight`: the distance from the `NOSE` fiducial to the `Chin` fiducial;
- `MouthIntersection`: the position at which the `MouthWidth` line segment and the `MouthHeight` line segment intersect;
- `NoseCentre`: the distance from the `Nose` fiducial to `MouthIntersection`;

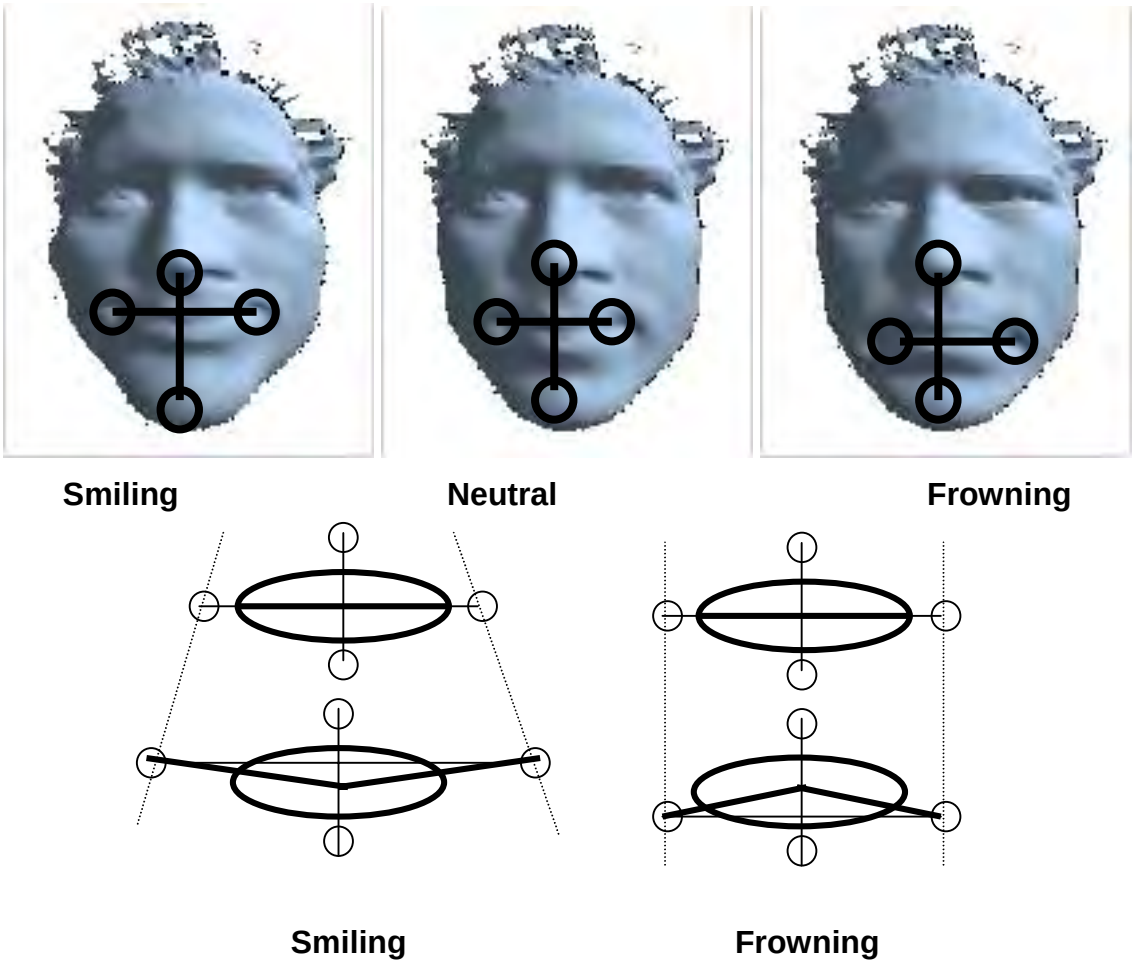


Figure 5.7 Expression Classification Rules.

- `ChinCentre`: the distance from the `Chin` fiducial to `MouthIntersection`.

Each of these components is computed every time a new `GlobalTemplate` is evaluated. The two sets of parameters that are initiated are for both the current `GlobalTemplate` and the original acquisition data. The final expression analysis compares these two sets of parameters to each other to classify the expression. The current `GlobalTemplate`'s parameters have prefix `Curr` i.e. `CurrMouthWidth`, `CurrMouthHeight` etc., while the original acquisition data's parameters are characterised by an `Orig` prefix (`OrigMouthHeight` and `OrigMouthWidth` etc.).

The rule-based evaluation occurs as follows:

```
IF (CurrNoseCentre – OrigNoseCentre < 0) THEN
    IF (CurrMouthWidth > OrigMouthWidth + WidthDelta) THEN
       Expression ← SMILING
       RETURN
    END IF
```

```
END IF
IF (CurrChinCentre < OrigChinCentre) THEN
    Expression ← FROWNING
    RETURN
END IF
Expression ← NEUTRAL
RETURN
```

From the above algorithm, it is evident that once an expression has been classified the evaluation ceases. Additionally, the system defaults to the `NEUTRAL` expression if none of the constraints for the other expressions can be satisfied.

The classification of whether the user's mouth is open or closed occurs as follows:

```
IF (CurrMouthHeight  > OrigMouthHeight + HeightDelta) THEN
     MouthState ← OPEN
ELSE
     MouthState ← CLOSED
END IF
```

The addition of a `WidthDelta` and `HeightDelta` respectively in the above algorithms ensures that some room for deviation and error correction is included as well. This aids in removing very erratic behaviour with respect to the expression identified; an expression is only identified if the fiducials move a minimum distance, thereby increasing the robustness of the selection process. The two deltas are typically computed as a fraction of the `OrigMouthWidth` and `OrigMouthHeight` parameters, for example:

```
WidthDelta ← OrigMouthWidth DIV 10
```

During the process of expression identification, the EPLT concept comes into play. With the pose estimation and expression identification complete, the conversion of each live video frame to a quaternion (and associated translation vector) and Expression ID has been achieved.

5.6 Experimental Results

5.6.1 Brute Force NCC Accuracy

This section illustrates the effects of increasing the size of the correlation window on the robustness of the basic NCC algorithm. The images in **Figure 3.5** are used as input to the

matching algorithm. The white dots highlighted in the left image are provided as input to the NCC algorithm, with the goal of finding the appropriate matches in the right image. A total of 16 points are used as input.

NCC Matching Results				
Size of Correlation Window	**No Matches**	**Correct Matches**	**False Matches**	**Time Taken**
2x2	0	5	13	6.24
4x4	1	14	3	22.192
6x6	1	15	2	50.385
8x8	1	15	2	88.626
16x16	8	7	3	356.428

From the above table it becomes evident that, as the correlation window size increases, a point is finally reached where the accuracy of the implementation begins to decline, i.e. many matches are found. For all correlation windows up to the 8x8 window, the number of false matches declines. The **False Matches** column represents the number of pixels in the search window that are classified as matches i.e. have a matching score that is more than the minimum threshold NCC score, but are not true matches. The **No Matches** column indicates the number of pixels that no matches have been found for. This is representative of matches that fall below the minimum threshold for a specific pixel, indicating a choice of minimum threshold that is too high. With a 16x16 window size, the number of pixels for which no matches is found matches, increases sharply (from 1 for the 8x8 correlation window to 8 for the 16x16 correlation window), the number of correct matches begins declining (from 15 down to 7). The false matches also begin to increase from 2 at 8x8, to 3 at 16x16. Additionally, one can see that, as the correlation window size increases, so the time taken to find the matches increases as well. The measurements in the above table have been obtained on a 133 MHz Intel Pentium with 82 MB of RAM, running Linux RedHat 4.2.

5.6.2 Tracking Performance

It is mentioned in section 5.3.1 that the parameters that determine the robustness and performance of the NCC tracker are: the frame size, the search window size and the correlation

window size. The experiments discussed in this section show that by minimising the frame size and the search window size, and by maximising the correlation window size, one can obtain robust and real-time tracking performance.

The term **search window parameter** is used in this section. It refers to a value by which the frame size is divided by to determine the search window size. As the search window parameter decreases or the frame size increases, so the search window size increases. This means that a search window parameter of 16 and a frame size of 256x192 will lead to a search window size of 32x24.

The following sections discuss the NCC tracker's performance as each of the three parameters is altered. This section concludes with a discussion on the selection of the parameter value for generating the most robust tracking results, given a minimum performance requirement of 10 frames per second. Complete measurements are tabulated in **Appendix A**; as mentioned in that section, all measurements have been performed using a 333MHz Intel Pentium II, running RedHat Linux 6.1 with 128 Mb RAM, and unless otherwise specified, have been taken over a 20 second time period. No other CPU-intensive software was running during the execution of the NCC tracker application.

5.6.2.1 Frame Size

An increase in the frame size will lead to a decrease in performance; the increase (x) will cause

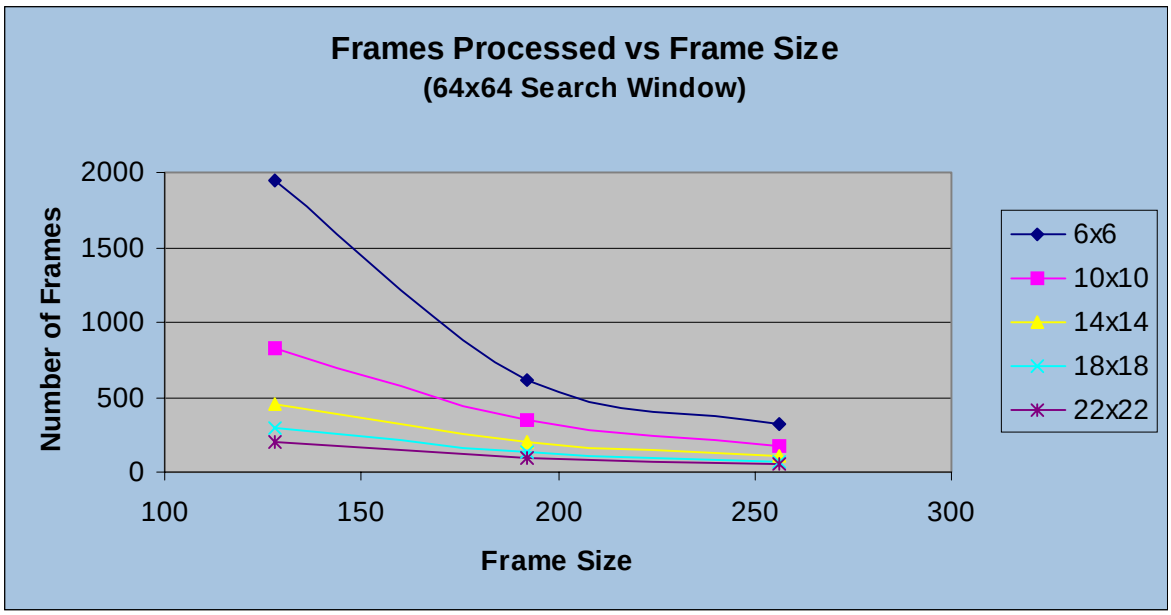


Chart 5.1 Number of frames Processed vs. Frame Size.

an x^2 decrease in performance overhead. This means that the inter-frame image coherency undergoes roughly a fourfold increase if the size is doubled i.e. about four times the number of frames will be processed. **Chart 5.1** illustrates the number of frames that are processed as the frame size is altered; each of the five curves represents the performance of the NCC algorithm with a given correlation window size.

The graph reveals an $1/n^2$ relationship between frames processed and the frame size for each correlation window size. This relationship repeatedly occurs for each of the different correlation window sizes. occurs for each of the different search window sizes, as is illustrated by the different graphs overlaid on the same chart. With a 10x10 correlation window, a decrease of the frame size from 256x256 to 128x128 causes an increase in the number of frames processed from 177 (8.85 fps) to 821 (41.05 fps). The search window parameter is 64 for this experiment. The performance gain is approximately 4.638.

5.6.2.2 Search Window Parameter

An increase in the search window parameter causes an increase in tracker performance; this is illustrated by **Chart 5.2**. Increasing the search window parameter from 32 to 64 causes an increase from 57 frames processed (2.85 fps) to 177 frames processed (8.85 fps) . The frame size is 256x256 and the correlation window size is 10x10. The performance gain is 3.106;

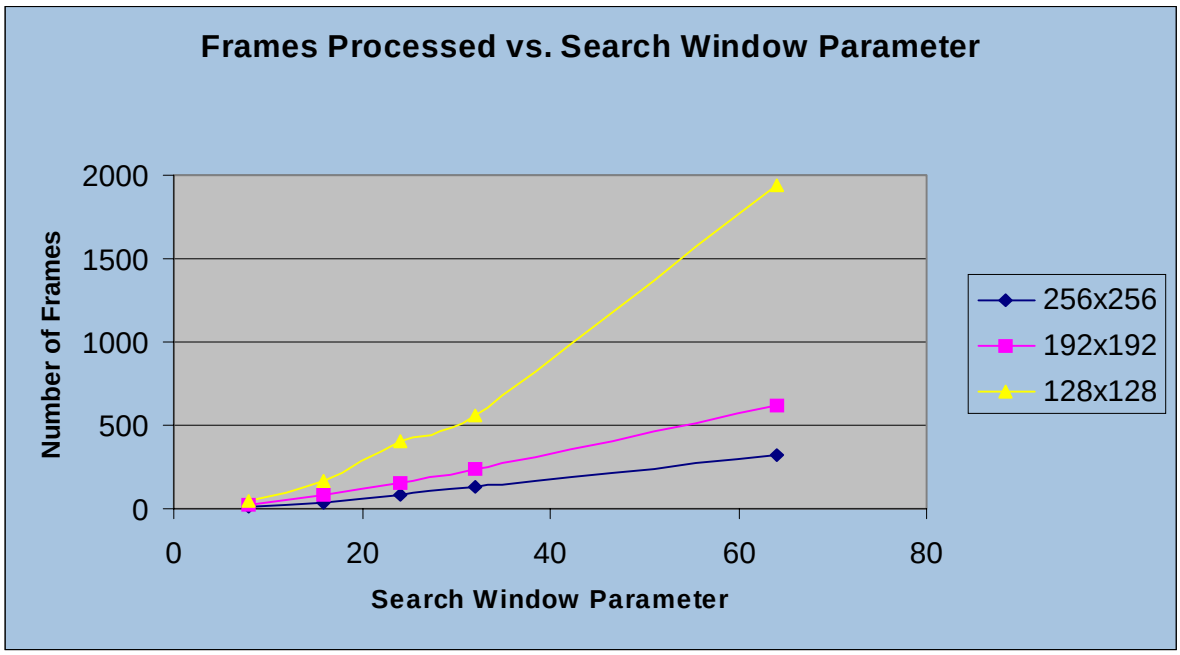


Chart 5.2 Number of Frames Processed vs. Search Window Parameter.

The relationship between the search window parameter and the number of frames that are processed by the tracker seems to be *n* for a specific frame size; as the search window parameter increases, the number of frames processed approximately double. This is supported by the measurements for the search window parameters with a frame of size 192x192. With a search window parameter of 192x192, the number of frames processed increases from 239 (search window parameter of 32) to 619 (search window parameter of 64).

As the frame size decreases, the marginal performance gain per search window parameter increases. For a search window parameter of 32, the number of frames processed increases from 127 (frame size of 192x192) to 239 (256x256). The relationship is n^2.

5.6.2.3 Correlation Window Size

The effect of a change in the correlation window size on tracking performance is depicted by **Chart 5.1**. Increasing the correlation window size decreases the performance; increasing the pixel window size from 6x6 to 14x14 causes a decrease in the performance from 322 frames processed to 107 frames processed. These measurements are associated with a frame size of 256x256 and a search window parameter of 64. The performance penalty is 3.009.

Using a 192x192 frame size, the change in tracker performance decreases from 619 frames processed (30.95 fps with a 6x6 correlation window) to 199 frames (9.95 fps with a 14x14 correlation window), given the same parameters as those used previously. The performance penalty is 3.111.

At 128x128, a similar change to the pixel window size leads to the following change in performance: 1946 frames processed (97.3 fps) to 453 frames processed (22.65 fps). The performance penalty is 4.296.

5.6.2.4 Performance vs. Robustness Tracker Parameter Selection

This section discusses the selection of tracker parameter values to achieve the goal of robust and real-time image-based tracking of a number of fiducial points. This translates into a problem of determining the combination of a minimum frame size/search window size and maximum correlation window that will facilitate robustness and speed. Choosing too small a search window size will translate to very fast performance but decreased tracking accuracy due to the reduction in the search space.

Simply choosing a small frame size results in a remarkable algorithmic speedup as described in section 5.6.2.1. In fact, it has shown the best performance enhancement (4.638 for reduction in frame size from 256x256 to 128x128) over the other two possible performance alterations, namely the search window parameter/size (performance gain of 3.106 at 256x256) and the correlation window size (performance penalty of 3.009 for increase of size from 6x6 to 14x14). The additional implication is that, using the search window parameter, the search window size is also decreased as the frame size decreases.

Decreasing the frame size implies that the size of the correlation window needs to be considered and, if necessary, adjusted as well. The disadvantage of a large correlation window with respect to the object being tracked has been illustrated in **Figure 5.2**. If the object being tracked becomes small due to the scaling of the frame, the tracker, although delivering much faster performance, will be less reliable. Further performance benefits are thus afforded by decreasing the correlation window size to achieve increased robustness. Unfortunately, too small a correlation window size also means that not enough of the background is included in the correlation window to facilitate accurate tracking.

The process of determining the appropriate values for the different parameters can be viewed from a different perspective: given a robust tracker such as the NCC tracker, the values of the parameters constrain the amount of motion that can be handled. This means that the selection of specific values for each parameter constrains how well the tracker is able to perform tracking and therefore, if robust tracking results are a pre-requisite, how fast a subject being tracked may move. Thus, given a very large frame size and a very small search window size, if the subject moves very slowly, he will be tracked very robustly. As he/she moves faster, so the tracker will tend to fail. Similarly, the given a small frame size and a large search window size, the user is able to move much faster than previously, with the tracker being able to track the subject robustly.

To determine the most appropriate tracker parameters for the purposes of fiducial tracking, a subject is instructed to look left and right, up and down and to smile and frown over a 20 second time period. The results of the tracking algorithm for this movement sequence are recorded for varying parameter values. The results are tabulated in **Appendix A**. The frame sizes evaluated are: 256x256, 192x192, 128x128 and 64x64. The search window parameters evaluated are: 64, 32., 28, 24, 16 and 8. The pixel window size parameters evaluated were: 3, 5, 7, 9 and 11. All possible combinations of these three parameters are evaluated and the following measurements taken, namely:

- The number of frames processed, to determine the speed of the tracker given specific parameters;
- The number of user-assisted corrections required, a subjective assessment of the robustness of the tracker for a given parameter combination.

The goal with this experiment is to choose parameter values that results in frame rates exceeding 10 fps (to facilitate near-real time tracking performance) and at the same time requiring minimal (but not necessarily no) user intervention. The following behaviour is observed:

- With a search window parameter of 64, the tracker is completed unreliable, requiring continuous user correction. Additionally, the tracker was completely unusable with a frame size value of 64x64;
- With a search window parameter of 32, more results that are reliable are observed, with user intervention decreasing over the previous parameter selection. The primary limiting factor with this set of parameter values is the frame rate: except for a frame size of 128x128, the frame rate falls below the minimum required frame rate. Additionally, with the frame size of 128x128, all combinations that are able to achieve more than the minimum required exhibit a high degree of user correction, namely 22 corrections for the (128x128, 32, 10) combination. Given the frame rate achieved with this combination (11.55 fps) it is found to be one of the better combinations. However, it suffers due to a problem of drifting, indicating one of two problems: insufficient calibration (as discussed in section 5.3.2), but more likely, too high a search window parameter (that led to a small search window size), thereby excessively reducing the NCC search space;
- The cut-off point based on the minimum 10fps manifested itself with the (128x128, 28, 10) combination. With only 4 corrections, and a frame rate of 9.95 fps (199 frames processed in 20 seconds), this combination appears the most favourable;
- While the remaining parameter combinations evaluated returned minimal errors in tracking (due in part to the small search parameter values, and therefore a large search window size), their performance falls below the minimum threshold set, with the frame rate falling from 4.1 fps for the (256x256, 24, 10) combination to 0.55 fps for the (256x256, 8, 6) parameter combination.

The final parameter combination thus chosen is a frame size of 128x128, a search window parameter of 28, and a correlation window size of 10x10.

The performance of the tracker is linear with respect to the number of fiducial points that are tracked. This is because each fiducial point is tracked independently of the next. The number of

times the tracking algorithm is executed is proportional to the number of fiducials that are tracked.

The selection of the above parameters thus means that robust image-based tracking of six fiducials is facilitated at a steady 10 fps.

5.6.3 Rotational Robustness

Figure 5.8 illustrates the benefits of the dynamic feedback loop employed by the NCC-based tracker. A part of a monitor is tracked as the CCD camera used is rotated through 180^0. As can be seen, the tracker manages to track the same point through the entire rotation. Failure occurs twice, but only because the camera is moved too quickly. This is evident in illustrations D and F. The tracking parameters used are as follows:

- frame size : 256 x 192
- search window size : 36 x 14
- correlation window size : 10 x 10

The large correlation window size and frame size also contribute to the sluggish performance and thus the tracking errors.

This experiment is not possible using traditional template-based tracking approach. Additionally, this experiment also illustrates the primary advantage of the NCC tracker, its

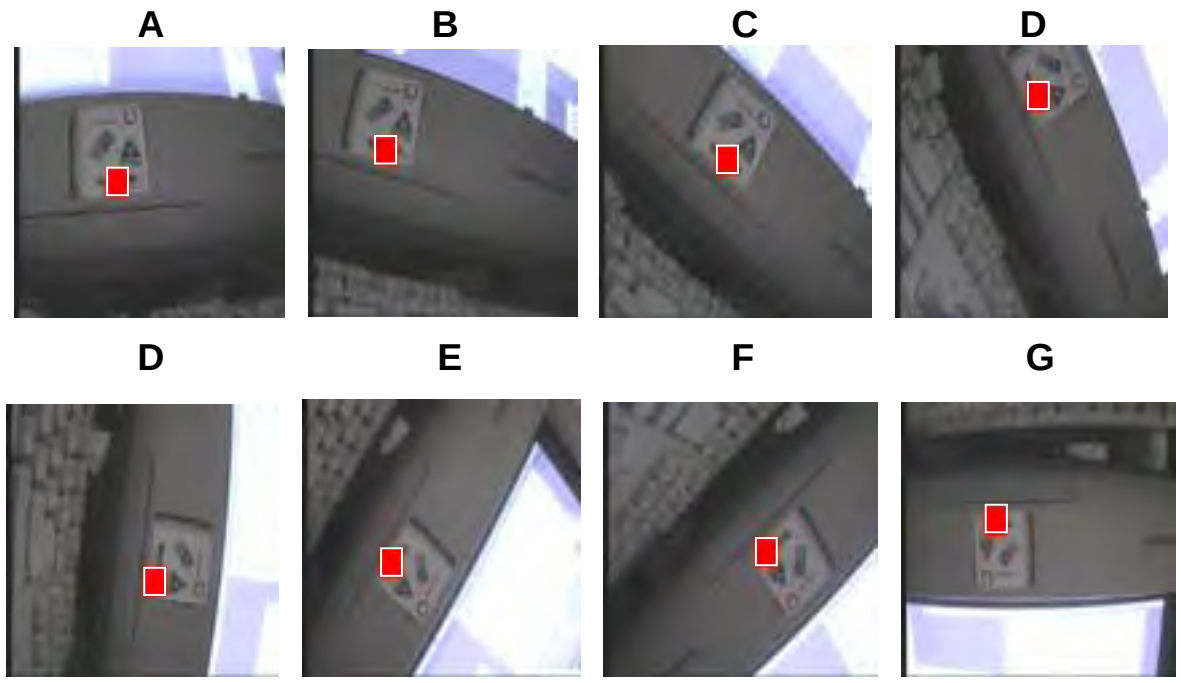


Figure 5.8 Results of the NCC Tracker with Rotation.

ability to adjust to changing environments. Provided the area surrounding the position being tracked is static, the algorithm is able to handle situations where the background to a scene is dynamic and changing.

5.6.4 Scaling Robustness

The robustness of the NCC tracker to scaling is illustrated in **Figure 5.9**. The experiment to test the proposed robustness involves the positioning of the fiducials with respect to the subject's face, as depicted in illustration A. Subsequently, the subject is instructed to move away from the camera for a period of time (illustrations B to F) and then move towards the camera again (illustrations G and H).

The tracking parameters for this experiment are:

- frame size : 256 x 192
- correlation window size : 14 x 14
- search window size : 36 x 14

If one compares the positions of the fiducials around the mouth in illustration B with those in illustration H, one can see that, although the positions are not exactly the same, they are approximately the correct areas. This error in tracking is a direct side-effect of the dynamic feedback loop mechanism around which the tracker is designed.

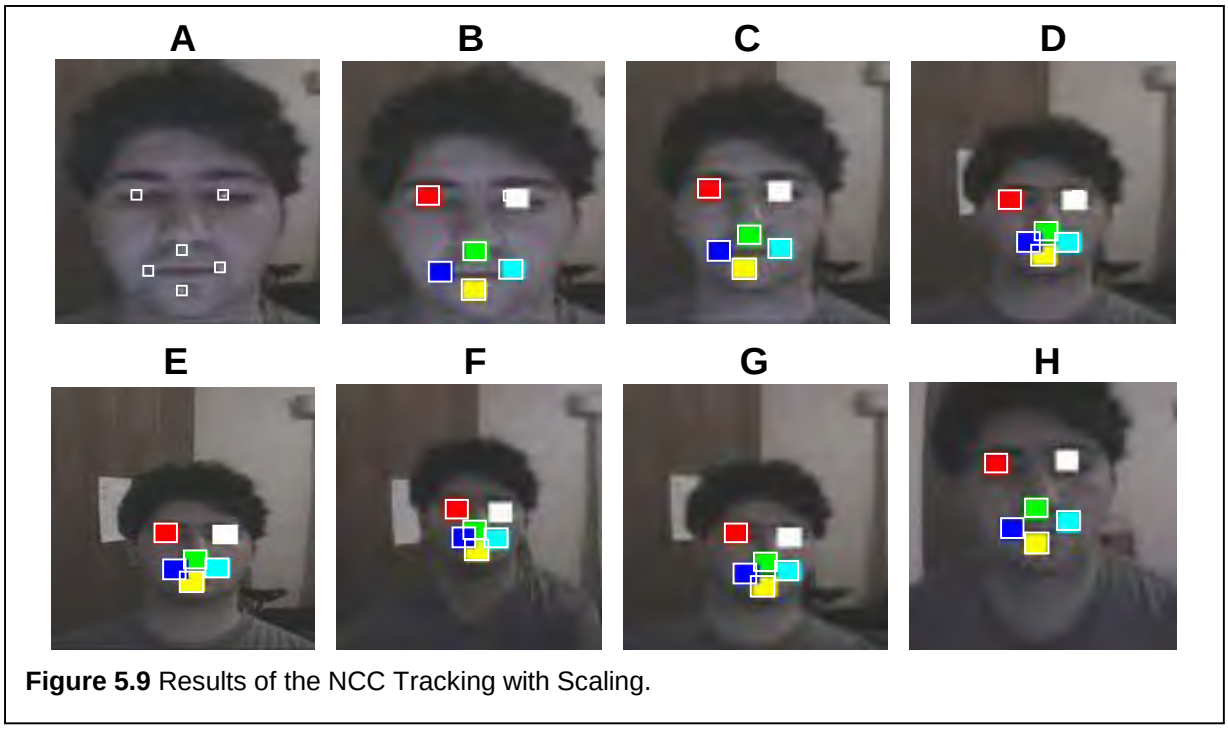


Figure 5.9 Results of the NCC Tracking with Scaling.

5.6.5 Pose Estimation

This section summarises results obtained from a number of experiments that have been performed to test the accuracy of the pose estimation calculations detailed in section 5.4. These experiments have been performed using the avatar depicted **Figure 5.10**. The NCC tracker has been instructed to capture a video stream from the frame buffer rather than from the CCD camera attached to the system. As the avatar is rotated, the true magnitude of rotation is compared to the computed rotation angle, thus facilitating an evaluation of the accuracy of the pose calculation. Illustration A depicts the avatar model that is captured, while illustration B show the fiducial points that are overlaid on top of the avatar for tracking. The NCC tracking parameters in all cases below were:

- frame size: 192 x 192
- search window size : 26x26
- correlation window size : 8 x 8

5.6.5.1 Roll Recovery

The recovery of the roll component involves rotating the avatar about the Z-axis, i.e. parallel to the image plane. The rotation occurred at 100 intervals from 0^0 to 90^0. The measurements obtained are depicted in **Chart 5.3**. The straight curve (entitled "True Average") depicts a truly accurate recovery. Both positive and negative measurements are depicted. In every case, the measured magnitude is typically 10^0 less than the actual measurement. As the magnitude of rotation approaches 90^0, so the difference between actual and measured rotation increases to approximately 20^0. Furthermore, because the rotation is parallel to the image plane and none of

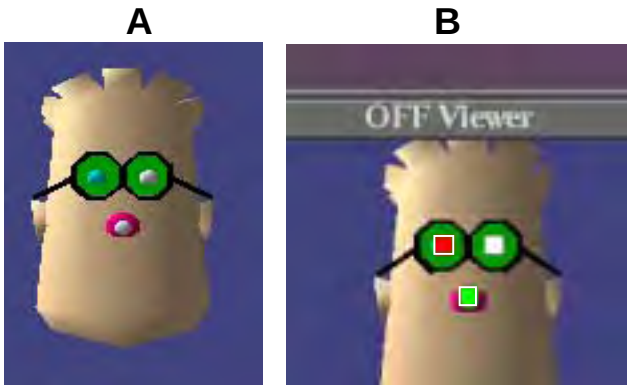


Figure 5.10 Pose Estimation Setup.

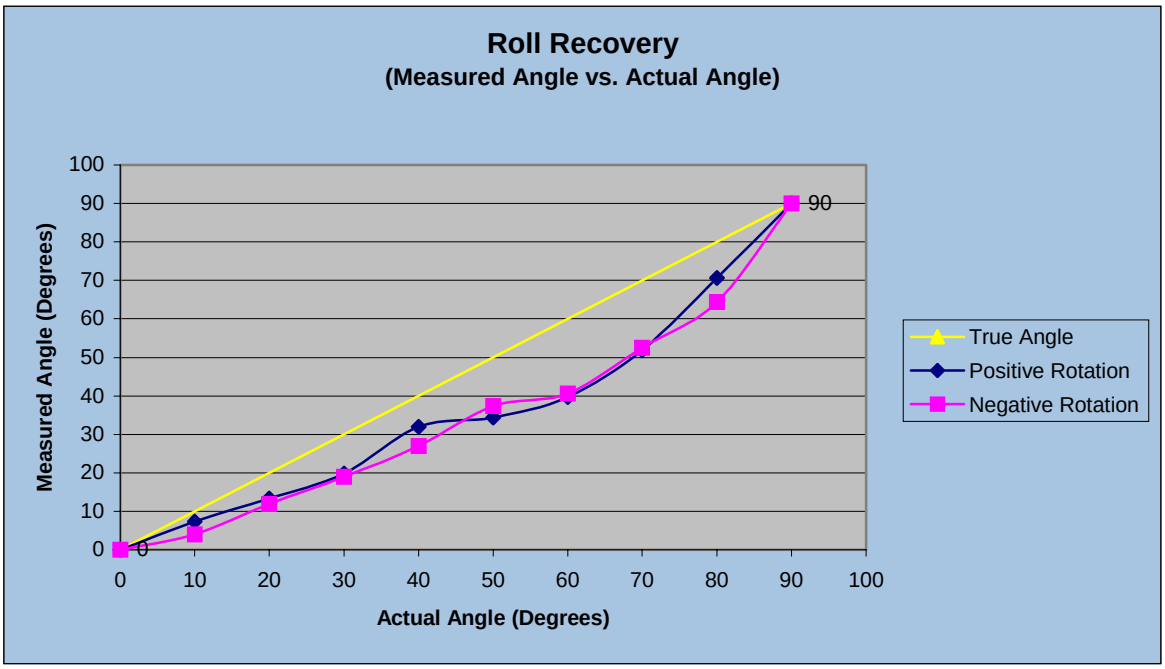


Chart 5.3 Accuracy of Roll Recovery.

the fiducials is ever occluded, measurements for every angle from 0^0 to 90^0 (at 10^0 intervals) have been obtained, and are part of the chart.

5.6.5.2 Yaw Recovery

Chart 5.4 depicts yaw measurements. For this experiment, the avatar is rotated about the Y-axis. The measured angle is initially very close to the true angle, but progressively becomes worse. The recovery proceeds until the head has rotated through 40^0, at which point the system fails due to occlusions, i.e. one of the eyes becomes occluded. For a true 40^0 rotation, an angle of 33.1^0 was recorded.

5.6.5.3 Pitch Recovery

The pitch recovery experiment involved rotating the avatar about the X-axis. The measurements obtained during this experiment are depicted in **Chart 5.5**. The **True Angle** curve represents correct angle recovery.

Only three measurements are depicted. The reason for this becomes evident if one evaluates the head rotation in **Figure 5.11**. Illustration A shows the head rotating backwards, while illustration C depicts the head nodding forward. This shows that X-axis does not go through the

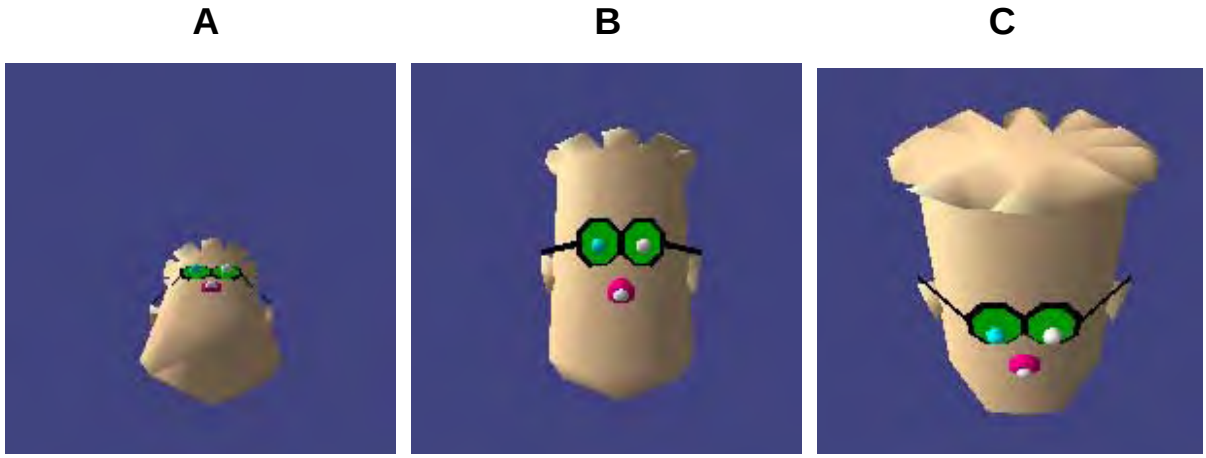


Figure 5.11 Pitch Recovery with off-centre X-Axis rotation.

centre of the head, but rather through the bottom of the model, namely the nape (the position at which the head connects to the rest of the body). This behaviour mimics real-world head movement.

As can be seen from **Chart 5.5**, it becomes evident that recovery of this orientation component is the most inaccurate, with tracking accuracy rapidly failing. The reason this is because of the rotation about the nape. As the head rotates backwards, so the fiducial points collectively move further away from the image plane. Similarly, the points collectively move closer towards the

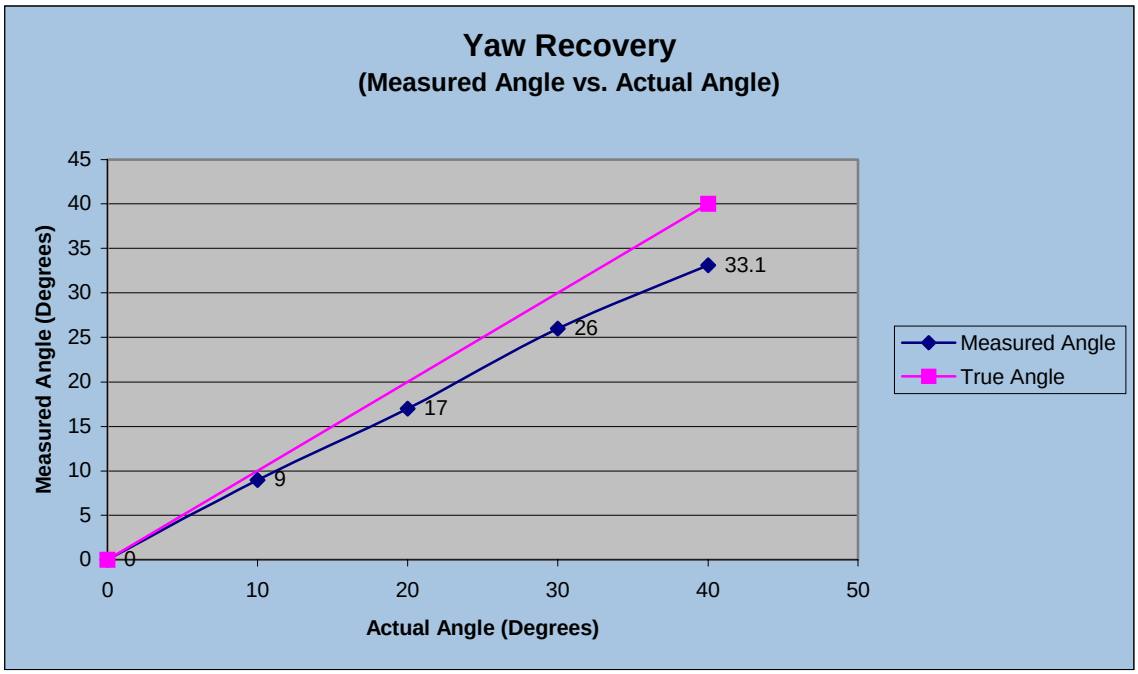


Chart 5.4 Accuracy of Yaw Recovery.

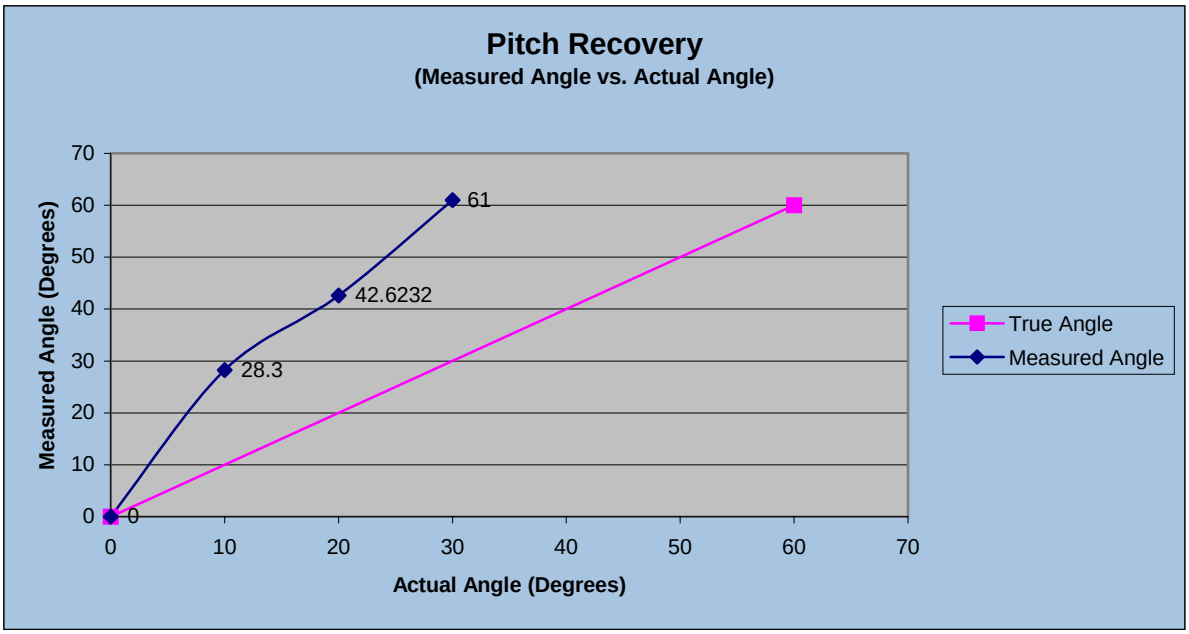


Chart 5.5 Accuracy of Pitch Recovery.

image plane as the head nods forward. As such, the measured angle rapidly deteriorates.

One possible solution to obtain better pitch recovery results is to re-centre the model as it rotates about the X-Axis, but the goal is to determine the accuracy of pitch recovery by mimicking real-world head movement in a controlled environment. As such, this has been avoided.

5.6.5.4 Scale Recovery

The accuracy of scale recovery is illustrated in **Chart 5.6**. These results are related to the illustrations depicted in **Figure 5.12**. For this experiment, the actual scale has not been reproduced exactly. The goal is to determine whether the algorithm is able to determine whether the subject moves toward or away from the image plane. Because the scale value is normalised using the acquisition data, the initial value, where the subject is in the same position his representation in the acquisition frame, its value is 1. This represents the acquisition position of the avatar.

As can be seen from **Chart 5.6**, the relationship between the between the measured scale and the actual scale is linear; as the avatar moves away from the camera, so both scale values decrease. The same behaviour is true for the opposite situation.

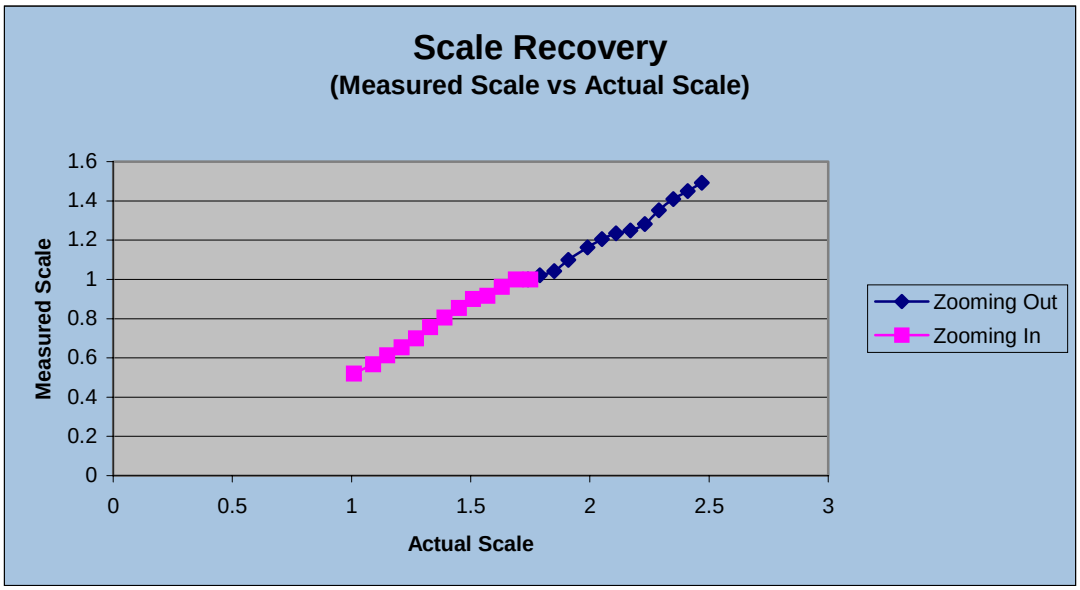


Chart 5.6 Accuracy of Scale Recovery.

It appears that the most fundamental problem with the current calculation is the problem of tracking errors due to scaling; if one compares illustration B to illustration E in **Figure 5.12**, this problem becomes evident.

The results are nonetheless acceptable for scaling purposes.

5.6.5.5 Summary

The results of the pose recovery implementation have been discussed. The roll recovery is the most accurate, partly because all fiducials are constantly visible. Problems of occlusions affecting yaw recovery, and off-centre head rotations, that adversely affect pitch calculations, lead to accuracy problems with the recovery of these two components. The maximum true angle that the pitch implementation is able to recovery is 30^0 (61^0 measured), while the yaw recovery fails at 40^0 with a measured/computed angle of 33.1^0. It has also been was also shown that the recovery of scale works and is able identify whether a subject's head is moving towards or away from the camera. It is a relative computation and is not meant to accurately recover the distance of the subject to the image plane.

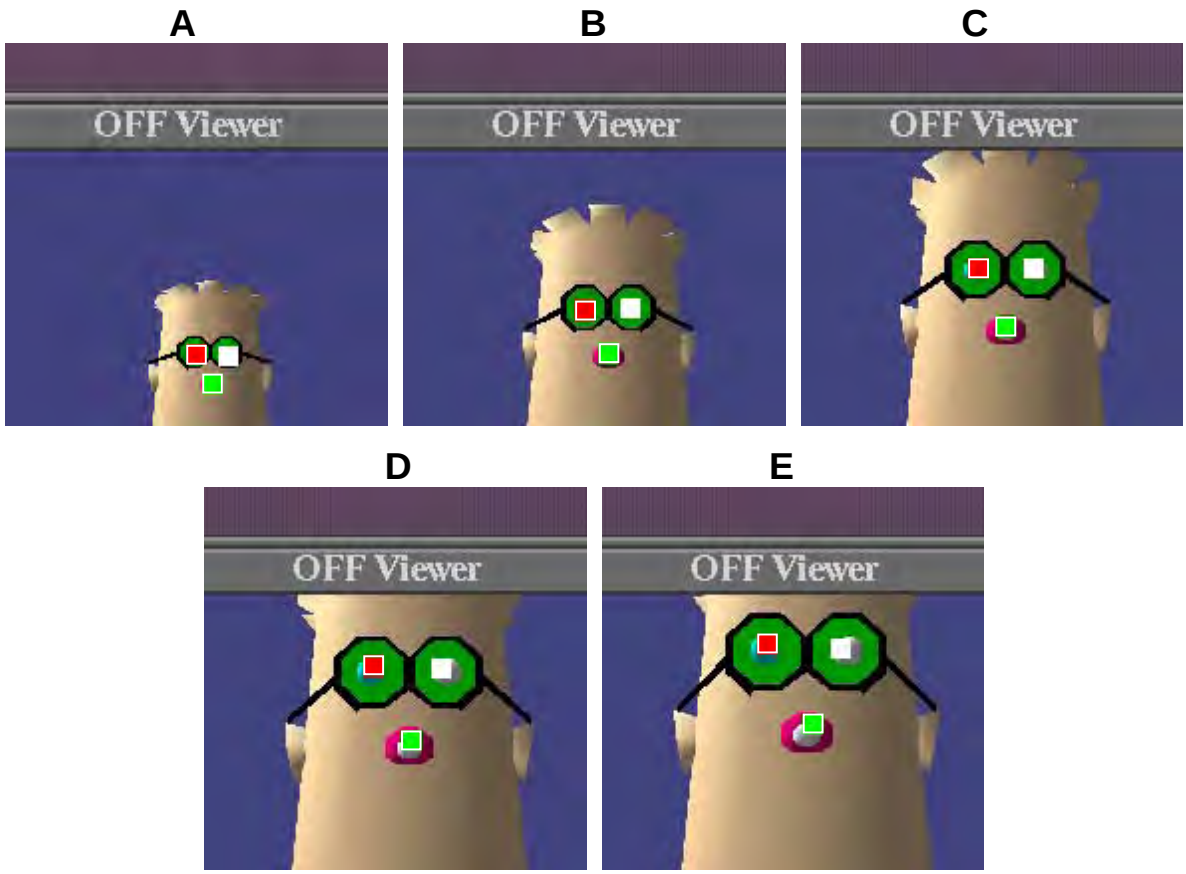


Figure 5.12 Scale Recovery Results.

5.6.6 Expression Analysis

To test the rule-based expression analysis process, a subject assumed different expressions during the tracking process, as illustrated in **Figure 5.13**. The following expressions are correctly identified:

- Illustration A = neutral
- Illustration B = frown
- Illustration C = open mouth
- Illustration D = smile

The tracking parameters for this experiment are as follows:

- frame size : 192 x 128
- correlation window size : 12 x 12
- search window size : 13 x 9

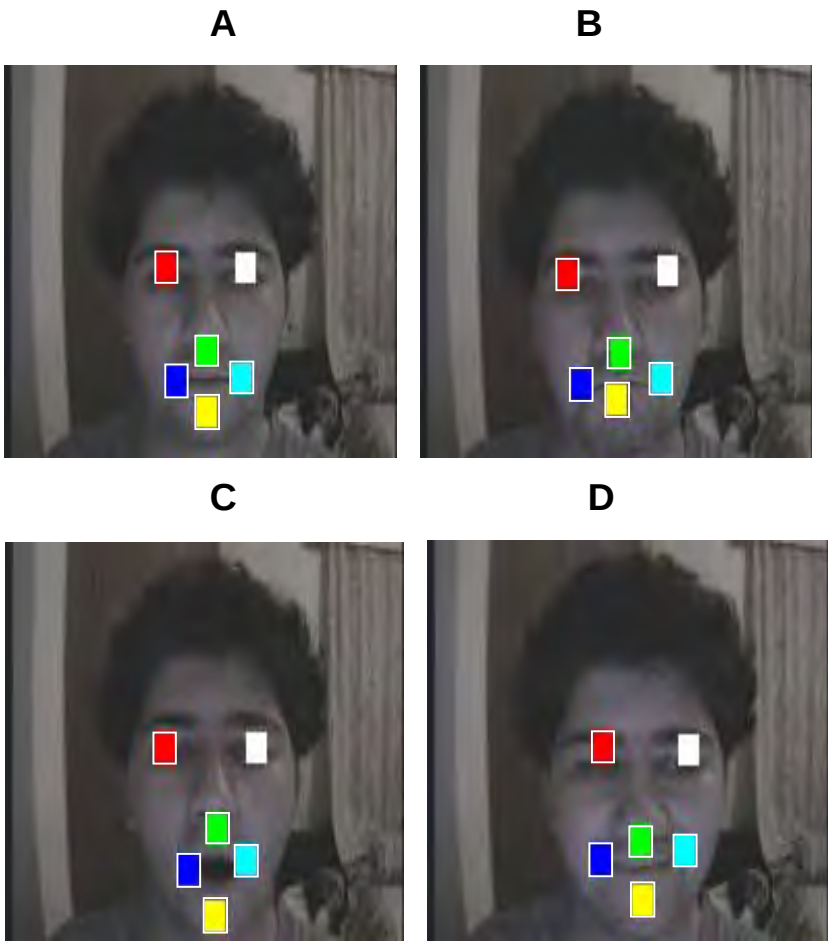


Figure 5.13 Expression Classification Results.

5.6.7 Overall Performance

It is found that the overhead the pose estimation and expression analysis implementations add to the overall performance of the NCC tracker is negligible. This arises because of two factors:

- The simplicity of the pose estimation implementation,
- A rule-based expression analysis that is currently able to classify only two expressions, and whether the subject's mouth is open or closed.

Depending on the enhancements made to these two aspects, the performance could be considerably affected.

5.6.8 Compression

The perceived benefit of model-based video/image coding systems is the elimination of the need to transmit any form of video across the network, thus leading to a couple of orders of magnitude reduction in the amount of bandwidth required in the final implementation. This benefit is feasible if one considers the compression example below.

The process of encoding takes as input a live video stream and generates as output a quaternion for controlling the avatar's position and orientation, as well as an integer representing the expression that has been identified. Given that a 100x100 uncompressed 8 bit video frame, represents about 30Kb of data, the encoding method transforms this into about 25 bytes of data (5 bytes for the Avatar ID, about 16 bytes for position and orientation, and 4 bytes for the Expression ID). This represents approximately a 1000-fold reduction in bandwidth requirements of the final application per analysed frame of video. This translates to approximately 250 bytes per second (2 Kbps), given the 10 fps minimum required frame rate. The bandwidth requirements are thus directly affected by the performance of the NCC tracker. The relationship is linear: given a tracking performance of 30 fps, the bandwidth requirements would increase 3-fold to 750 bytes per second i.e. 6 Kbps. This measurement is consistent with Eisert *et al.*'s [15] reported bandwidth usage of only 1 Kbps (see section 2.3.1). Their improved performance measurement presumably arises from the absence of transmission of the Avatar ID, and pose and orientation information per video frame processed in the data stream. Omitting the Avatar ID, bandwidth utilised decreases to 4.8 Kbps given a 30 fps processing rate. Using a byte to represent the Expression ID instead of an integer, this value falls to 1.36 Kbps at 30 fps.

This technique is suited to very low-bandwidth networking conditions, such as modem/dialup connections. The latency bottleneck has been moved from the network to the encoder.

5.7 Improvements

5.7.1 NCC Drifting

The most fundamental problem with the NCC tracking algorithm is that of drifting. This problem occurs as a side effect of the dynamic feedback loop employed whereby the results of the NCC tracking are employed as input to the tracking process. Given that the algorithm is required, for example, track a subject's eyebrows, the tracker tends to drift from its desired tracking location.

The reason for this is the inherent symmetry that exists with straight lines. This means that, given a desired pixel to track and a correlation window area that is divided between two regions, such as with the middle of the eyebrow where one half of the correlation window represents the skin-coloured forehead and the other half represents the eyebrow, the NCC evaluation will find numerous positions along the eyebrow returning similar correlation values. As such, the position of the desired point will begin drifting. The drifting is manifested specifically by the selection of a pixel that is a match according to the NCC evaluation and is used as input on the next iteration of the NCC tracking.

The obvious way to avoid this problem is to track points that do not exhibit a symmetrical intensity with respect to the correlation window. In practice, the only fiducials that have exhibited these problems have been the eyebrows. In those situations, the centre of the eyebrow was selected for tracking, subsequent to which, the fiducials began to drift. Additionally, correlation window that encompasses a sufficiently interesting amount of background to break the symmetry, can be utilised.

These makeshift solutions do not solve the inherent problem. A possible enhancement of the NCC algorithm to include the effects of curvature of edges in the final evaluation is required.

5.7.2 Pose Estimation

The implementation of a more accurate pose recovery solution than the one presented above, is the most important improvement required. The current pose recovery mechanism independently computes the roll, pitch and yaw components of the pose of a subject. There are two fundamental problems underlying this implementation, namely:

- Each orientation component is computed independently of the next. This is similar to the approach discussed by Horprasert *et al.* [30]. Unfortunately, it is suspected that the components are not independent of one another;
- Perspective effects have not been considered as part of the computation, leading to inaccurate results.

A possible solution that promises to address both of these problems lies in solving the following equation:

$$p' = Ap \tag{11}$$

where:

- *A* represents a transformation matrix composed of a rotation(*R*), followed by a translation (*T*) and then finally a perspective (*P*) transformation;
- *p* represents the 3D point to which transformation matrix is applied to; and
- *p'* represents the projection of p onto the image plane.

The goal is to solve *A*. To this end, the three fiducials used to perform pose estimation, are used. Additionally, two positions of these fiducials are available, namely the position during acquisition (*p*), and the current position (*p'*). *A* can be used as a representation of the head transformation, or decomposed into the parameters specifying the original rotation, translation and projection.

5.7.3 Automatic Fiducial Point Location

The use of face bunch graphs in conjunction with neural networks to automate the location and placement of fiducial points on a subject's face is a very desirable extension.

5.7.4 Combining Pose Recovery and Expression Analysis

The current implementation does not attempt to classify expressions correctly as the head assumes different pose. Because all threshold measurements are relative, movements that are parallel to the image plane (such as rotations about the z-plane) will result in accurate expression identification. The remaining possible motions, however, will lead to incorrect classifications. As fiducials are tracked, the tracking results must be applied to a virtual fiducial graph, which can then be used to perform the expression classification. In essence, the classification process must be applied to corrected fiducial data.

5.7.5 Background Elimination

A promising way of improving the performance of the algorithm is to make use of a background elimination technique as discussed in 3.4.3. A pre-defined background template is subtracted from every incoming video stream. In theory, the result of this kind of operation in a videoconferencing situation would be a frame of video with the background subtracted. In addition to minimising the search space, this approach also promises to simplify head tracking in terms of estimating the following parameters:

- Position: given that only the head is in the video frame, it is possible to determine its position very quickly;
- Scaling: taking the rough elliptical shape of the head, it is possible to determine how far the user is from the camera very easily, simply by evaluating the size of the ellipse representing the head.
- Rotation about the Z-Axis: comparing the current head-forming ellipse to a acquisition-type ellipse would allow for very quick and accurate in-plane rotation calculation.

One additional benefit is that of prediction. Since the head is the only remaining part of the image that must be tracked and facial features remain relatively fixed with respect to the head, one can pre-determine the best place to start looking for specific features, and thus add intelligence to the searching process.

The process of background elimination has performance penalty $O(n^2)$, because it performs per-pixel calculations and subtractions. It should nonetheless return better tracking results because it constrains the search space to the region of interest, namely the face. An evaluation is required to determine the effects of background elimination on the overall performance of the tracker.

A sample background elimination implementation is provided below. Results are discussed and problems identified.

5.7.5.1 Sample Implementation

At initialisation, a template is generated. This typically involves capturing a number of images of a static background. To ensure statistical normality, forty images are captured. Rekimoto [54] suggests using the YUV/YIQ colour space to perform background elimination. Each RGB image captured by the capture software is converted to an equivalent YIQ representation through the following matrix conversion (Vishnevsky [67]):

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

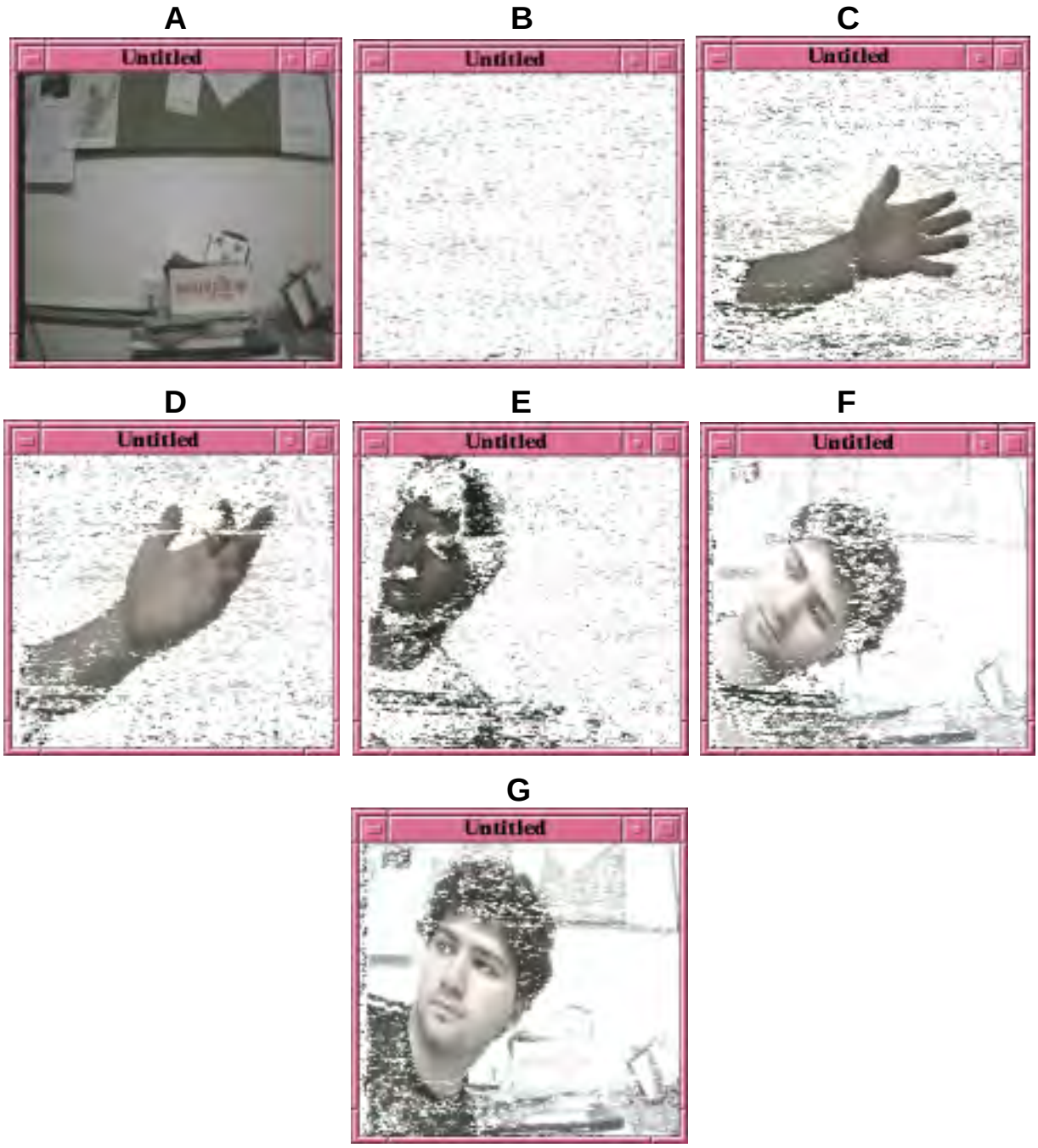


Figure 5.14 Background Elimination Results.

where: the Y component represents the perceived luminance, and the I and Q components represent the actual colour and some of the luminance information (Vishnevsky [67]). The above conversion is performed at a pixel level. Subsequently to this conversion for each image, the template information is computed. The template information consists of a template image and a list of standard deviations for each pixel. The template image simply represents the average of the 40 YIQ values for each pixel in the image.

Because the *Y* component represents luminance in the YIQ equation, only the *I* and *Q* components are used during background elimination. Each pixel in the current image is compared to the associated pixel in the template image according to the following algorithm:

```
A ← YIQ (CP.RGB)
IF (A.I - B.I <= 2 * SD.I) AND (A.Q - B.Q <= 2 * SD.Q) THEN
   CP ← Background Colour
END IF
```

where:

- `CP` represents the current pixel being evaluated;
- `YIQ(CP.RGB)` represents the conversion of the current pixel's `RGB` representation to an equivalent `YIQ` representation using the previous conversion matrix;
- `A` represents the `YIQ` representation `CP`;
- `B` represents the template's `YIQ` representation for the `A`'s position in the template; and
- `SD` represents `B`'s standard deviation.

5.7.5.2 Results

Figure 5.14 illustrates results of a background elimination implementation based on the above algorithm. Illustration A depicts a sample background image that forms part of the template. Illustration B shows a frame of video of the background with background elimination enabled. Illustrations C and D depict background-eliminated frames containing an arm. Illustrations E, F and G depict the elimination of the background from a frame of video containing a subject’s head. Comparing E to F and G, it becomes evident that the scene becomes brighter as the subject moves further into the centre of the image. This shift in brightness is attributed to the application of the CCD camera’s white balancing to attempt to maintain a fixed level of brightness in the scene. As one can see, this adversely affects background elimination, with more of the background filtering through to illustrations F and G. The results are obtained using a 333MHz Intel Pentium II with 128Mb RAM running Linux RedHat 6.1.

5.7.5.3 Problems

Real world testing of the above implementation has shown that this algorithm is susceptible to a number of problems.

The first problem, namely white-balancing, has been introduced briefly in the previous section. This term refers to camera's internal auto-correction system. This occurs when a camera attempts to maintain a certain level of brightness in a video sequence. If a very bright object is introduced to the system, the system attempts to maintain a fixed level of brightness by decreasing the brightness as a whole in the entire image. The result with real-world use of a CCD camera is that as a subject moves around, the camera auto-corrects to maintain a fixed level of brightness. One possible solution to this problem is to perform image normalisation. This involves dividing each of the R, G and B components by the (R+G+B) sum to obtain a normalised value for each component.

The second problem is one of noise. It may occur that pixels that are part of the head are removed. This is the case when parts of the face are almost the same colour as the background. Although the NCC tracker is robust, such situations could present tracking problems. Ways must be found to overcome this situation as well.

A noise reduction filter is also required to remove pixels in the image that do not form contiguous blocks.

5.7.6 Virtual Conferencing with GSM Networks

Du *et al.* [13] discuss the possibilities of video transmission using Global System for Mobile communications (GSM) networks. The four transmission modes supported by GSM networks (as reported by Du *et al.* [13]) are:

- Packet Data on Signalling channels service (PDS), which supports a transmission rate of between 0.6 and 9.2 Kbps;
- Short message service (SMS) provides support for short messages;
- High Speed Circuit Switched Data Services (HSCSDS) - supports up to 8 synchronous or asynchronous channels. It is "intended expressly for data transmissions". Each channel can transmit a maximum of 13Kbps, translating to a full 110 Kbps by utilising all 8 channels;
- General Packet Radio Service (GPRS) - supports up to a maximum of 170 Kbps.

As can be seen by the above configurations both HSCSDS and GPRS offer transmission capabilities far exceeding the requirements of the system (6 Kbps transmission rate quoted in section 5.6.7), with PDS appearing to be the most suitable transmission mode. An interesting extension would be to use HSCSDS to facilitate a multiple conferencing-type situation. Full use

could be made of all 8 supported channels to interconnect up to eight encoders/decoders to one another.

5.8 Summary

This chapter details the development of a real-time image-based tracking system based on the Normalised Cross Correlation mechanism. The use of NCC is beneficial, because it is a low-cost solution, i.e. does not require electromagnetic trackers. However, it suffers (as with other optical flow techniques mentioned previously in section 3.4.1.3) from occlusional discontinuities and very fast motion. Currently, the NCC tracker implementation does not fare very well with a high amount of motion. The algorithm can be extended to adopt a more hierarchical approach to matching. This would improve tracking results with respect to the user moving around a lot in the scene. Occlusions are a source of concern as well. The current implementation does not address these problems. As such, if the user rotates his head too much and a tracked point is occluded, the tracker will register a false match for that point and continue tracking that point.

Fiducial points are used as input to the tracking process. Six fiducial points are grouped together to form a fiducial graph. The choice of these points thus facilitates a subject-independent face specification, a factor that is used to perform pose estimation and expression analysis later. This facilitates the extraction of pose information and expression identification. As discussed in section 5.4, the pose estimation process involves triangulation of the nose, left eyebrow and right eyebrow fiducials to determine a final 6-DOF Quaternion representing the subject's pose. The translation vector specifying the position of the user is estimated by utilising the movement of the user's head parallel to the image plane, as well as a normalised scaling factor to determine the distance of the head to the camera. The scaling factor is determined using a ratio approach viz. the current over the original. The problems that plague traditional analytical perspective solutions to pose estimation, namely ambiguity, are avoided by adding a number of intelligent observations to the pose estimation process. This involves utilising inherent face symmetry to determine which way the head is rotated.

The expression identification process as defined in section involves the rule-based evaluation of the fiducials surrounding the subject's mouth. Once an expression has been identified, the process is halted. Based on fiducial information obtained initially, thresholds for, smiling (mouth width and mouth height vs. mouth width intersection) and frowning (mouth height vs. mouth width intersection) are utilised.

The Expression Parameter Lookup Table (EPLT) is introduced in section 5.5.1 as a mechanism to facilitate a mapping between subject-specific expression analysis and model-specific expression generation rules. For every subject/model combination, the generic EPLT (consisting only of a list of generic expressions and associated Expression IDs supported by the system) is enhanced by the addition of subject-specific expression analysis rules (for threshold classification) and avatar-specific expression generation parameters. For each of the universal expressions, there exists a collection of subject-independent threshold combinations that define the expression. The extensibility of the EPLT is discussed in section 5.5.2. An expression can be added to the EPLT provided its threshold rules are defined. Additionally, existing rules that do not quite apply to a given subject can be altered to suit the individual. The result is that only the Expression ID is transferred to the decoder for each expression generated.

The goals of the NCC tracker are identified in section 5.6.2.4 as the robust and real-time tracking of fiducial points in a video sequence. The factors that are found to affect the NCC tracker's performance are: the frame size, the search window size and the correlation window size. The final selection of the appropriate size of each of these parameters in section 5.6.2.4 is based on the evaluation of combinations of the three parameters, each with different values. Only combinations that are perceived to be robust (requiring minimal user assistance during tracking) and which manage a tracking performance exceeding 10 frames per second are considered. The final combination found to be robust and to achieve the minimum desired frame rate is: 128x96, 28, 14x14 (frame size, search window parameter, correlation window size).

The results of the recovery of each component of the subject pose are discussed in section 5.6.5. The experiment involves the tracking of an avatar to compare the true scale and angle of rotation for each orientation component to the recovered, or measured, angle. It is shown that the maximum true angle the pitch implementation is able to recovery is 30^0 (61^0 measured), while the yaw recovery fails at 40^0 with a measured/computed angle of 33.1^0. The roll recovery is the best, with computed angles being returned for a series of rotations between 0^0 and 90^0. It is shown in section 5.6.5.4 that the scale recovery implementation is able to determine a relative distance of the subject to the image plane.

Section 5.6.8 discusses compression issues relating to the encoding process. The encoder manages to achieve a compression rate (and subsequently a reduction in bandwidth requirements) of 3000 to 1 per video frame processed, from approximately 32kb for a 100x100 uncompressed frame of video, to about 25 bytes of encoder data. This translates to a transmission rate of approximately 2 Kbps given an NCC processing speed of 10fps. The transmission rate is linearly related to the performance of the NCC tracker.

The problem of drift is discussed in section 5.7.1. It occurs as a side effect of the dynamic feedback loop employed, where the results of the NCC tracking are employed as input to the tracking process again. It manifests itself when a point to be tracked has an associated correlation window that is near symmetrical in nature, i.e. the point being tracked lies between two very distinct and symmetrical intensity areas. An enhancement to the NCC tracker is required to deal with this problem. Possible temporary solutions include the use of a larger correlation window to break the symmetry and thereby cover more of the image, and the tracking of fiducials that do not possess the symmetrical characteristic.

Accurate pose recovery is identified in section 5.7.2 as the most fundamental problem with the current pose estimation implementation. The isolated computation of each pose component (roll, pitch and yaw), along with the avoidance of perspective affects is identified as the fundamental problem with the approach. A proposed improvement involves solving a transformation matrix (consisting of a rotation, translation and perspective transformation) to determine the appropriate orientation parameters.

A possible enhancement to tracking that utilises background elimination is discussed in section 5.7.5. The algorithm is based on the assumption that a static background is used and is subsequently subtracted from the current video frame to give a frame containing only the object of interest, in this case, a subject. It promises to reduce the search space the NCC tracker has to evaluate, but suffers because it performs per-pixel operations, and thus has performance $O(n^2)$. Other problems that are identified in section 5.7.5.3 are: white balancing and noise.

The possibility of virtual conferencing over GSM networks has also been introduced in section 5.7.6. This technology could be used to interconnect up to eight people in a single conferencing session.

5.8.1 Contributions

The most important contribution presented in this chapter is the development of a near real-time NCC-based tracking system. The tracking mechanism is completely unobtrusive, avoiding the need to attach markers to the subject’s face. A user-assisted feedback loop and caching mechanism is employed to overcome tracking problems due to occlusional discontinuities.

The further advantage of the NCC based tracker is that it is very robust. It has been shown that the tracking mechanism is invariant to scaling and in-plane rotations. This is attributed to the dynamic nature of the feedback loop.

The introduction of a concept of an Expression Parameter Lookup Table has led to a system that is able to facilitate an independent mapping (to an Expression ID) between the subject-specific expression analysis rules and the model-specific expression generation rules. This means that only the Expression ID need be transmitted to the decoder for the purposes of expression generation.

A transmission of 2 Kbps has been achieved. It has been shown that this can be reduced to approximately 1.3 Kbps by avoiding the continual transmission of the avatar's name and using a byte to represent the expression ID.

Chapter 6 - Decoding

6.1 Introduction

This chapter discusses the decoding aspect of the component chain. The previous chapter introduced the concept of an Expression Parameter Lookup Table, the function of which is to provide a mapping between the expression analysis phase and the process of expression generation. In addition to a discussion on expression generation, the decoding tasks of avatar movement and expression management are also discussed as well.

The notion of an expression editor is introduced. Its function is to define the model-specific expression generation parameters for each entry in the EPLT.

6.2 Design

The decoding process (as illustrated in **Figure 2.2**) consists of a number of subcomponents, namely: Avatar Movement, Expression Generation and Expression Management.

The control data for avatar movement, as determined during pose estimation by the encoding process, involves the positioning of the avatar in the virtual world. It is assumed that the underlying Virtual Reality system, upon which the implementation of the decoder is based, offers the functionality of updating the position and orientation of the object on-the-fly.

The process of expression generation is based on a notion similar to MPA and FACS-based systems. It is achieved by applying a combination of deformations to the different parts of the polyhedral mesh representing the avatar. Like FACS, it is assumed that every expression can be decomposed into a number of more simple facial movements.

A vertex interpolation deformation (VID) mechanism was discussed earlier in section 3.7.2.2 (see equation 10). It was mentioned in that section that deformation is controlled by a number of parameters. These parameters are repeated here:

- K = direction of muscle contraction (`1` denotes a contraction while `-1` represents an expansion);
- P_o = source force; this indicates the position where the deformation force originates;
- P_d = point of target force; this value represents where the force ends.
- R = radius of influence; Hung *et al.* [31] recommend that the radius cover approximately the same area as a bounding box surrounding the vertices being deformed;

- t = time value varying between 0 (no deformation) and 1 (complete deformation);

The VID approach allows deformations to be repeated, with the same results being achieved for specific deformation parameters every time.

The VID approach has been chosen as the underlying implementation because (as will be shown) it allows for real-time deformations of the avatar model. The guarantees offered by the more rigorous and standard FFD approaches, such as C^1 and C^2 surface continuity during deformation (amongst other things), are avoided in favour of the performance offered by vertex interpolation systems. Additionally, to guarantee surface continuity of the deforming object, it is assumed that the polyhedral mesh being deformed has a high degree of vertex saturation, i.e. that there are many vertices in close proximity to one another. This translates into ensuring that the radius of influence is large relative to the distance between any two vertices in the polygon mesh. **Figure 6.1** illustrates the problem of surface continuity with the VID approach given a low degree of vertex saturation and a radius of influence that is small relative to the distance between each vertex pair. The shape is deformed at vertex V with a radius of influence R. The result is that all edges connected to vertex V will begin moving to the target deformation position. Since R is so small relative to the shape, all remaining polygons (A, B, C, D, E and F) will remain undeformed. The result is a marked discontinuity appearing in the surface of the deforming object. While C^0 surface continuity is maintained, C^1 and C^2 continuity is broken. Given this situation, traditional algorithms begin introducing vertices to the polygon structure to avoid the

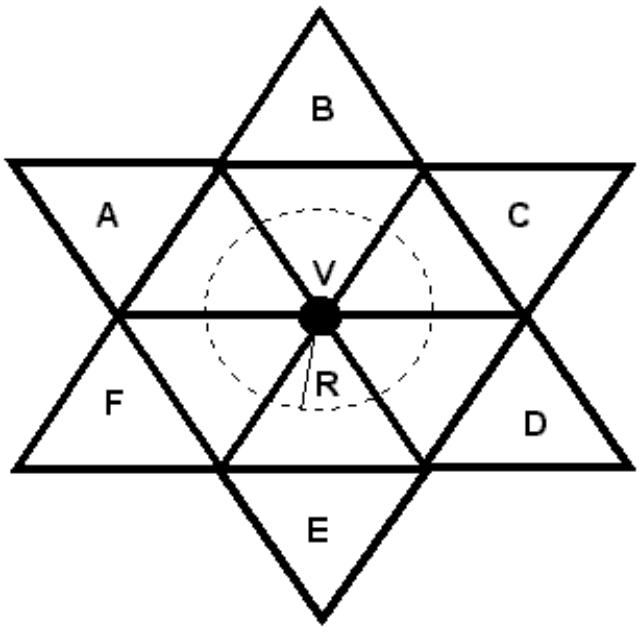


Figure 6.1 Problematic VID Situation.

discontinuity.

A further benefit of the VID approach is that the level of control offered by the time parameter in the VID equation is beneficial in terms of controlling the speed at which the expression generation takes place. This feature facilitates a simple mechanism for controlling avatar expression generation.

The EPLT, as discussed in section 5.5.1, implies that each avatar to be used with the final system, the lookup table will contain expression generation parameters for each expression it contains. This translates to a list of deformation parameters that, when applied to the avatar, will result in an expression being generated. To facilitate the avatar-specific generation of deformation parameters for each expression in the lookup table, an expression editing system is required. In addition to allowing expressions to be scripted on a per-model basis, this system is used to determine which deformations to apply in which order. The expression editing system allows individual deformations to be applied to the avatar. This maps very appropriately to the concept of expression subdivision (the classification of an expression as a collection of indivisible facial movements).

A three-tier solution to expression editing is employed, namely:

- **Single Deformation Management** for determining the deformation parameters for a single deformation;
- **Multiple Deformation Management** to facilitate the composition of expressions using a playlist paradigm; and
- **Avatar Expression Management** that allows each EPLT model-specific expression to be tested at a high level. The selection of different expressions and the generation of a selected expression at a high level have an added benefit for the later implementation of the entire system. When a decoder receives a request for a specific expression to be generated, the implementation that is used for this tier is used to force expression generation at a high level.

This approach is very similar to Lee *et al.*'s [39] four-tier mechanism for specifying expressions, as described in section 3.7.1. A video-editing paradigm is employed, whereby an expression is built up using a playlist of deformations; once the appropriate parameters for a specific deformation have been decided upon, the deformation is added to a list of deformations. As this list is built up, so the expression is defined.

Expression management is handled using a weighting system, whereby, given that an expression is being generated and a request for a new expression is received, the weighting

system gradually transfers the deformation subsystem from generating the current expression to generating the new expression.

6.3 Expression Generation

This section provides a high-level description of the tools that have been developed to allow for simple and model-independent expression generation.

6.3.1 Lower Level Implementation

6.3.1.1 Deformation Filters

The basic deformation implementation as defined by equation 10 in section 3.7.2.3 has been extended to support **deformation filters**. A deformation filter defines which vertices in a polyhedral mesh can be deformed by the deformation algorithm. The whole concept of a deformation filter, where a deformation is constrained to a specific part of an object's structure, contrasts with the term **Global Deformation**, where deformation parameters are applied to an entire object.

It is possible, using the radius of influence *R* in equation 10**,** to determine whether a vertex in the polyhedral mesh is to be deformed. This involves checking whether the vertex is within the allowed radius of influence. If it is, the deformation algorithm is applied to the vertex with the parameters that have been defined for that deformation; if not, the deformation of the vertex is avoided. The use of deformation filters allows this computation to be avoided at the expense of storing an array representing each deformation filter in memory. The benefit of the original algorithm, however, is that no prior knowledge of the object's topology is required.

The internal representation of a deformation filter can be described as a list of Boolean flags. This represents the deformation state of the vertex. If a vertex can be deformed, a value of TRUE is associated to it. Each deformation filter is constrained to have the same number of entries as there are vertices in the polyhedral mesh. This leads to a 1-1 mapping between each vertex and its deformation flag.

The primary advantage the deformation filter approach has over the original algorithm is that many different parts of the object's surface can be deformed simultaneously with a single deformation. This is performed by defining a deformation filter containing TRUE deformation

states for each vertex that comprises a part to be deformed. This is illustrated in the following example: given a bottle object and a deformation filter defining the bottom and neck of the bottle to be deformable, a deformation can be applied to deform both of these points to a given position.

6.3.1.2 Timing Considerations

Timing is one of the most important factors when performing deformations. The deformation implementation makes use of time as a weighting factor that allows gradual deformation of an object to be performed. It has been mentioned in section 6.2 that the VID time component can vary between time 0 and time 1. Time 0 implies that the application of the deformation algorithm to a sample vertex produces no effect. Similarly, a time value of 1 causes the full effect of the algorithm to be passed on to the vertex.

The generation of animations based on the deformation of an object over time involves the introduction of a time controlling parameter, `TimeDelta`. The process of animating a deforming object is as follows:

```
X ← StartTime
Y ← EndTime
WHILE X < Y DO
BEGIN
   Deform the object with time value X
   Update the rendering window to show the deformed object
   X ← X + TimeDelta
END DO
```

where: `StartTime` represents a value between 0 and 1 indicating what the magnitude of the time parameter in the VID equation will be at the beginning of the deformation, and the `EndTime` represents the value of the time parameter at the end of the deformation.

The above algorithm generates an animation of the deforming object. Controlling the `TimeDelta` parameter facilitates a method of controlling the realism of the animation. By choosing a very small value for the `TimeDelta`, a slow, yet extremely realistic deformation is achieved. Choosing larger values will speed up the animation. The `TimeDelta` parameter provides an additional level of control over the animation process; if, for example, an animation is too slow, it can be speeded up by increasing the value of `TimeDelta`.

6.3.1.3 Multiple Deformations

It has been mentioned above that expressions can typically be classified as a collection of atomic facial movements, each of which translates to an individual deformation. Combining a number of these deformations allows expressions to be simulated. To illustrate this concept, a smiling expression can be generated by deforming the left and right cheeks of an avatar in lock step.

The previous sections made no mention of support for multiple deformations. The ability to perform multiple deformations with an arbitrary number of deformation filters or even global deformations is very important. The reason for this is to support the notion of expressions being composed of a number of single deformations.

The basic algorithm has been extended to support the case of multiple deformations. This is illustrated below:

```
A ← Obtain a copy of the undeformed object
FOR I FROM 0 TO X DO
   FOR each vertex in A DO
      A ← Deform(A,D[I])
   END FOR
   D[I].CT ← D[I].CT + D[I].TimeDelta
END FOR
```

where:

- `D` represents an array of deformations that, taken as a whole, represents an expression;
- `Deform (A,D[I])` deforms `A` using deformation `D[I]` and returns the deformed vertex;
- `X` represents the number of individual deformations that must be applied to the object;
- `CT` represents the specific deformation parameter's current time (`t`);
- `TimeDelta` refers to the amount by which `D[I]`'s current time is updated.

Because each deformation filter has its own `TimeDelta`, this implies that if, for example, one deformation specifies an expansion (positive `TimeDelta`) and another deformation specifies a contraction (negative `TimeDelta`), these can also be performed in lockstep independently.

A copy is made of the polyhedral structure before the deformation is applied to it. All deformations are then applied to the copy (`A`). The result is that all deformations comprising a specific expression are applied successfully to `A` (the copy) for the current global time.

The above algorithm illustrates how more than one deformation can be applied to a polyhedral structure at the same time. This is a very desirable feature, since the final goal of the deformation

systems is the generation of possibly complex expressions. The addition of multiple deformation support to the system means that expressions can be created/modified with an expression editing system.

6.3.2 Expression Editing System

Appendix C describes the development of an expression editing system. Its function is to define the model-specific expression generation rules for each generic expression listed in the Expression Parameter Lookup Table. The implementation underlying the expression editing system is the three-tier expression hierarchy introduced in section 6.2, namely: Single Deformation Management, Multiple Deformation Management and Avatar Expression Management**.**

6.4 Expression Management

When connected to the encoder via the network, the decoder constantly receives a stream of data containing the pose recovery and the Expression ID sent by the encoder. The task of expression management refers to the process of determining how to handle the situation of receiving a different Expression ID to the one that is currently being generated. Two possible ways to handle the situation are:

- To morph from the current expression to the requested one. If a smile is being generated and a frown is requested, a linear interpolation method system can be employed to handle the subsequent animation. A weighting system is used to gradually transform one expression into the next, according to an equation such as:

```
FOR  X from 0 to 1 by Δ DO
BEGIN
     A ← (1-X) * B + X * C
END FOR
```

  where:

  - B represents the current expression,
  - C represents the new expression;
  - Δ represents the speed of transfer from the B to C; and

- X represents the weighing factor constraining the effect of each expression on the mesh;

- To place every Expression ID received onto the end of a queue of expressions to be generated. The generation system deforms the model according to the one specified at the front of the queue. Once this is done, the expression is removed from the queue, and the next one processed.

6.5 Results

Figure 6.2 illustrates the deformation of a plane using multiple deformations. Illustrations A, B and C show the plane deforming in the middle, while illustrations D, E and F depict the plane being deformed in every corner. Illustrations G, H, I, J and K show effects of combining both the first and the second deformation.

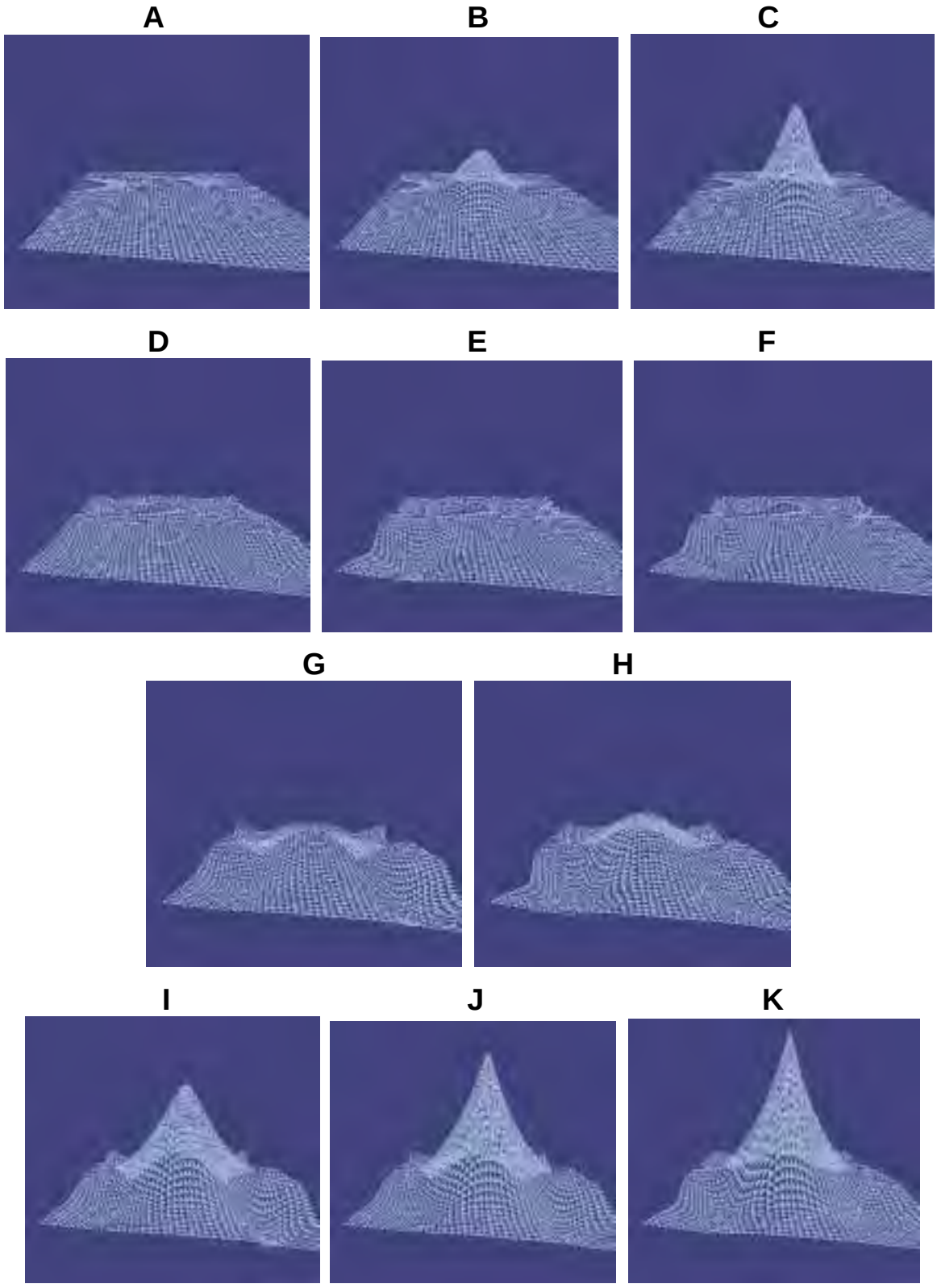


Figure 6.2 Deforming a Plane.

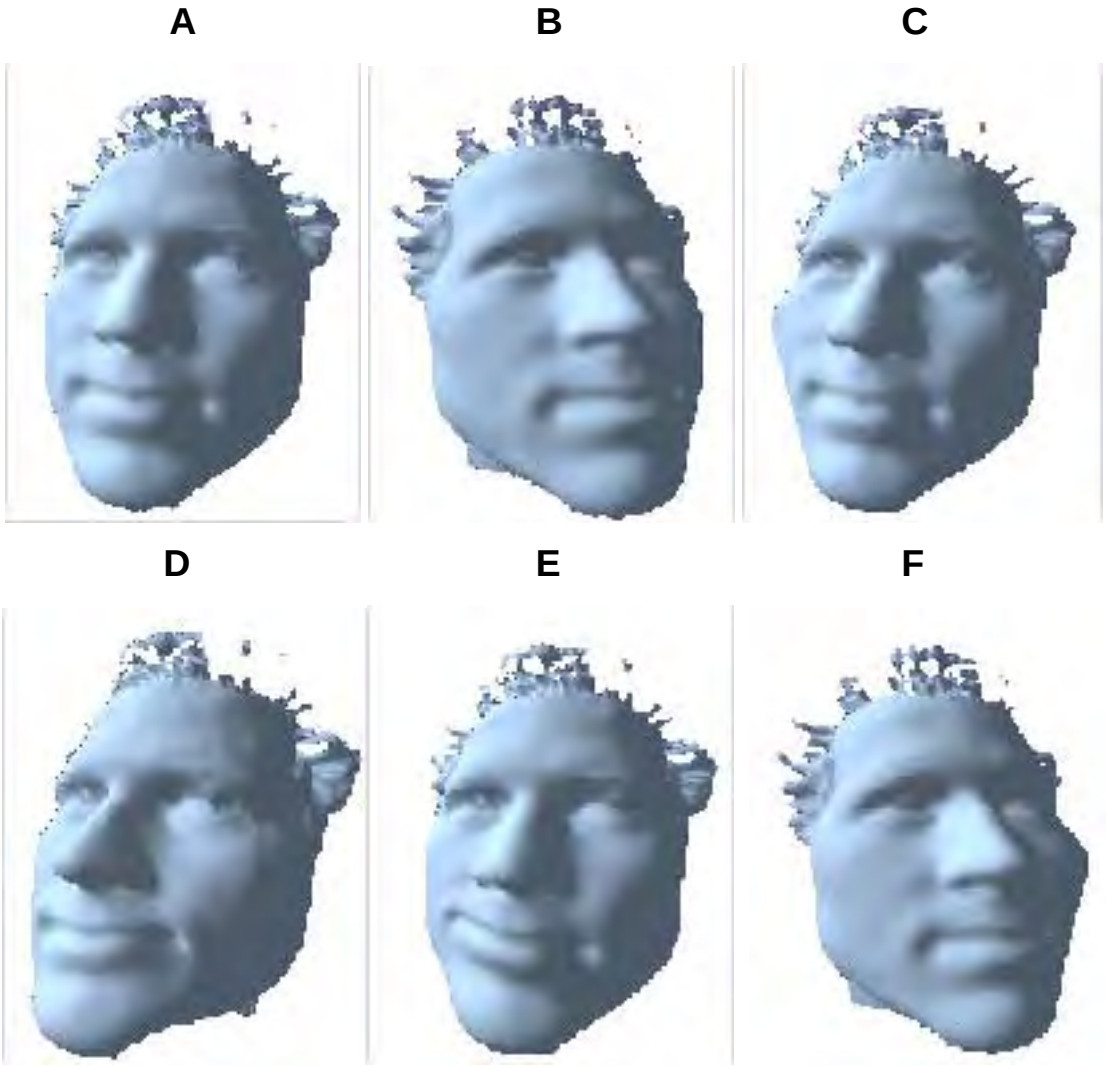


Figure 6.3 Generation of a Smile.

Figure 6.3 illustrates the step-wise generation of a smile. The generation of a smile with the avatar depicted is assumed to consist of four basic facial movements, namely:

- A movement of the left mouth edge upwards and outwards (illustration E)
- A movement of the left cheek outwards (illustration C);
- A movement of the right mouth edge upwards and outwards (illustration D);
- A movement of the right cheek outwards (illustration F);

Illustrations A and B depict the undeformed face from each side.

The table below illustrates the deformation parameters used for each of the above facial movements.

Smile VID Deformation Parameters			
Part of Face	**Source Force**	**Target Force**	**Radius**
Left Mouth Edge	-0.5,0.3,2.0	-1.50,0.10,0.50	1.20
Right Mouth Edge	0.5,0.3,2.0	1.50,0.10,0.50	1.20
Left Cheek	-1.30,0.10,1.00	-1.70,1.50,0.80	1.05
Right Cheek	1.30,0.10,1.00	1.70,1.50,0.80	1.05

All four deformations share the same `EndTime` and `TimeDelta` parameters, namely 0.2 and 0.02. The final smiling avatar is illustrated in **Figure 6.4**. The generation of a smile and a frown applied to an avatar using the VID deformation system is illustrated. Illustration A indicates a neutral expression and illustration B shows the smile. Finally, illustration C depicts the avatar frowning.

6.5.1 Rendering Performance

The head model contains 7013 vertices and 13338 polygons, all triangulated, while the plane consists of 2500 vertices. The impact of the deformation system on the rendering engine used is illustrated in the performance table below.

VID Deformation Performance			
Model	**Vertices**	**No Deformation (FPS)**	**Full Deformation (FPS)**
Plane	2500	65	61
Avatar	7500	22.9	16.5

The results are achieved using a dual 195 MHz MIPS R10000 processor subsystem. To remove the effects of texturing on the performance measurements, neither of the models was textured. As can be seen from the above table, the performance penalty incurred by deformation with the vertex interpolation system is negligible. A rendering performance penalty of 27.9 % is measured during the deformation of the avatar object. This contrast in performance over the plane's penalty of 0.8 percent is attributed to the following factors, namely:

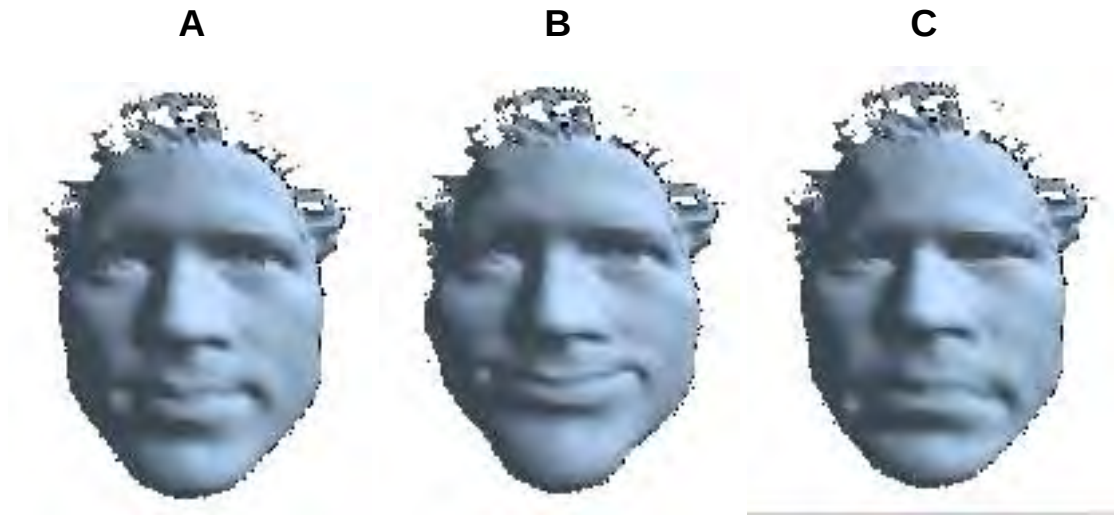


Figure 6.4 A Deforming Head.

- The plane is three times smaller than the avatar in terms of number of vertices;
- The generation of the avatar smile consists of four deformations, while only a single deformation was applied to the plane.

6.5.2 Deformation Issues

The vertex interpolation mechanism does not attempt to guarantee C^1 and C^2 surface continuity or lack of self-intersection during deformation. One aspect that is highlighted by **Figure 6.3** (specifically) and **Figure 6.4** (in general) is the degree to which expressions can be specified with facial movements requiring minimal structural change. The evidence seems to suggest that the VID approach is sufficient for the purposes of expression generation, since the possibility of self-intersection due to large facial movements does not seem likely to occur during realistic facial expression modelling.

6.6 Improvements

6.6.1 Fiducial Deformation Control

The biggest problem with the current design and implementation is that the resulting expressions are very rigid. An alternative approach could be to bypass the use of an Expression Parameter

Lookup Table altogether. A mapping of the fiducial points to the polyhedral mesh structure could be employed. The idea is that, given this kind of mapping, the tracked fiducials can be treated as deformation forces applied to the avatar representation of the user. As the subject smiles, for example, the fiducial points surrounding his mouth widen and move upwards. Provided a mapping exists for these points from the subjects face to the polyhedral mesh, the position of the fiducials could be used to control the VID system's Target Force parameter. This would generate far more realistic and accurate expressions and would remove the need to perform expression management. An avatar that physically represents the subject would be required for such an improvement. The reconstruction system detailed in Chapter 4 satisfies this requirement.

6.6.2 Performance Enhancements

Because the deformation system is vital to generating expressions, its performance must be as fast as possible. The current implementation remains unoptimised. A possible optimisation that should be considered is the use of a cosine lookup table to minimise the time taken by the cosine function in the VID equation (see section 3.7.2.2).

6.6.3 Expression Management

A weighting system has been defined that specifies how the problem of receiving requests to generate multiple expressions at the same time. The implementation of this feature is desired.

6.7 Summary

This chapter details the structure underlying the decoding part of the component-based system design, and the development thereof.

Avatar movement, as identified by the encoder, is assumed the responsibility of the underlying system used. It is further assumed that the position and orientation of the object can be updated in real time and on-the-fly.

The development of a vertex interpolation deformation system for the purposes of expression generation is discussed in section 6.2. An expression is classified as a collection of deformations that are collectively applied to the avatar representing the subject. The development of an expression editing system is discussed in section 6.3.2. Its purpose is to build up a list of EPLT-

compliant expressions. This imposes the constrain that every avatar used has a collection of deformation parameters defined for every expression in the EPLT.

Expression editing is solved using a three-tier solution, namely:

- Single Deformation Management for specifying a single deformation;
- Multiple Deformation Management to combine one or more single deformations, and thereby represent an expression. The video-editing paradigm is used to manage the list of deformations;
- Avatar Expression Management provides a high-level mechanism to test all expressions defined for an avatar; a selection can be made from a list of expressions, and the selected expression generated. This firstly allows for improvements to specific expressions to be identified. Secondly, the underlying implementation simplifies the implementation of the decoder component. Given that the encoder identifies a smile, the EPLT equivalent Expression ID is selected and passed to the decoder. The deformation parameters related to the received Expression ID subsequently replace the current active parameters. The result is a simplified control mechanism for altering the expression of an avatar.

The task of expression management is defined in section according to a weighting system, allowing for gradual morphing from one expression to another.

It is shown in section 6.5 that the vertex interpolation deformation system is able to facilitate real-time expression generation, and, given its constraints, namely that it does not observe surface continuity and self-intersection issues, is well-suited to the purpose of realistic facial expression generation. This is due to the observation that expressions consist of a collection of deformations that do not overly deform the avatar structure.

6.7.1 Contributions

This chapter makes the following contributions:

- An expression has been defined as a collection of individual facial movements;
- A vertex-interpolation deformation system has successfully been used for the purposes of expression generation. The advantages of the this approach are:
    - It has complexity of magnitude *O(n)*;
    - It facilitates real time deformations;
- The basic VID specification has been enhanced through the introduction of deformation filters. Each filter translates to an array of deformation states for each vertex in the

avatar’s polyhedral structure, defining whether or not a specific vertex is subject to deformation;

- An algorithm to combine more than one deformation for the purposes of expression generation has been defined and implemented.

The proposed use of the VID system has been motivated by the observation that, taken in isolation, the deformations that underlie a specific expression result in structural change that is small with respect to the size of the avatar used. This is valid primarily when realistic facial expressions are modelled. As such, surface continuity and self-intersection issues can be avoided altogether.

The final contribution of this chapter is an implementation of a three-tier expression editing system.

Chapter 7 - Virtual Conferencing under CoRgi

7.1 Introduction

The technologies underlying each of the components defined as part of the system design in Chapter 2 have been identified and discussed. This chapter provides an overview of the implementation of the system as a whole on top of the CoRgi architecture.

CoRgi meets all the low-level functional requirements identified in section 2.5, namely:

- A connection-based component model;
- A platform-independent networking model;
- An OpenGL-based renderer to handle the rendering of avatars; and
- Low-level video-capture capabilities.

As such, CoRgi satisfies most of the functional requirement of the envisaged system. It handles low-level implementation details, while allowing for the rapid development of applications with full multimedia capabilities. Additionally, it introduces true VR capabilities to this class of application, an aspect addressed by very few authors. Most of the systems developed to date have been discussed by their respective authors as autonomous implementations. CoRgi not only facilitates a VR paradigm as part of the development process, but its component model means that any components developed to deal with specific problems with respect to the system design become available to other applications.

As is mentioned in section 2.5, the implementation of the design involves the creation of two applications, the one serving as the encoder, and the other being the decoder. One possible solution is to develop an application in its entirety to do the encoding of the video stream into avatar control data, transmission of this data across a network, expression generation with 3D avatar representations, and finally rendering of the results to the screen. In addition to adding unnecessary complexity to the problem at hand, the different technologies are absorbed into a system that is aimed at solving a specific problem. The use of CoRgi avoids this problem. To summarise, the advantages the use of CoRgi affords are as follows:

- There is a tight coupling between the component structure of the coding system and CoRgi's component-based nature;
- Technology developed for the coding system is absorbed into the existing VR system and does not form part of any autonomous system. The VR system is enriched and enhanced

in this way, leading to functionality not previously present. Examples include deformation support and image-based tracking;

- As mentioned above, it facilitates high-level development of the coding system.

A more complete description of CoRgi is provided in section 7.2. This is followed by a description of the structure of a typical CoRgi application in section 7.3, to provide a practical context on the use of the system. The remainder of the discussion in this chapter concentrates on the component-based implementation of each part of the system design. Each of the sections relating to the implementation of the system design below describes, firstly, how a certain aspect of the CoRgi system works. The focus is then shifted to show how the requirements of a virtual conferencing system have been moulded to suit CoRgi. Issues relating to dissemination of the technology developed for this application, and system performance, conclude this discussion.

7.2 CoRgi

CoRgi can be described as an object-oriented, component-based, distributed VR system. A connection-based paradigm is employed to facilitate quick prototyping of applications requiring multimedia capabilities generally, and VR capabilities specifically. While the major focus during the development of this system has been on the VR side, the ability to seamlessly integrate audio, video and VR provides enhanced versatility.

The system is composed of a number of components that, when combined together, form an application. They can be broadly sub-divided into three categories, namely:

- System or base components: these are the most basic components that comprise the structure of the system. They are mostly base classes that specify interfaces that must be adhered to. In doing this, they define the structure of the remaining descendent components. The most fundamental components here deal with basic network transmission and also provide a basic framework for the connection of components to one another;
- VR components: these components encapsulate the core VR functionality of CoRgi; included here is a rendering engine and VR device components;
- Multimedia components: these components are responsible for the generation and management of multimedia data, namely video and audio. For example, a component that deals with the low-level capturing of video from a CCD camera can be linked to a component that handles the display of a video feed to a screen.

Once the functionality required from an application has been identified, the components that offer facilities satisfying the requirements can be chained together to create the final system.

7.3 Skeleton CoRgi Application

Every component in the CoRgi system is subclassed from a root class called **`Component`**. From this root class, the `VRComponent` and the `VideoComponent` classes are derived. These classes form the basis of all components in the VR and the multimedia parts of the system.

The CoRgi system has been designed in such a way that any class derived from the `Component` class will have its `ThreadRoutine()` method executed repeatedly during the execution of the main application using an instance of this class. Declaring instances of any `Component`-derived sub-class during the execution of an application forces the registration of that class with the base `Component` class. The base `Component` class defines a linked list of pointers to the derived `Component` instances. When a `Component`-derived class is instantiated, the constructor forces its addition to the linked list. The basic `Component` class also has a `friend` method called `RunComponents()` defined which, when called, executes the `ThreadRoutine()` method for each of the registered component instances in the order they are instantiated in the application i.e. the linked list is traversed, and each entry's `ThreadRoutine()` method is executed. Every component that is derived from the `Component` base class is assumed to override and implement its own `ThreadRoutine()` method. The base `Component::ThreadRoutine()` method forces an application exit, so it is imperative that this method be overridden in a derived class to ensure useful system behaviour. The reason for this design decision becomes apparent when one looks at the structure of a typical CoRgi application. A CoRgi application can be described by the following steps:

- The components that form the component chain must be defined. For a videoconferencing-type application, a `VideoSource` component and a `VideoSink` component are required;
- Devices for sources or sinks used by the components must be instantiated. A `VideoDeviceCamera` device allows for the capture of video from a CCD camera, while `VideoDeviceOutput` device displays the captured video to the screen in a window. Therefore:

```
VideoDevice * in;
In = new VideoDeviceCamera;
VideoDeviceOutput * viewwin;
```

```
viewwin = new VideoDeviceOutput;
```

- The components are then initialised with their respective devices, namely:

```
VideoSource vidsource (in);
VideoSink vidsink (viewwin);
```

- The application's component connection chain must be specified. This definition specifies the order in which components are connected to one another. A `Connection` class is responsible for specifying the connections between each of the components. An example of this is:

```
connection SourcetoSink(vidsource,vidsink);
```

  It is important to note that the order in which connections are made is critical. If a specific component is dependent on a previous component's `ThreadRoutine()` execution, that connection must be specified earlier, so it appears earlier in the component linked list to improve application efficiency;
- The `RunComponents()` method is executed. This ensures that the `ThreadRoutine()` method of each component that is part of the connection chain is executed. The order in which the connections are specified dictates the order in which the relevant `ThreadRoutine()` methods are executed. Therefore, `vidsource.ThreadRoutine()` is executed before `vidsink.ThreadRoutine()`.

A CoRgi application thus depends very heavily on each component's `ThreadRoutine()` method. Combining the `ThreadRoutine()` methods of all the components describes the core functionality of the typical CoRgi application. Having a component without a derived `ThreadRoutine()` method is thus meaningless in terms of application execution.

7.4 System Implementation with CoRgi

The design introduced in section 2.4 is broken down into an encoder involved in a one-way communication with a decoder via a network. At initialisation, a connection is established between the encoder and decoder. The encoder is responsible for a number of functions, namely:

- Model acquisition for the reconstruction of an avatar representing the subject;
- Capturing of video from a CCD camera;
- Image-based fiducial tracking; and

- As a direct result of the fiducial tracking, pose estimation and expression analysis.

Other than the model acquisition task, all functions listed above form part of the encoder's implementation under CoRgi. The model acquisition component has not been implemented as part of CoRgi, but rather as a standalone application. It does ensure model compatibility with CoRgi by generating a 3D geometry and vertex colour map representation of the reconstruction that can be used by CoRgi.

Once the relevant data (pose estimation and Expression ID) has been extracted, it is packaged up into a network packet and sent across the network to the decoder at the receiving end. The decoder's responsibilities then include moving the avatar according to the position information received and deform the model according to the expression identifier received.

In order for the encoder and decoder to understand each other, the structure of the network packet has to be standardised. For the data to be correctly understood by the decoder, a file naming convention has been adopted. This means that, given the name of an avatar (`avatarID`) that is to be used in a conferencing session, it can be deduced where the appropriate information for that specific avatar is to be found. Thus, given an `avatarID`, the file naming convention is as follows:

- `<avatarID>.OFF/<AvatarName>.COL`: the 3D geometry and associated vertex colour maps of the Avatar;
- `<avatarID>.setup`: the pixel coordinates of the fiducial points and the associated frame resolution of the measurements;
- `<avatarID>-explist`: the expression list for this specific avatar. This file lists the expression files containing deformation parameters for each expression in the EPLT;
- `<avatarID>-<expression>.exp`: where `<expression>` denotes the name of an expression, such as smile or frown. Each of these files contains deformation parameters required to generate the relevant expression.

On start-up, the decoder waits to receive the first network packet from the encoder. Once this occurs, the decoder uses `avatarID` to initialise the decoder. This initialisation involves setting up the `VisualRepresentation` for the avatar as well as the expression generation subsystem.

7.4.1 Networking Support

The desired result of the networking implementation is that the encoder will be able to communicate avatar control data to the decoder. It is also assumed that the connection will be a one-way connection, and thus the UDP network protocol has been used.

The data that must be packaged up and transmitted every time a packet is sent is:

- `expressionID`: the expression that has been identified. This value is determined from the mapping performed using the Expression Parameter Lookup Table. The value is an integer, but in the interests of lower bandwidth requirements, a byte can be used;
- `Position`: the position at which the head is located. this value is of type `Point3D`, essentially a 3D coordinate;
- `Orientation`: the orientation of the head. The pose is represented compactly with a single quaternion;
- `PuppetID/avatarID`: A string identifying the name of the puppet to which the control data is to be applied.

7.4.1.1 CoRgi's Networking Approach

The networking protocol with CoRgi assumes that each item to be sent across the network has a header and associated body of information. The header is called a `VRDeviceInputHeader`. It specifies the type and amount of data that is being transmitted, and allows the data to be classified when it is received across the network. Assuming a device was used to generate the data being transmitted, the header allows one to classify the data. In the case of the encoder/decoder system, a new type has been derived from the `VRDeviceInputHeader` type and is called `VRDevicePuppetControl`.

At a low level, the body of information associated with the item of data is a contiguous block of data. Each `VRDeviceInputHeader`-derived type has an associated mapping mechanism that defines what data is present in the block and where it can be accessed. This mapping mechanism is type specific. For the `VRDevicePuppetControl` type, the associated structural decomposition is represented in a class called `VRDevicePuppetControlData`. Its definition is illustrated in **Figure 7.1**.

A mechanism within the encoder is responsible for the generation of data with the above structure. This data generator is then connected to a `VRNetworkInput` object.

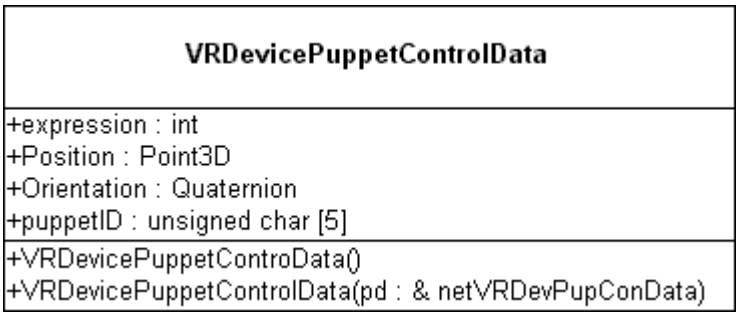


Figure 7.1 VRDevicePuppetControlData Class UML Specification.

`VRNetworkInput` handles the transfer of the `VRDeviceInputHeader` and associated data across the physical network. The transmission is based on the Unix-like socket mechanism. When the `VRNetworkInput` object is initialised, it is provided with a socket connection number on which to transmit the data received from the data generator.

At the decoding end, a `VRNetworkInput` object is initialised to listen for data on the same port as the encoder is transmitting. Any data that is broadcast from the encoder to the decoder is thus collected. Additionally, a `VRInputClient` object is instantiated and connected to the decoder's `VRNetworkInput`. It is responsible for handling the extraction of the both the data and header from the `VRNetworkInput` object.

7.4.1.2 Platform-Independent Networking

While the above scenario illustrates the simplicity of CoRgi's networking approach, the underlying networking implementation involves some added complexity. Different computer architectures employ different mechanisms for data representation, namely little-endian and big-endian representations. Fortunately, CoRgi's networking structure offers a type-safe environment for this situation by defining a **net** equivalent data type for each of the standard data types. For example, if an integer were to be transmitted across the network, the actual data transmitted would be a `netint`, instead of an `int`. This process involves marshalling and unmarshalling of the data. The result of this is that for each network data type that is defined, a platform-independent `net` equivalent must be defined. To this end, a `VRDevicePuppetControlData` type has been defined for the `VRDevicePuppetControl` header type. The platform-independent `net` version is called `netVRDevicePuppetControlData`.

7.4.2 Encoder

It was suggested in the previous section that puppet control data is generated in some way, packaged up and then transmitted across the network. This section discusses the development of a CoRgi-specific device that is able to generate the appropriate control data.

7.4.2.1 Encoder Design

Both the pose estimation and expression analysis components are implemented as part of a single class hierarchy called `VideoAvatarTracker`. This class takes as input a video stream from a `VideoSource` component, and generates as output the head tracking information and the expression that has been analysed. Additionally, the video stream with the fiducial points overlaid on each frame is displayed to the encoder's screen. The reason for this is to provide feedback to the user, and therefore allow the user-assisted tracking process to take place. The encoder's component chain is illustrated in **Figure 7.2**. The bevelled square attached to the chain

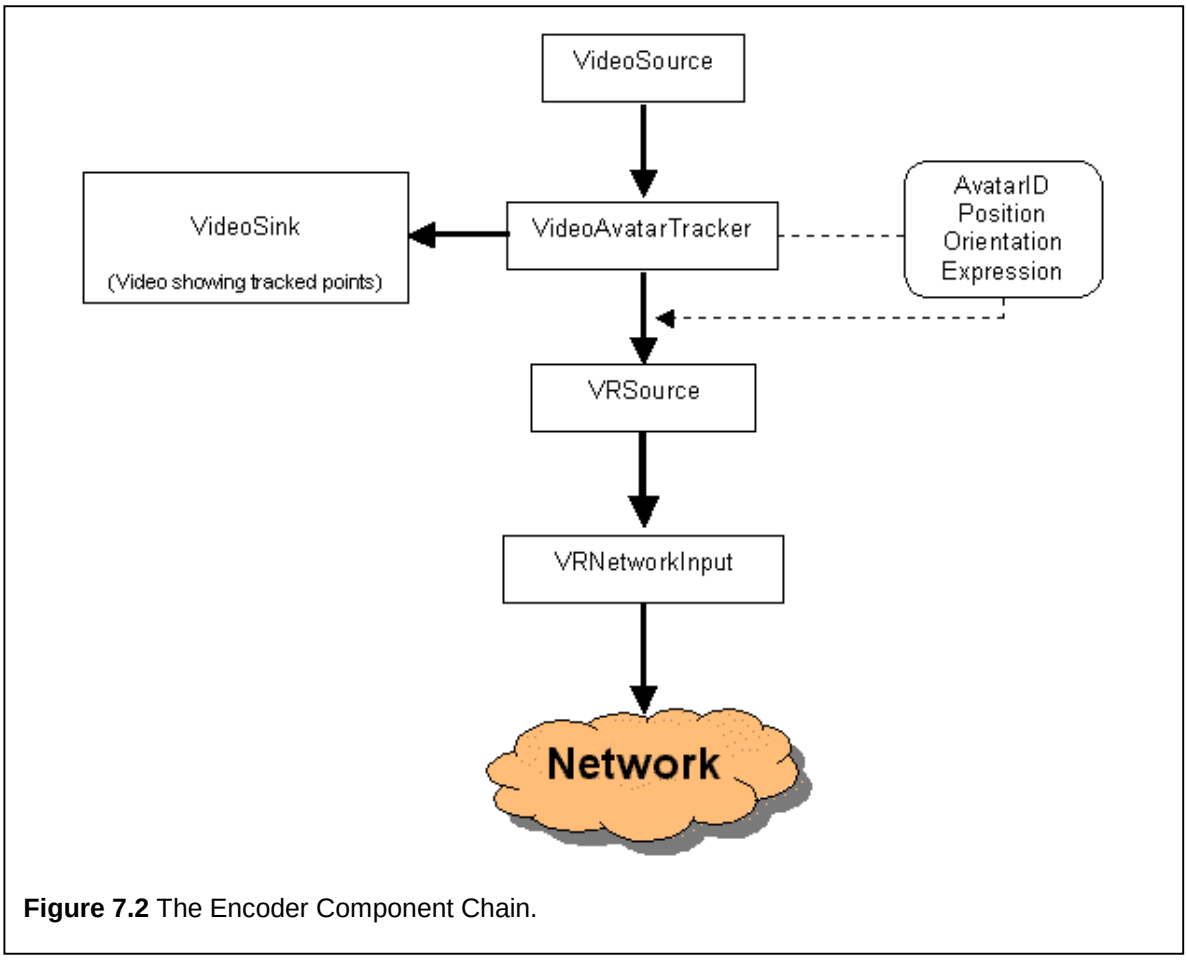


Figure 7.2 The Encoder Component Chain.

via a dashed line is representative of the data that is generated by the `VideoAvatarTracker` component. The `VideoSink` is responsible for keeping the user informed of the current state of the fiducial point tracking, and is displayed to the local machine on which the encoder application is executing.

The implementation discussed here marries together two distinctly different parts of CoRgi, namely Multimedia (specifically the video components) and VR. The class `VideoAvatarTracker` is defined as having a multiple inheritance hierarchy, inheriting from both `VideoTracker`, and `VRPuppetControlInputDevice`. The hierarchy is illustrated in **Figure 7.3**

The `VRPuppetControlInputDevice` component can be classified as a VR device that is responsible for the generation of avatar control data, namely the pose and expression data. The `VideoTracker` component implements the NCC tracking algorithm to perform image-based tracking.

The `VideoAvatarTracker` component inherits `VideoTracker`'s ability to perform NCC

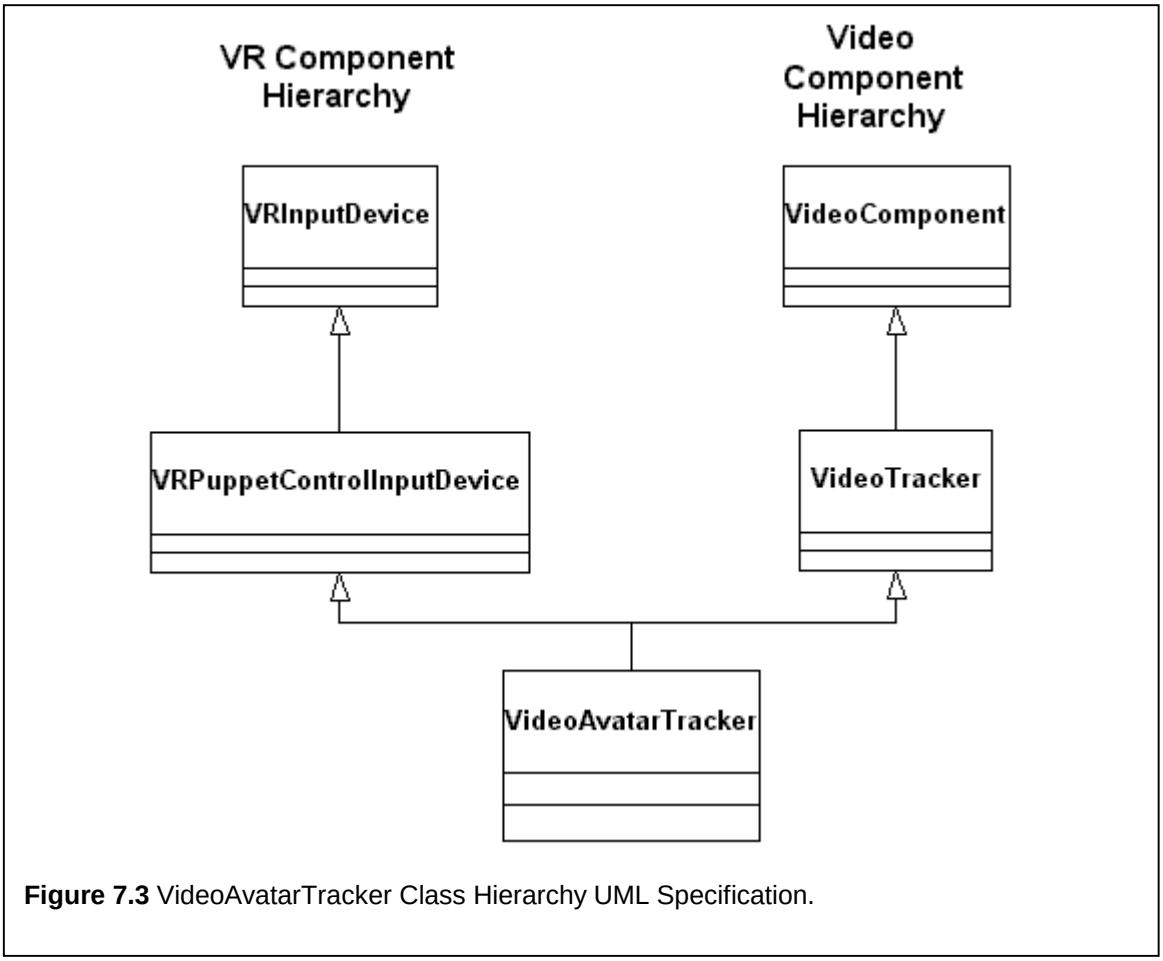


Figure 7.3 VideoAvatarTracker Class Hierarchy UML Specification.

tracking, and intelligently evaluates the results of the tracking. It also inherits `VRPuppetControlInputDevice`'s avatar control data generation capability. The result is a component that takes as input a live video stream and generates as output a continual stream of `VRDevicePuppetControlData` network packets. Additionally, the live video captured from the CCD camera is altered by overlaying the fiducial points' current positions on top of each video frame. This allows the user to observe the tracking process, and thus facilitates the user-assisted tracking mechanism discussed previously, since it allows the user to identify when the tracking system fails.

The last component that forms part of the encoder component chain is the `VRNetworkInput`. This is a sink component that packages up the data read by the `VRSource` and transmits it across the network to the decoder listening on a predefined port.

7.4.2.2 CoRgi Devices

As mentioned in section 7.3, any application developed with CoRgi implies the connection of components together to represent the final system. The logical partitioning has so far been along the lines of VR or multimedia components. Another way of looking at the CoRgi design philosophy is from a data flow perspective. There are components that can be classified as **sources** whose sole purpose it is interface with **devices**. They obtain their data from devices connected to them. These sources are in turn connected to **processing** components. The processing components perform operations on the data, the results of which are passed on to the **sink** components.

All devices have two things in common, namely:

- They are all derived from the `VRInputDevice` component; and
- They all override `VRInputDevice::GetData()`; every processing component to which a device is connected expects to find a valid `GetData()` method as part of that device, and as such will attempt to call it.

The linkage between a device and source component is typically initiated through the source component's constructor. Given an instance of a `VRInputDevice` called `vidtracker`, for example, the device can be linked to a `VRSource` as follows:

```
VRInputDevice vidtracker;
VRSource vrsourcepuppet(&vidtracker);
```

The `GetData()` method's interface is defined as follows:

```
void VRInputDevice::GetData(VRDeviceInputHeader &Data,
                            netbyte * & data, int Device);
```

This means that any `VRInputDevice` subclass overriding the `GetData` method must specify what the type of data is to be generated by this device.

7.4.2.3 Encoder Data Generator (`VRPuppetControlInputDevice`)

Given the definition of a `VRDevicePuppetControl` network packet type in section 7.4.1.1, the associated device that is responsible for generating datagrams of this type is called `VRPuppetControlInputDevice`. This new device component is subclassed directly from `VRInputDevice` and as such, the base class's `GetData()` method is overridden to cater for the generation of puppet control data (`VRDevicePuppetControlData`). This method has the following implementation structure:

```
Void VRPuppetControlInputDevice::GetData(VRDeviceInputHeader
                        &Data, netbyte * &data, int Device)
{
    if ((Device >= 0) && (Device < numdevices))
    {
       Data.type = VRDevicePuppetControl;
       Data.length = sizeof(VRDevicePuppetControlData);
       data = (netbyte *) malloc (sizeof
                        (VRDevicePuppetControlData));
       VRDevicePuppetControlData * newdata;
       newdata = (VRDevicePuppetControlData *) data;
       newdata->expression = incoord.expression;
       for (int i = 0; i < NameLen; i++)
          newdata->puppetID[i] = incoord.puppetID[i];
       newdata->Position = incoord.Position;
       newdata->Orientation = incoord.Orientation;
   }
}
```

where:

- `Incoord` acts as a global storage structure containing the most recent pose and orientation information, and

- `Namelen` represents the maximum allowable length of the name that will be transmitted across the network.

All methods in the encoder responsible for pose estimation and expression analysis update the `Incoord` global variable every time they are executed. The construction of the data packet is performed from data stored in `Incoord`.

For the puppet device `VRPuppetControlInputDevice`, the type of data that is generated is `VRDevicePuppetControl`, and the length is determined through the standard C/C++ `sizeof()` function.

7.4.2.4 NCC Tracker Implementation (`VideoTracker`)

The `VideoTracker` component performs the NCC tracking of a number of fiducial points. This component is a dumb tracker; its basic purpose is to track any number of fiducials. There is no limit on the number of points that can be tracked. The implementation of the NCC-based tracking algorithm resides in `VideoTracker`. This class is derived from the `VideoComponent` class, and as such overrides the base class's `ThreadRoutine()` method to perform the tracking.

During the discussion on the design of the encoder in section 5.2, it is mentioned that the user is responsible for specifying the points of interest on the subject's face. Given this constraint, the head tracker can be classified as being in one of two modes, namely **Acquisition** or **Tracking Proper**, a classification introduced by Carceroni *et al.* [10] (see section 3.5). The normal mode of operation is tracking proper mode, while the acquisition mode is required to set the acquisition data. The currently active mode is determined by a flagging system. The state-based system is based on the following algorithm that is implemented in `VideoTracker::ThreadRoutine()`:

```
IF data is being received  THEN
BEGIN
    CURRFRAME ← INCOMING VIDEO FRAME
    IF data can be transmitted THEN
    BEGIN
          IF Current Acquisition Data is Invalid THEN
              Acquisition Data ← CURRFRAME
          END IF
          IF template is present THEN
             IF Acquisition Mode is active THEN
                Update the Acquisition data
                Disable Acquisition Mode
```

```
            END IF
            IF tracking proper mode is active THEN
               Perform Tracking with the CURRFRAME
            END IF
         IF fiducial points must be shown on   CURRFRAME
         THEN
              Show the new positions of each fiducial on
              CURRFRAME
         END IF
      END IF
   END IF
   OUTGOING VIDEO FRAME ← CURRFRAME
END IF
```

Initially, the system is in Acquisition mode. Once the coordinates of each fiducial point in the current frame have been determined, the `VideoTracker::TrackPoint()` method allows the fiducial points and acquisition image to be set independently of each other. Additionally, the `VideoTracker::EnableShowPoints()` specifies whether the fiducial points are highlighted in the video sequence. This method provides the feedback required for the user-assister correction process.

Once the acquisition data has been registered with the system, Tracking Proper mode can be initiated by executing `VideoTracker::EnableTracking()`. The process of performing the actual tracking occurs by calling `VideoTracker::PerformTracking()` method. The details for the current frame (`CURRFRAME`) are passed to this method. At a high level, the `PerformTracking()` method deals with the tracking process.

It has also been mentioned in section 5.3.4 that a caching mechanism is employed to facilitate the user-assisted correction loop. A fixed-sized array is employed to keep a tracking history. The `VideoTracker::PerformTracking()` method manages this task as well.

7.4.2.5 Derived Implementation (`VideoAvatarTraker`)

`VideoAvatarTracker` is subclassed from both `VRPuppetControlInputDevice` and `VideoTracker`. It performs a dual role of acting as a device and a `VideoComponent`. It enhances `VideoTracker`'s functionality by adding a number of evaluation steps to each frame that is processed. Conceptually, it also introduces the notion of a Fiducial Graph. Given that the `VideoTracker` component is responsible for tracking a number of points in the incoming video feed, it takes each of the points and groups them into a Fiducial Graph. This conversion process is based on the assumption that the number of points being tracked is equivalent to the

number of points in the fiducial graph. The conversion process is done with the `VideoAvatarTracker::UpdateFiducials()` method.

Two fiducial graphs are present and active at any time, namely the acquisition data's fiducial graph and the current fiducial graph. To facilitate the conversion process, the updated `VideoAvatarTracker::ThreadRoutine()` method has the following structure:

```
VideoTracker::ThreadRoutine()
IF Tracking Proper Mode is Active THEN
BEGIN
     Update the Acquisition Data's fiducial graph
     Update the current fiducial graph
     Extract Pose
     Extract Expression
END IF
```

The fiducial graphs are updated every time the `VideoAvatarTracker::ThreadRoutine()` is executed.

The subclassing from `VRPuppetControlInputDevice` implies that process of puppet control information generation must be handled. It has been mentioned in section 7.4.2.3 that datagram construction occurs from an `Incoord` data element, a process that becomes evident if the structure of `VRPuppetControlInputDevice::GetData()` is evaluated. The process of data generation thus implies updating the `Incoord` variable to accurately reflect the avatar control information. This is a two-fold process and is based on evaluating the changes between the two fiducial graphs. Once the respective fiducial graphs have been updated, the `ExtractPose()` method is called to determine the user pose. The `ExtractExpression()` method then performs expression analysis and classification according to the Expression Parameter Lookup Table. Both of these methods update the `Incoord` variable introduced in section 7.4.2.3. When `VRPuppetControlInputDevice::GetData()` is called, the appropriate data is generated and the `incoord` variable updated to reflect these updates. The `VideoAvatarTracker::ExtractPose()` and `VideoAvatarTracker::ExtractExpression()` methods ensure that these updates occur.

7.4.3 Decoder

The decoder is responsible for extracting the appropriate avatar control data from the incoming network packets and applying deformations to the subject avatar. Its component chain consists of the following parts, namely:

- `VRNetworkInput`: responsible for decoding the incoming network packets;
- `AnimatedDeformableVisualRepresentation`: once the packets have been decoded, the avatar can be deformed to the expression specified in the network packet;
- `VRSink`: this is the OpenGL-based renderer used to display the animating avatar.

The decoding chain is illustrated in **Figure 7.4**.

Initially the decoder is not aware of which avatar to use for the conferencing session. This is where the file naming convention introduced in section 7.4 becomes important. Once a connection has been established between the encoder and the decoder, and the first network packer has been received, the decoder uses the `puppetID` identified to determine which avatar geometry to load from disk. Additionally, the deformation parameters for the puppet are loaded from disk as well.

This section concentrates on the implementation of the vertex-interpolation based deformation algorithm for the purposes of expression generation. During the implementation of this algorithm, a number of shortcomings with the CoRgi system were identified. These

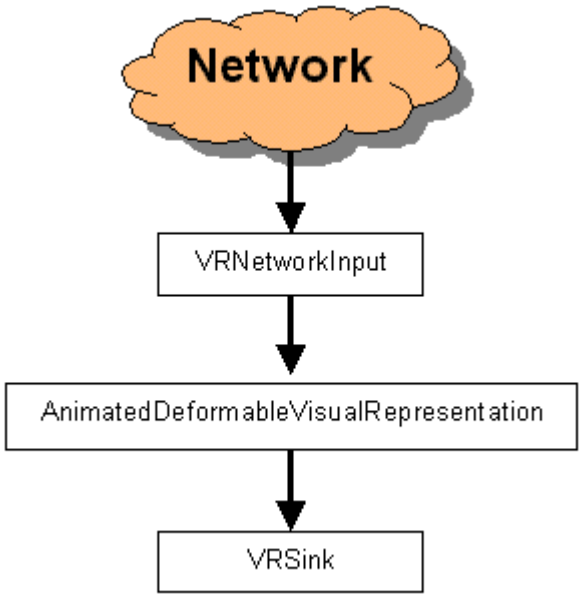


Figure 7.4 The Decoder Component Chain.

shortcomings, and the way each has been addressed, are discussed below. The result is the development of an `AnimatedDeformableVisualRepresentation` class that handles all aspects of expression generation.

7.4.3.1 Internal Data Access

The first problem that was encountered was the lack of access to the 3D object data within a rendered scene. Each object in a CoRgi virtual scene is internally represented by an instance of the `VisualRepresentation` instance. A `VisualRepresentation` is constructed from a `ShapeDescription` object. The `ShapeDescription` object class is a base class that is intended for the transparent conversion of a specific 3D data file formats to a standard internal polyhedral representation. There are a number of derived classes that perform such conversions. These include the `OFFShapeDescription` and the `ASEShapeDescription` classes. The `OFFShapeDescription` class takes as input an OFF file and stores it internally in the standard format. Unfortunately, however, no public methods exist the `ShapeDescription` family class to facilitate the internal manipulation of vertex/face data. The problem is complicated by the fact that the polyhedral data structures are defined as **protected**, thus preventing direct array access from externally objects. Additional public methods became necessary.

The solution to the problem entails the subclassing of `OFFShapeDescription` to `OFFManipulateShape`. Additionally, though, two new public methods were added to the class interface. These are:

```
virtual void OFFManipulateShape::SetInternalVert(int index,
Point3D P);
virtual Point3D OFFManipulateShape::GetInternalVert (int
index);
```

`SetInternalVert()` allows the vertex denoted by the `index` parameter in the internal data structure to be set to the 3D vertex `P`. In exactly the same fashion, the vertex information at position `index` in the internal data structure can be obtained with the `GetInternalVert()` function method. The advantage of these methods is that they provide the facility to write higher-level object manipulation routines without cluttering the `OFFManipulateShape` method list.

The disadvantage of this approach is one of speed. In an ideal situation, the number of procedure/API calls must be minimised during the execution of any code segment. Repeatedly calling either of these two methods would result in overall performance degradation.

7.4.3.2 Structural Deformation Implementation

The one major requirement of the deformation system is that it be as efficient as possible. To maximise speed and minimise procedural overhead, a method has been added to the `OFFManipulateShape` class that provided a basic deformation interface. Its class interface is illustrated in **Figure 7.5**. The parameter `defparams : Deformation_Parameters` has been included for brevity. It expands as follows:

```
VertListFilter[] : int
bbox :int
Po : Point3D
Pd : Point3D
R : float
t : float
```

All future references to `defparams` or `Deformation_Parameters` refer to this mapping.

The deform method takes as input via parameters a copy of the current object's shape (`shape`), and a number of other parameters. The following functionality is desired from the deformation algorithm:

```
Copy ← internal polyhedral representation of the object
FOR each vertex V in Copy DO
    V ← Deform (V)
```

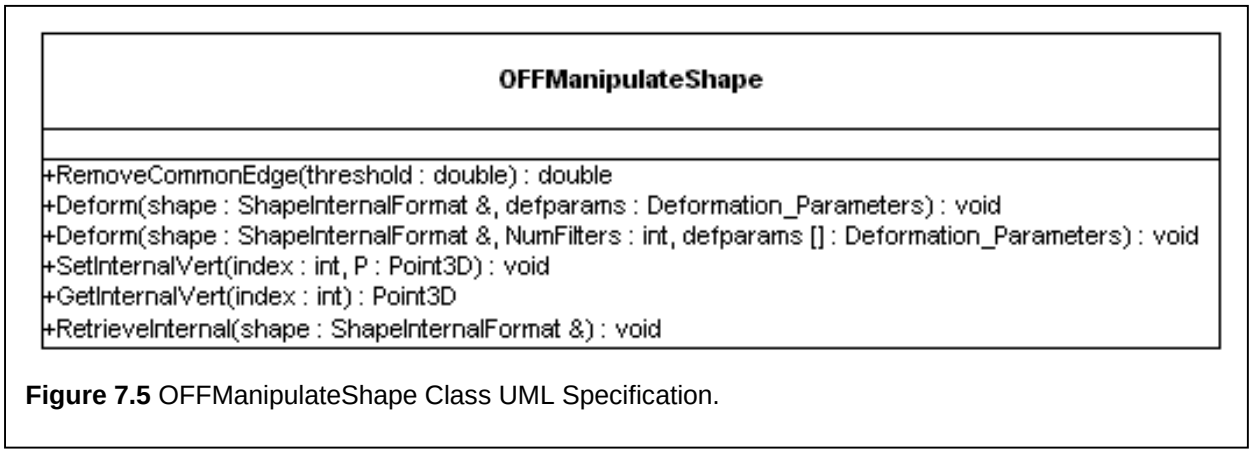


Figure 7.5 OFFManipulateShape Class UML Specification.

```
END FOR
```

This means that the vertex deformation algorithm alters the value of `V`, and the changed value overwrites what was previously stored in `V`. The reason for the duplication of object geometry in memory (obtaining the copy of the object's internal representation) is necessary when the `Deform()` method for a specific object is called a number of times.

he deformation algorithm permanently alters the internal representation of the object each time the deformation algorithm is applied to it. By making a copy of the unaltered object before any deformations are applied to the object, two objectives are achieved, namely:

- There is a guarantee that even though the object's actual internal `ShapeInternalFormat` may be incorrect, the deformation algorithm will always be applied to a completely unaltered copy of the object's original `ShapeDescription`.
- Generating the rendered output of the deforming object. Since the rest of the CoRgi system makes use of the internal representation of the object's `ShapeDescription`, the api-call overhead can be reduced because of the direct manipulation of the representation's data. The other alternative involves using the internal representation as the original, unaltered copy and generating a deformed copy of the original. This is more difficult and convoluted since it involves fooling CoRgi's rendering component into believing that the deformed copy is the actual internal representation.

The `OFFManipulateShape::RetrieveInternal()` method has been overridden to implement the object retrieval/copying process. It allows a snapshot of the entire object's internal polyhedral representation to be obtained. A copy is made of the internal structure, a pointer to which is then passed back to the calling application via the `ShapeInternalFormat` parameter (shape).

Two overloaded instances of the `Deform()` method exist. The first one involves the case where a single deformation must be applied to an object, while the second one adds support for multiple deformations. Concerning the first instance of the method, the `VertListFilter` parameter identifies the type of deformation that must be applied. If a `NULL` value is passed to this parameter, this implies that a global deformation must be performed on the object. If, however, it happens to be pointing to an array, the classification becomes a deformation filter. This approach is a very simple one, since the only real work involved in actually using the deformation technology is to make a call to `RetrieveInternal()` and then to the `Deform()` method, with a `NULL` value for the relevant parameter. It can thus be very easily deployed and integrated with any other existing systems.

This has been illustrated very successfully with the incorporation of this technology into a VR interaction system. The interaction system allows a user to grab hold of objects in a virtual environment. The addition of deformable objects to the interaction system implies that the user is provided with immediate visual feedback when he/she grabs the deformable object. When an object is grabbed, it is deformed in some way to illustrate that the grab has occurred.

The second `Deform()` method adds support for multiple deformations. The method accepts a list of deformation filters/global deformations with associated deformation parameters as input. Each entry in the list can be a deformation filter (a list of integers) or a NULL pointer (denoting a global deformation). The deformation parameters are also defined as an array of values for each of the deformation parameters. Hence, there are the same number of `R` (radius of influence) and `T` (max time) parameters as there are filters defined for the multiple deformations. The same applies for the rest of the parameters. Each vertex is deformed according to the list of filters and deformation parameters. Although it is possible to use the single `Deform()` method to perform multiple deformations by simply calling the method a number of times, the approach we have adopted can be equated to the way multi-texturing is performed. With multi-texturing, two or more textures are blended and applied to a surface in one pass. Similarly, the necessary deformations can be applied to each vertex in the order that the deformations are listed in the `VertListFilter` array. This is done for each of the shape's vertices. Aside from minimising the number of calls made to the `Deform()` method, there is another major benefit to this approach. It was mentioned above that the deformation algorithm uses a copy of the object's original structure to perform deformations on the object. Using the `Deform()` method that only handles individual deformations to perform multiple deformations implies that a copy of the deforming structure has to be made every time the shape is changed while generating the deformation result. The additional overhead is illustrated in **Figure 7.6**. When compared to

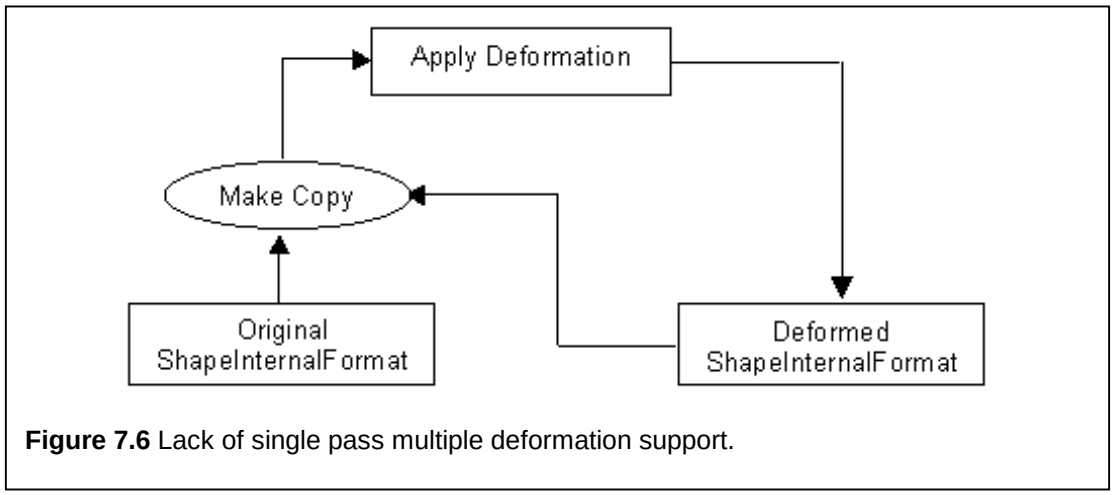


Figure 7.6 Lack of single pass multiple deformation support.

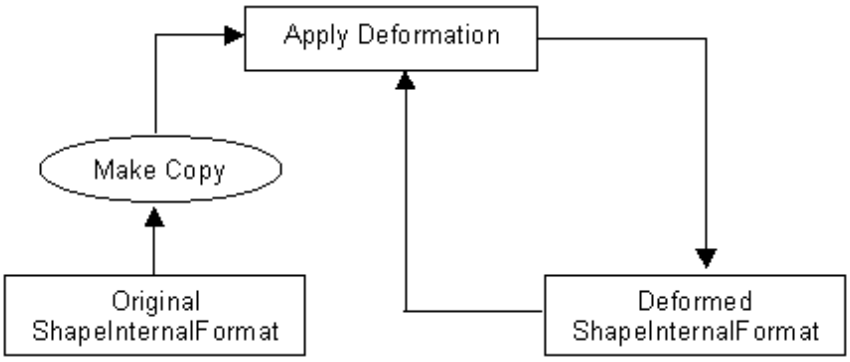


Figure 7.7 Supporting Multiple Deformations.

Figure 7.7, one can see that a copy of the object's structure must be made a repeated number of times with the `Deform()` method supporting only single deformations. A backup copy of the original unmodified structure is used. Multiple deformations may be applied to the shape. The result is then saved in the object's `ShapeInternalFormat`. The `Deform()` method supporting multiple deformations allows the deformation to be applied in a single pass.

The performance of the deformation system with this approach is also dependent on the number of deformations that have to be performed to generate an expression.

7.4.3.3 Displaying the animation

In order to display the deforming object while the deformation is being performed involves understanding the way the CoRgi graphics sub-system works. A typical scene consists of a number of objects upon which matrix transformations must be performed to generate the final rendered scene. An optimisation technique that OpenGL supports is the notion of "display lists". A display list can be defined as "a cache of commands rather than a dynamic database… once a display list has been compiled, it can't be modified" (Woo *et al.* [68], pp257). This means that an OpenGL program is evaluated and all the OpenGL commands are combined into a static structure; the commands are stored in the list for execution at a later stage. If the geometry of the object being rendered remains the same, it makes sense to use display lists because "you can define the geometry and/or state changes once and execute them multiple times" [68], pp 257 (for a more complete discussion on the benefits of and problems with display lists, look at Woo *et al.*'s discussion in [68]). In a scene where the objects' geometry does not change, the display lists for the scene only need to be generated once, at the beginning of the program execution. If, however, any of the objects in a scene change shape, the display lists for that scene have to be

recompiled. If this is not done, changes to the object geometry will not be propagated to the rendered output. Any object that is deformed is required to have its display list regenerated. Each object in a CoRgi scene is represented by an instance of the `VisualRepresentation` class. The `VisualRepresentation` class handles the generation of a display list for the entire scene. There is a `VisualRepresentation::update()` method defined that, when called, generates a new display list for the scene.

The animation process involves deforming an object repeatedly and displaying the object after each deformation has been performed. The deformation means that the current display list associated with the object is invalid, and a new one must be generated. In order to regenerate a new display list for every object structure that has changed, the current display list must be flagged as invalid. Upon execution of the `VisualRepresentation::update()` method, the invalid state of the display list is detected and new display list for the modified object generated.

The `VRSink` class is responsible for management of the final CoRgi render window; it constantly executes the `VisualRepresentation::Render()` method for each `VisualRepresentation` instance in a virtual scene. The `Render()` method checks if the display list for that instance is valid, and if it is not, executes `VisualRepresentation::Update()`, that ultimately recompiles the display lists.

7.4.3.4 Multitasking support

The deformation system was initially developed as part of a standalone application without any regard for integration with any other part of the CoRgi system. This means that the system was developed without the notion of multitasking; the rest of the system had to wait for a deformation to be competed. The initial system was based on the co-operative multitasking paradigm: complete what must be done and then let the rest of the system do what it has to.
Execution was based on the following algorithm:

```
LOOP
   Perform Deformation for single time unit
   Regenerate Objects Display List
   Update the Deformation's time component
UNTIL target time has been reached
```

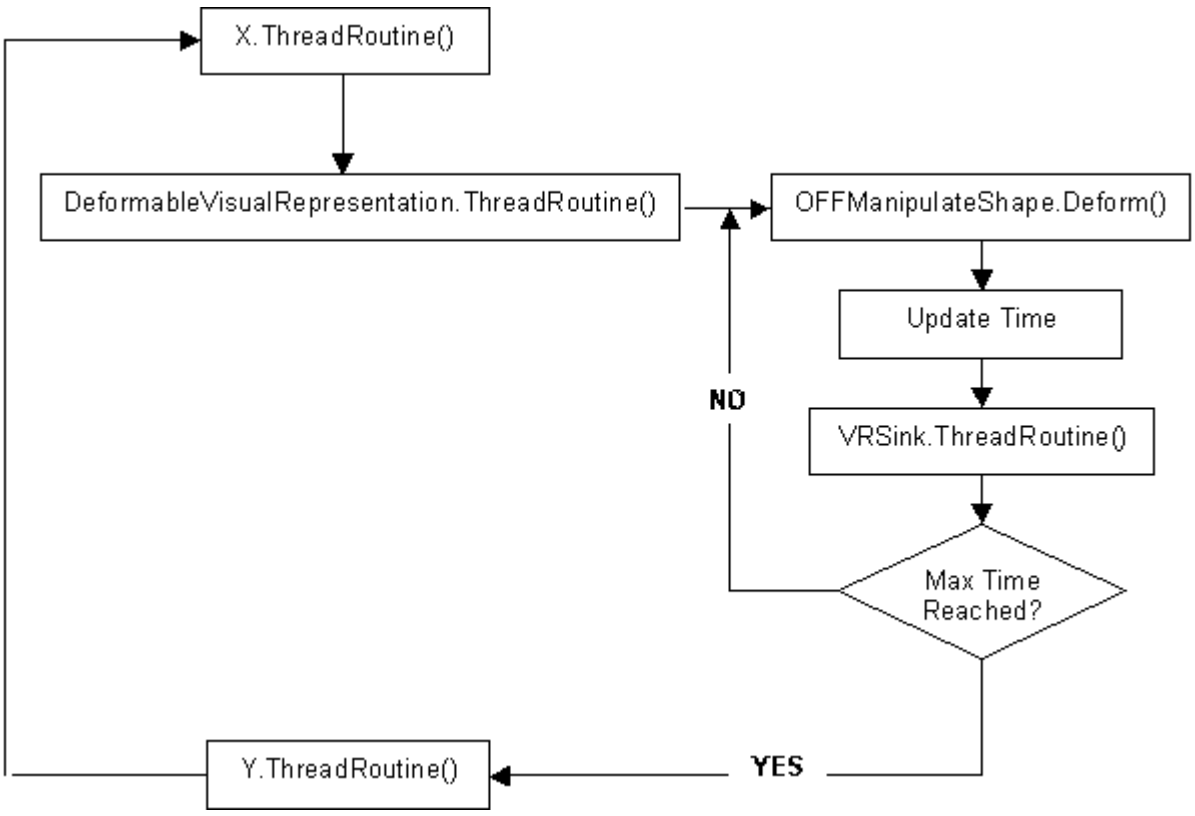


Figure 7.8 Component Flow Execution.

This execution model is illustrated in **Figure 7.8**. The lack of multitasking support is evident. The `DeformableVisualRepresentation` class has complete control over the execution flow until the deformation has completed

Implementing a multitasking-enabled deformation system involves three tasks:

- Performing a single deformation from the animation at time `t`;
- Updating `t` by `Time_Delta`;
- Returning control to the rest of the system.

These three control rules break the deformation animation down into its discrete components, thereby enabling the rest of the system to function while the animation occurs.

To facilitate a discrete animation generation approach, the deformation implementation had to be embedded in a `Component`-derived object. The reason for this is has been mentioned before: a characteristic that is common to all `Component`-derived objects in CoRgi is the overloading of the `ThreadRoutine()` method. Any instance of a `Component` object will lead to the execution of its `ThreadRoutine()` method during execution of the program.

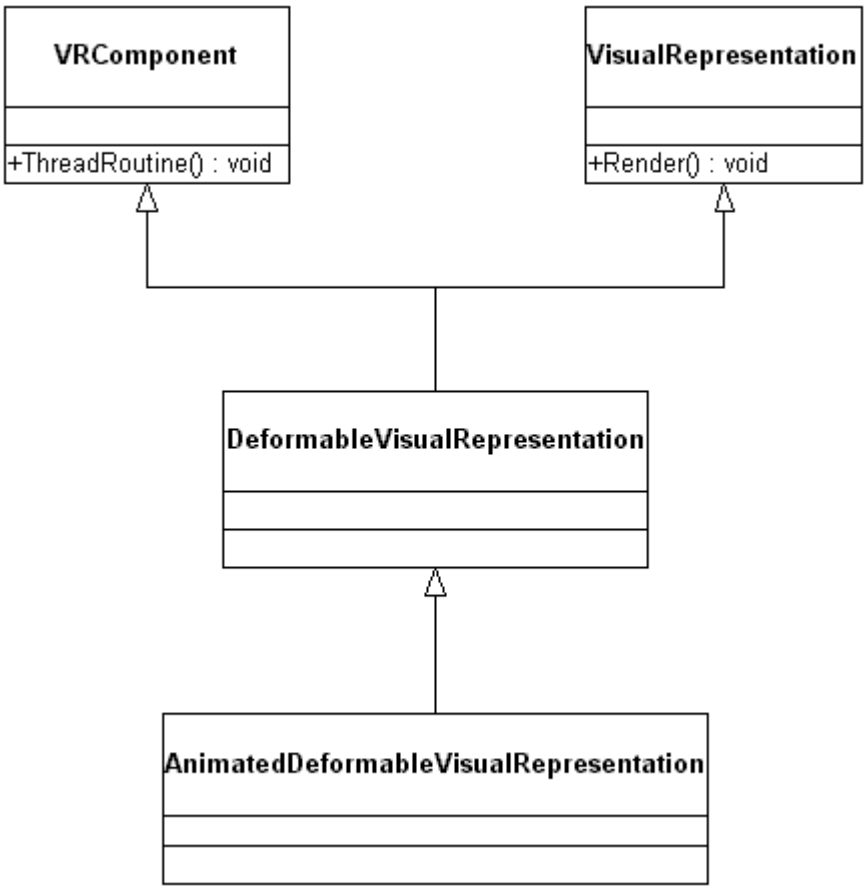


Figure 7.9 AnimatedDeformableVisualRepresentation Class Hierarchy UML Specification.

The final implementation has led to the development of a class called `DeformableVisualRepresentation`. The new class performs the same function as a standard `VisualRepresentation` (for managing the internal OpenGL object representation and display lists), while at the same time providing the same functionality as a CoRgi `Component` (for adding multitasking to the deforming animation, through the `ThreadRoutine()` method). Its definition is illustrated in **Figure 7.9**. The final class is one that is derived from both the `VisualRepresentation` and `VRComponent` classes.

It was also mentioned in earlier that the `ShapeDescription` base class has been subclassed to the `OFFManipulateShape` class. The constructor of the `VisualRepresentation` class is takes as a parameter a pointer to a `ShapeDescription` object. As such, the `DeformableVisualRepresentation` class is passed as pointer a `ShapeDescription` base class that refers to an `OFFManipulateShape` object, in the following way:

```
ShapeDescription *shape
shape = new OFFManipulateShape(filename)
VisualRepresentation *def
def = new DeformableVisualRepresentation(shape)
```

where `filename` refers to the 3D object data file.

`DeformableVisualRepresentation::ProcessDeformation()` has been defined as part of the class's interface and deals with the generation of both single deformations and **multiple single-pass deformations**. The term "multiple single-pass" refers to deforming an object with more than one deformation at the same time.

The final `DeformableVisualRepresentation::ThreadRoutine()` method has the following structure:

```
DeformableVisualRepresentation::ThreadRoutine()
BEGIN
     ProcessDeformation()
END
```

A data structure called `GlobalDeformation` exists that contains the current deformation parameters. The `DeformableVisualRepresentation::ProcessDeformation()` method passes the information stored in this data structure to `OFFManipulateShape::Deform()`. The use of the `ProcessDeformation()` method allows for a logical partitioning between the deformation process and the remainder of the system.

The implementation of `DeformableVisualRepresentation::ThreadRoutine()` in this way adds deformation animation support to CoRgi in a cooperative fashion. The deformation implementation allows the rest of the components defined in the application to execute their respective `ThreadRoutine()` methods after every frame of the deformation animation.

With respect to the integration of the deformation implementation with the rest of the system, the added multitasking ability allows the decoder to evaluate the incoming network packets and move the avatar around while the deformation-based animation occurs.

`DeformableVisualRepresentation` underlies the implementation of the Multiple Deformation Management tier of the three-tier expression editing system discussed in the previous chapter. The final tier, namely Avatar Expression Management is handled by the extension of `DeformableVisualRepresentation` to define a new class called `AnimatedDeformableVisualRepresentation`, as illustrated in **Figure 7.9**. This class

abstracts the process of setting up a multiple deformation handled by `DeformableVisualRepresentation` to a simple control mechanism whereby the selection of an expression using `AnimatedDeformableVisualRepresentation` translates to appropriate calls to `DeformableVisualRepresentation` to set up the correct deformations.

7.4.3.5 Deformation Filter Management

The use of deformation filters implies the need for an internal representation. The `VertexFilterListNode` class is a data structure that represents the currently loaded filters. The `VertexFilterListNode` class contains a linked list of deformation filters and associated deformation information, as illustrated in **Figure 7.10**. Each of the nodes in this linked list is of type `VFListNode`. The UML class definition is illustrated in **Figure 7.11**.

The `VertexFilterListNode` class affords a high level of control over the deformation filters as well as easy access to the filter data. Changes that are made to any of the deformation filters can be saved to disk.

7.4.3.6 Animation Management

This section describes the process of combining individual deformations to form expressions that are more complex. To accommodate this, the classes above, namely `VFListNode` and

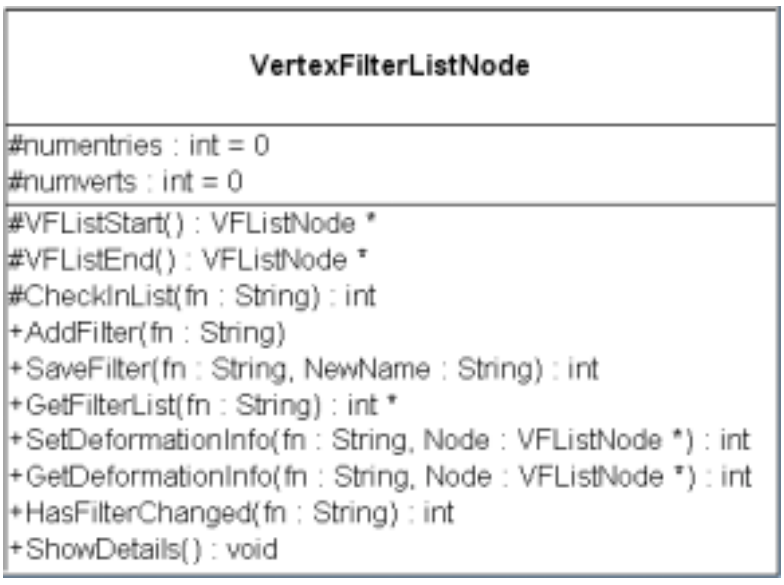


Figure 7.10 VertexFilterListNode UML Class Definition.

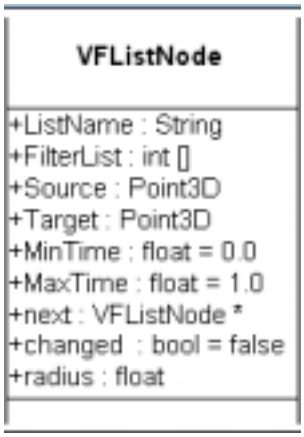


Figure 7.11 VFListNode UML Class Definition.

`VertexFilterListNode`, have been extended to support multiple deformations. The extensions involve two enhancements to the above implementation, namely:

- The addition of timing information to each node of the linked list, namely the `VFListNode` class, in `AnimatedVFListNode`;
- The addition of methods to control the construction of expressions based on combinations of individual deformations. The class that handles this responsibility is called `AnimatedVertexFilterList`. This class is derived from `VertexFilterListNode`.

The `AnimatedVFListNode`'s derivation is illustrated in **Figure 7.12**.

The newly added information provides all the necessary information for the generation of a deformation-based animation. While an animation is being performed, `CurrTime` is constantly updated. This value is incremented by `TimeDelta * Direction`. This translates to the following alteration to `CurrTime`:

```
CurrTime ← CurrTime + TimeDelta * Direction
```

As it stands, two compromising situations must be prevented, namely:

- When `CurrTime` exceeds the current filter's max time, i.e.:

```
CurrTime > VFListNode→MaxTime
```

- `CurrTime` becomes smaller than the `StartTime` (typically `0`).

If either of these situations arises, the direction of the increment is reversed, in other words:

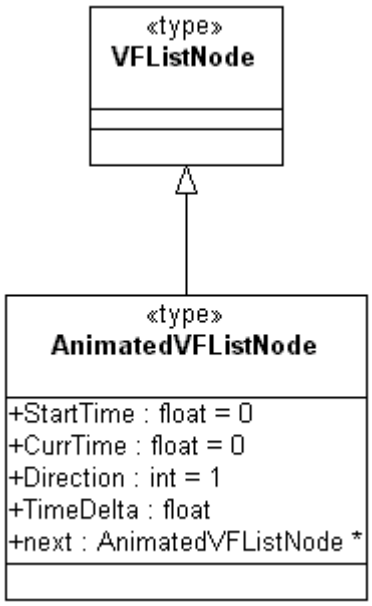


Figure 7.12 AnimatedVFListNode Class Hierarchy.

```
Direction ← -1 * Direction
```

In this way, a deformed model can be deformed back to its original state. This is because the `TimeDelta*Direction` increment to `CurrTime` becomes negative and so doing returns to zero, the point of no deformation.

In addition to handling the construction of expressions by grouping individual expressions, the new `AnimatedVertexFilterListNode` is responsible for implementing the new `AnimatedVFListNode`-based linked list. The reason for this is that a new linked list management class has to be defined to deal specifically with this aspect. The definition for the new class is illustrated in **Figure 7.13**. The `AnimatedVertexFilterList()` class provides control over the animation playlist and allows multiple deformations to be specified.

7.4.3.7 Controlling the Animation (`DeformableVisualRepresentation`)

The discussion so far has concentrated on the design and implementation of the basic deformation system. At this point all the facilities are in place to begin implementing higher-level animation deformation management. The `DeformableVisualRepresentation` class handles this process.

Its class interface is illustrated in **Figure 7.14**. As can be seen, a number of protected properties (marked with #'s) exist that specify what the currently active deformation parameters are. The idea is that for any object being deformed, the deformation subsystem need only refer to these properties to determine the parameters of the deformation. The properties (with their respective initial values) being referred to are as follows:

DeformableVisualRepresentation Constructor Initialisation	
Property	**Constructor Initialisation Value**
`DeformationQueue`	`new AnimatedVertexFilterList()`
`ListName`	`NULL`
`DeformationFlag`	`FALSE`
`DoMultipleDeformations`	`FALSE`
`DeformFilter`	`new VertexFilterListNode()`

The initialisation of each property within the class constructor is listed above in square brackets. `ListName` is used to identify whether a global deformation or a deformation filter

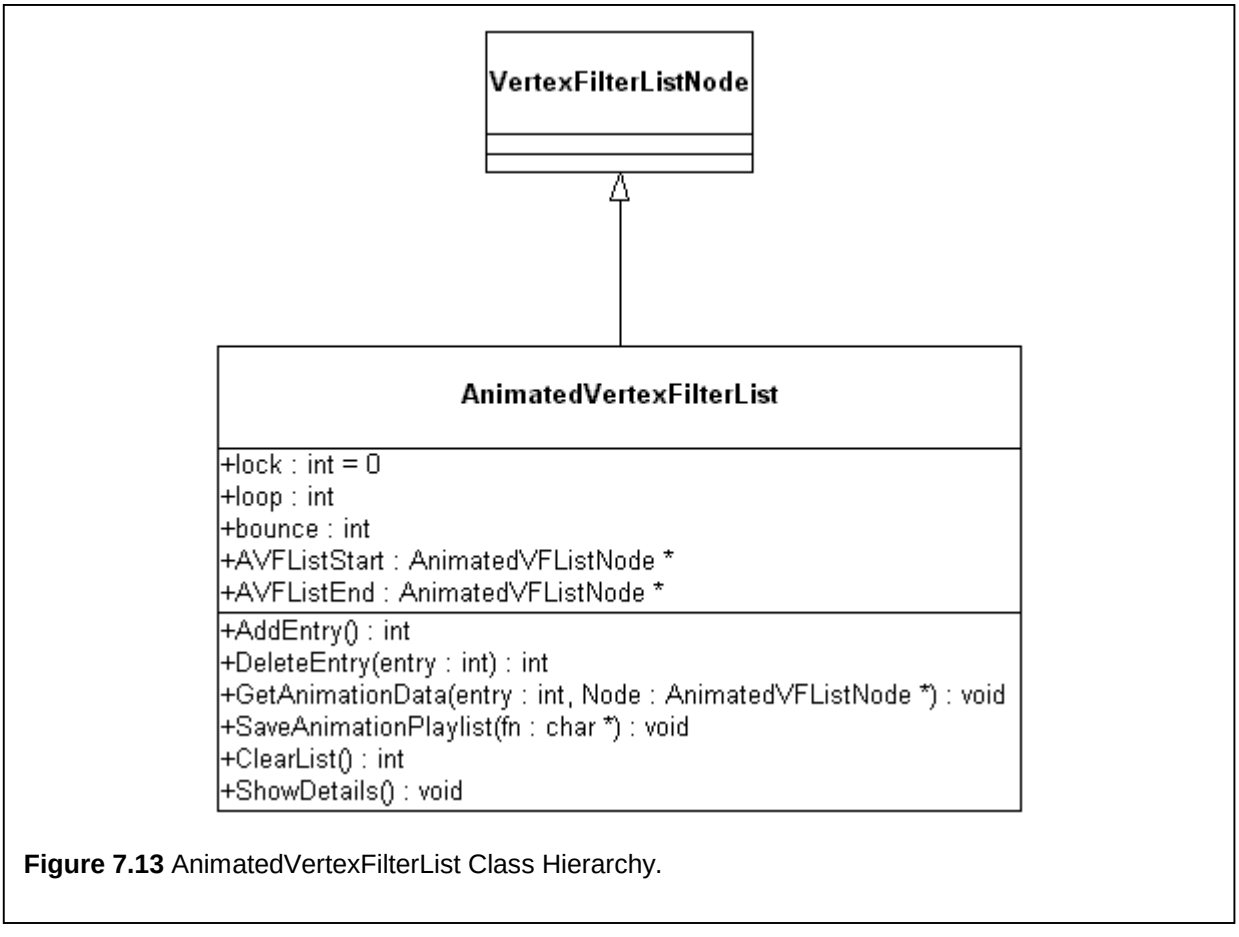


Figure 7.13 AnimatedVertexFilterList Class Hierarchy.

must be applied to the object. As mentioned above, a global deformation implies that the deformation must be applied to the entire object. If `ListName` is `NULL`, the deformation information is interpreted as being part of a global deformation. If not, it represents an entry in the `DeformFilter` linked list of deformation filters.

The `DeformFilter` data structure is an instance of a `VertexFilterListNode` class. As such, it represents all the currently loaded deformation filters.

`DeformationFlag` identifies whether the deformation system is enabled (`DeformationFlag` ← `TRUE`) or disabled (`DeformationFlag` ← `FALSE`). The `PerformDeformation()` triggers deformation calculations depending on the state of this variable. `DoMultipleDeformations` is similar in nature to `DeformationFilter`. A value of `TRUE` indicates that multiple deformations must be performed, while `FALSE` indicates that the animation will be based on a single deformation filter.

The next step in the implementation involved integrating these new classes into the rest of the

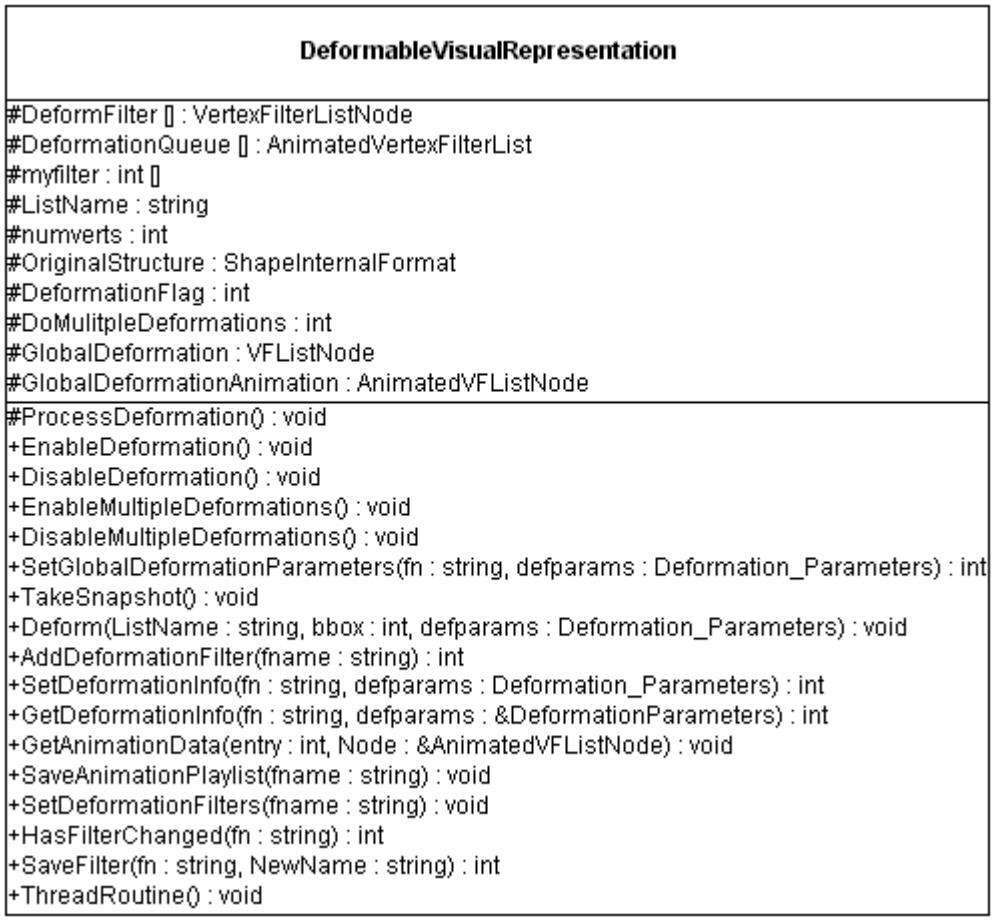


Figure 7.14 DeformableVisualRepresentation UML Class Definition.

deformation system. The `DeformableVisualRepresentation` class has been designed around the following paradigm: at any one time there is only one valid data set, viz. a single set of deformation parameters and associated deformation filter. The `GlobalDeformation` property is used to perform this function.

`DeformableVisualRepresentation::ProcessDeformation()` generates deformation animations based on the information stored in the `GlobalDeformation` and `GlobalDeformationAnimation` objects. The former holds the currently valid deformation parameters, as well as the deformation filter if a filter is used. The latter represents the timing information used to generate the deformation animation, i.e. the current time. As the deformation occurs, so the time parameter is updated.

In order to propagate changes in either deformation parameter of the deformation filter through to the final deformation animation, the `GlobalDeformation` object's data needs to be appropriately altered. This constraint provides the facility for an elegant solution to the problem of the user specifying different deformation parameters in the final application. The idea here is that if the user, for example, decides to change any of the deformation parameters, or even select a different deformation filter for the animation, the implementation dictates that these changes need to be made in only one place, namely `GlobalDeformation`.

The algorithm that `ProcessDeformation()` implements is as follows:

```
IF DeformationFlag THEN
BEGIN
   IF NOT DoMultipleDeformations THEN
   BEGIN
     GDA.CurrTime ← GDA.CurrTime + GDA.Direction *
     GDA.TimeDelta
     If GDA.CurrTime > GDA.MaxTime OR GDA.CurrTime < 0 THEN
         GDA.Direction ← -GDA.Direction
     END IF
     IF GDA.CurrTime > GDA.MaxTime THEN
          GDA.CurrTime ← GDA.MaxTime
     END IF
     IF GDA.CurrTime < 0 THEN
     BEGIN
        GDA.CurrTime ← 0
        DeformationFlag ← FALSE
     END IF
     OFFManipulateSHAPEshape::Deform(X)
     Current_Display_List ← Invalid
   END IF
END IF
```

where:

- G represents the GlobalDeformation class,
- GDA represents GlobalDeformationAnimation, and
- X represents the current deformation parameters for G, namely: the filterlist, the Source and Target force combination etc.

The algorithm provided above is for the deformation of using only a single deformation filter. The generalisation to multiple deformations involves, firstly, setting up the relevant deformation parameter arrays, updating TimeDelta for each of the deformations that must be performed, and then calling the appropriate OFFManipulateShape::Deform() method. There are two overloaded instances of the Deform method defined in the OFFManipulateShape interface. The one deals with single deformations, while the other handles multiple deformations.

7.4.3.8 Controlling Expressions (AnimatedDeformableVisualRepresentation)

During the process of receiving Expression IDs from the encoder application, a conversion to the appropriate deformation control data is performed. The task of the AnimatedDeformableVisualRepresentation (AnimDefVisRep) class is to extend DeformableVisualRepresentation's (DefVisRep) functionality to handle the management of the deformation data used to control the deformation process. This means that if an encoder specifies that a smile must be generated, the data structures used by DefVisRep must be altered to reflect the correct deformation filters and deformation parameters.

The interface of the class is illustrated in **Figure 7.15**. All expression IDs received by the decoder is mapped to the appropriate list of deformation parameters. As can be seen by the above class interface specification, the AnimDefVisRep class introduces methods for the management of expressions.

Each expression is represented by the AnimatedVertexFilterList linked list class. Its linked list contains all the individual deformations that will generate that specific expression. The two-dimensional ExpressionList property represents all the expressions that are associated with a specific avatar. Associated with this property is the ExpressionNames property, which provides lookup functionality.

The SetExpression() method allows the deformation parameters for a specific expression to become the active parameters that are used to deform the model. The parameter

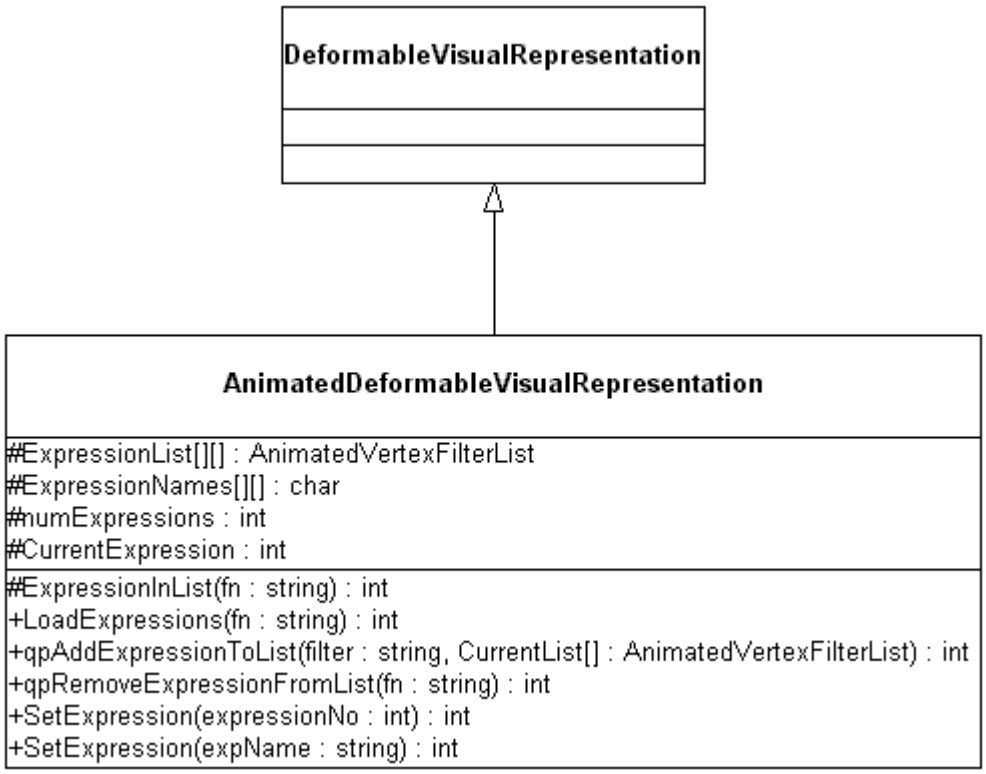


Figure 7.15 AnimatedDeformableVisualRepresentation Class Hierarchy.

passed to the `SetExpression` method is used as a lookup value into the `ExpressionNames` and `ExpressionList` arrays. Once the entries relating to the parameter have been identified, the `GlobalDeformation` and `GlobalDeformationAnimation` properties inherited from `DeformableVisualRepresentation` are updated to represent the deformation parameters being pointed to.

7.5 Technology dissemination

The integration of the NCC-based tracker and the deformation algorithms with standard CoRgi components has added much-needed functionality to the system as a whole. Because the implementation of the different classes is built on top of the standard base classes that comprise CoRgi, their functionality can be quickly incorporated into any of the existing applications. A field where the deformation implementation has been adopted is a VR interaction system that is under development. This facilitates, amongst other interaction techniques, the grabbing of objects in a virtual environment. The deformation system has been used to define a

`VRDeformableEntity` object that, when grabbed, is deformed using the deformation system. This provides a visual feedback mechanism that enhances the interaction experience.

7.6 Performance Issues

On the whole, the system performance depends on two factors, both of which are dependent on the speed of the host processor. On the encoding side, the speed of the NCC tracker is the primary limiting factor. At the decoding end, the rendering subsystem constrains the system performance. In addition, the overhead of performing deformations is noticeable, but not overly excessive.

With the current implementation, the performance of the encoder operates at 10 frames per second, including the overhead imposed by the pose estimation and expression analysis processes. The performance of the decoder is dependent primarily on the size of the object; more vertices implies higher performance overhead during deformation. It manages to achieve approximately 14 fps with an avatar containing 7500 vertices on a dual 195 MHz MIPS R10000 SGI Octane architecture with 192 MB RAM.

The compression rate has been discussed at length in section 5.6.7. To summarise, a transmission rate of 2 Kbps has been achieved, given the tracker's 10 fps performance measurement and the transmission of: the Avatar ID, the position and orientation information, and the Expression ID.

7.7 Summary

This chapter describes the implementation of the NCC-based head tracker and Vertex-interpolation-based deformation systems under CoRgi, a component-based, distributed virtual reality system that enables the quick development of distributed multimedia, specifically VR-based applications.

Three aspects are discussed, namely: the networking considerations, the encoder application, and the decoder implementation.

Section 7.4.1 discusses the networking implementation. It involves to the CoRgi networking model (for [un]marshalling aspects).

Section 7.4.2 details the component-based implementation upon which the encoder application is built. The class underlying the implementation, namely `VideoAvatarTracker`, is derived from two very distinct components, namely:

- `VRPuppetControlInputDevice` (discussed in section 7.4.2.3). This component acts as a device generating puppet/avatar control data through its `GetData()` method;
- `VideoTracker` (section 7.4.2.4). The implementation of the NCC-based tracker discussed in the previous chapter. It is a dumb tracker component able to track an unlimited number of points. It has facilities to perform corrective measures if the user decides at any point that a tracking error has occurred.

`VideoAvatarTracker` inherits `VRPuppetControlInputDevice`'s data generation ability and `VideoTracker`'s tracking ability, as discussed in section 7.4.2.5. It is ultimately responsible for computing pose and classifying subject expressions. For the final component specification, this device is connected to a `VRSource` component that is, in turn, connected to a `VRNetworkInputDevice`. The former handles collections of the data generated by the device, while the latter packages the data up and sends it across the network.

The development of the decoder application utilising the VID system highlights a number of shortcomings with the CoRgi system that are addressed in section 7.4.3. Additionally, this section also discusses the implementation of the final decoder application.

Section 7.4.3.1 discusses the lack of access to object data structures. A method is introduced to facilitate read/write access to the internal avatar geometry. Following this, a discussion in section 7.4.3.2 on the implementation of the Vertex Interpolation Deformation algorithm is provided.

Section 7.4.3.3 details the addition of multitasking support during expression generation. This involves the creation of `DeformableVisualRepresentation` that inherits from both the `VisualRepresentation` and `VRComponent` base classes. Subsequent to this, the `VisualRepresentation::ThreadRoutine()` is overridden to facilitate multitasking in a cooperative fashion.

With deformations supported, the remainder of the decoder discussion focuses on supporting the three-tier expression editing paradigm that is developed in the previous chapter. The lowest tier, namely individual deformations is encapsulated within the `OFFManipulateShape` class (section 7.4.3.2). A hierarchical implementation representing the remaining tiers is as follows:

- `DeformableVisualRepresentation` (section 7.4.3.7) to handle a single collection of deformations (i.e. a single expression); and
- `AnimatedDeformableVisualRepresentation` (section 7.4.3.8) for handling high-level management of groups of expressions.

7.7.1 Contributions

The CoRgi system has facilitated a high-level implementation of the system design detailed in Chapter 2. Excellent compression rates, and subsequently, orders of magnitude reduction in bandwidth requirements, have been achieved.

The following contributions are made to the CoRgi system:

- Image-based NCC tracking;
- Platform independent networking
- The addition of a three-tier deformation control mechanism to the system for expression generation and management;
- The use of CoRgi's `ThreadRoutine` approach to perform cooperative execution of the deformation implementation;
- The creation of a CoRgi device, namely `VideoAvatarTracker` that combines two very distinct areas of the CoRgi system, namely multimedia and VR.

Chapter 8 - Conclusions and Future Work

8.1 Summary and Conclusions

8.1.1 Introduction

This thesis discusses the development of a framework to solve the problem of model-based video coding using a component-based Virtual Reality system called CoRgi. The goal is the conversion of a live video stream into avatar control data.

The final system has to possess the following characteristics:

- It has to be cheap. The implementation had to be practical using standard desktop workstations;
- Its design has to be component-based;
- It has to be implemented using a Virtual Reality System;
- It has to possess the ability to generate realistic avatars.

These goals constrain the choices made during development of the final system.

The system is decomposed into two stages: encoding and decoding. The encoding process involves: model acquisition, pose estimation and expression analysis. Decoding consists of: avatar management, avatar movement and expression generation.

8.1.2 Model Acquisition

8.1.2.1 Summary

A quantitative analysis of different reconstruction approaches has been performed and the most appropriate method chosen. The approaches compared are: Laser Range Finder techniques, a collection of Light Illuminated Triangulation approaches, NCC-based depth reconstruction and a Visual Hull approach. The factor dictating the choice of reconstruction approach is cost, both economic and computational. The reconstruction approach that is found to require the least cost as well as the least amount of post-processing is the visual hull class of reconstructions.

The reconstruction implementation is based on the concept of a reconstruction pipeline. The object to be reconstructed is assumed to lie in a bounding volume of voxels. Images of the objects are taken of the object from different angles and projected into the bounding volume to recover the basic shape of the object. The pipeline consists of five components, namely:

- Space carving for the determination of the object's basic shape;
- Isosurface extraction for the generation of a hollow shell of voxels representing the surface of the object. This component can be likened to a 3D filter;
- Mesh connection to facilitate the connection of the remaining voxels and thus the generation of a closed, connected polyhedral mesh.
- Texture generation, facilitated through the creation of a vertex colour map. During rendering, traditional texturing is simulated through the use of the colour map to Gouraud shade all polygons representing the reconstruction;
- An optional, efficient mesh decimation component that attempts to locate contiguous rectangular patches of voxels in the final reconstruction and replace them with two triangles.

The result is a reconstruction algorithm that has a linear relationship with the resolution of the voxel-based bounding volume, with the performance being dependent on the number of voxels in the bounding volume. Additionally it is shown that none of the components constituting the pipeline noticeably affects the execution time of the pipeline.

Because it assumes that every voxel is initially part of the object, hole-filling as a post-processing task is avoided. The final reconstruction does also not depend on surface colour. Light Illuminated Triangulation techniques that typically use red laser light suffer from this problem when trying to reconstruct surface that are black or surfaces that are highly diffractive.

The current implementation suffers from two problems, namely: an inability to generate realistic textures, and an inability to reconstruct surface concavities that are not part of the object's visual hull. An enhanced NCC space carving algorithm is proposed that should in theory be able to perform concave surface reconstruction. The development of this algorithm arises from the following observation: a 3D point in space observed from different positions will have different colours, while a 3D point on the surface of an object observer from different positions will return the same colour, bar lighting effects. After space carving has been performed, the colours that map to each voxel can be used to determine whether it is indeed on the surface of the object or not. A possible problem with this approach is the inability of the NCC algorithm to handle bland areas of the object's surface very well.

It is also shown that by varying the bounding volumes used, progressively more detailed reconstructions are generated. This leads to the possibility of the use of the reconstruction technology for Level of Detail reconstructions.

8.1.2.2 Conclusions

- This low-cost reconstruction solution (utilising only an inexpensive CCD camera) is a valid alternative to expensive, proprietary technology. In certain situations, this approach generates better results than the more expensive solutions;
- The use of a reconstruction pipeline suits the component-based nature of the higher-level system design. It provides a logical decomposition of the reconstruction process into its most important tasks, thereby simplifying a very complex problem.
- The algorithm is unable to reconstruct surface concavities that are not part of the object's outline shape. This constraint is not as serious as initially anticipated. The visual hull algorithm works on an object's outline shape, thus being suited to the reconstruction of a large class of objects;
- The proposed use of an NCC evaluation to perform reconstruction of concave surfaces that do not form part of the object's outline is very important. The primary constraint of visual hull algorithms is thus removed;
- The results obtained from the visual hull reconstruction mechanism show that this class of algorithm is well suited to the low-cost reconstruction of generic objects, and not avatar heads specifically.

8.1.3 Encoding

8.1.3.1 Summary

The encoder encapsulates both the pose estimation and expression analysis component, because both these components are recovered from an evaluation of the movement of a number of fiducial points on a subject's face. The fact that both components utilise the same underlying technology, namely an image-based tracking algorithm, acts as further motivation for the grouping together of these components.

The image-based tracking mechanism employed utilises an NCC evaluation. It is shown that this algorithm lends itself to robust image-based tracking that is shown to be invariant to rotations and scaling of the subject. The speed of the NCC tracker is dependent on three parameters, namely: the frame size, the search window size, and the correlation window size. It is shown that the requirements of real-time, robust tracking are achieved by experimenting with these parameters.

The selection of fiducial positions on a subject's face is user-specified. The choice of fiducials is constrained by a fiducial graph representing both eyes, the nose tip, both mouth edges and the chin tip of a subject in a given image. The fiducial graph provides a subject-independent face specification mechanism.

During the tracking process, the user continually evaluates the tracking results as the fiducials are tracked. If at any point a tracking error occurs due to occlusional discontinuities or overly fast motion, the user issues a warning to that end and corrective measures are adopted. The corrective measures involve using a caching mechanism, whereby previously correct fiducial graph is used to continue tracking.

The pose estimation process involves triangulation of the nose, left eyebrow and right eyebrow fiducials to determine a final 6-DOF quaternion representing the subject's pose. This approach is chosen because of its perceived simplicity; each of the pose parameters (roll, pitch and yaw) is derived independently. The translation vector specifying the position of the user is estimated utilising the movement of the user's head parallel to the image plane, as well as a scaling factor to determine the distance of the head to the camera. The scaling factor is determined using a ratio approach viz. the current over the original. The problems that plague traditional analytical perspective solutions to Pose Estimation, namely ambiguity, are avoided. This is achieved by adding a number of intelligent observations to the pose estimation process.

The expression identification process involves the rule-based evaluation of fiducials surrounding the subject's mouth. The Expression Parameter Lookup Table provides a mechanism for mapping an expression that is identified to a method of expression generation. Each entry consists of an Expression ID, subject-specific expression analysis rules and model-specific expression generation rules. Every avatar used with the system is thus constrained to have a collection of rules for each expression in the database for expression generation. Similarly, it is assumed that every avatar has an associated collection of deformation entries defined for each entry. The subject-specific expression rules for each expression are derived from subject-independent threshold rules that apply for the collection of universal expressions. Additionally, the EPLT is extensible in that the rules can be tailored to each subject if need be,

and additional entries added to the lookup table. The result is that only the Expression ID is transferred to the decoder for each expression identified using the threshold rules during the analysis process.

It is shown that the maximum true angle that the pitch implementation is able to recovery is 30^0 (61^0 measured), while the yaw recovery fails at 40^0 with a measured/computed angle of 33.1^0. The roll recovery is the best, with computed angles being returned for a series of rotations between 0^0 and 90^0. It is also shown that the scale implementation is able to identify head movement towards and away from the camera.

For each frame of video evaluated by the encoder, the encoder facilitates a substantial reduction in bandwidth requirements. This translates to a reduction in bandwidth requirements from approximately 30 KB of data (100x100 uncompressed frame of video) down to 25 bytes for each frame of video evaluated. Given a tracker performance of 10 fps, the final transmission rate for the connection between the encoder and decoder is about 2 Kbps. The bandwidth usage increases linearly with an increase in the tracker performance. A 30 fps tracker performance thus translates to a 6 Kbps transmission rate.

Enhancements that are discussed include: an improved solution for pose estimation, a solution to the problem of drift, and tracking that utilises background elimination.

8.1.3.2 Conclusions

- NCC-based tracking is a viable alternative to more expensive electromagnetic tracker solutions. Its robustness to scaling and rotations makes it well-suited to facial feature tracking. Although it is a slow algorithm, the ability to modify the input parameters implies that it is flexible, and that real time tracking rates can be achieved;
- The pose recovery results illustrate that, by modelling the face as a plane, the results that are obtained, while not completely accurate, are acceptable for approximating head movement. Furthermore, the recovery of each component (roll, pitch and yaw) individually appears to be problematic, with a better solution being required.
- The user-assisted nature of the tracking feedback loop and caching history facilitates intelligent corrective measures that simplify the tracking algorithm. The result is a fast implementation.
- The use of the EPLT system provides a mechanism to abstract subject-specific expression analysis rules from the avatar-specific expression generation parameters, an important requirement for reducing bandwidth requirements. With every expression identified, only

the Expression ID need be transmitted to the decoder. The function provided by the lookup table is very important in ensuring an efficient remote expression replication mechanism. Furthermore, the extensibility and flexibility of the EPLT is crucial for the addition of new expressions to the EPLT, and also for tweaking specific threshold rules for each entry;

- The conversion of a video stream to avatar control results in a remarkable reduction in bandwidth requirements, making this technique perfectly feasible for very low-bandwidth connections;
- The bandwidth requirements depend on the performance of the encoder since the amount of control data transferred across the network depends on the number of frames that are processed by the encoder. Considering that a 30 fps processing rate requires only 6 Kbps, any modern network connection can comfortably handle this capacity. This implies that the bottleneck with this conferencing application becomes the encoder, and is no longer the network.

8.1.4 Decoding

8.1.4.1 Summary

The responsibilities of the decoder are: avatar movement, expression generation, and expression management.

Avatar Movement is defined as being a function of the underlying VR system. As such, it has not been discussed.

The expression generation system is implemented on top of a Vertex Interpolation–based deformation system that is able to perform deformations in real time because of the avoidance of C^1 and C^2 surface continuity and self-intersection issues that traditional FFD consider during deformation. An expression is classified as a collection of individual deformations that, when collectively applied to the avatar, will achieve a desired facial animation of an expression. The development of an expression editing system is discussed. Its primary function is to provide a method determining model-specific deformation parameters for each generic expression entry in the Expression Parameter Lookup Table. Expression editing is solved using a three-tier solution, namely: single deformation management for controlling individual deformations, multiple deformation management to facilitate combining one or more single deformations to represent an

expression, and finally, avatar expression management for the purposes of selecting expressions and testing the expression generation at a conceptually high level.

A weighting system is proposed to gradually transform the avatar from one expression to another, and thereby provide mechanisms for expression management.

8.1.4.2 Conclusions

- Even though vertex interpolation deformation is not as complicated as traditional FFD, it is suited to facial expression generation. This is observation arises from the observation that the individual facial movements that comprise a given expression do not cause too much change to a subject's face. As such, it is well suited to realistic facial expression replication.
- The VID system is beneficial because of its simplicity. It facilitates real time expression generation.

8.1.5 Virtual Conferencing under CoRgi

8.1.5.1 Summary

The integration of the NCC-based tracker and vertex interpolation deformation system with CoRgi forms the basis of the application development under that system. The development of an encoder application and a decoder application is discussed. The encoder application utilises NCC tracking technology overlaid on top of a live video stream. User assisted feedback is facilitated through the use of a video feedback loop that displays the video read from the camera as well as the fiducial points being tracked overlaid on top of the video feed. The points that are tracked are converted firstly into fiducial graphs, after which pose estimation and expression analysis is performed.

The networking implementation involves conforming to the CoRgi networking model (for [un]marshalling aspects). This provides a mechanism for platform-independent networking.

The discussion relating to the decoder implementation is handled from the perspective of the way the CoRgi system works with regard to a specific function, how the system design fits into CoRgi, what is required from the CoRgi architecture to achieve the desired implementation, and the enhancements that are implemented. The results of these improvements are:

- The ability to access the access of the internal object representation;
- The development of a cooperative multitasking paradigm to facilitate concurrent deformation and rest of system execution;
- The three-tier deformation system developed during the design of the decoder leads to a hierarchical implementation for the handling of individual deformations, single expression generation, and finally, the management of expressions at a high level.

8.1.5.2 Conclusions

- The development of a system such as this, where the encoder and decoder can exist on different architectures, has highlighted low-level networking issues required to facilitates platform-independent communications;
- The development of the encoder and decoder applications highlights the following core notion: the CoRgi system consists of components that define standard ways of performing certain actions, such as the `ThreadRoutine()` mechanism for component execution, or the `GetData()` mechanism for data generation using devices. Each application to be developed using the CoRgi system should be defined in terms of components, each of which should be classified according to a base CoRgi component, and its functionality defined in terms of the standard mechanism defined for the relevant CoRgi component.

8.1.6 General Conclusions

The requirements defined in section 8.1.1 have been met:

- The final solution is cheap. The implementation of each component that forms part of the final system design is based on low-cost technology; technologies that have been identified as viable alternatives to more expensive solutions form the basis of each component's implementation. This approach leads to a final implementation that is practical using standard desktop workstations.
- The design is based on a functional breakdown of the problem into its various components. It thus meets the requirement of being component-based;
- A sample implementation of the final system design has been successfully implemented using CoRgi.
- The model acquisition system has been shown to have the ability to generate realistic avatars.

The ultimate goal of the development of a low cost model-based coding system built on top of a VR system has been achieved.

8.2 Contributions of the Thesis

This dissertation details the development of a framework for model based coding and a sample implementation based on that framework with CoRgi. The result is a system that works using desktop workstations, is able to perform reconstructions using inexpensive CCD cameras, is able to perform image-based tracking in near real-time, and is able to determine pose and expressions exhibited by a subject. The pose is determined with two images, the first containing the subject in a neutral, upright position, and the second representing the subject in his/her current position. The general goals set in section 1.2 have thus been met. These are:

- The development of a system that is able to generate a realistic avatar of the subject. This has been achieved using only a CCD camera;
- The development of a framework for the implementation of a virtual videoconferencing system that works in real time at the expense of accuracy of facial modelling. Tracking 6 fiducial points, the system is able to achieve a frame rate of 10 fps with pose and expression classification. The system is able to identify two expressions, namely a smile and a frown;
- A sample implementation of the framework developed as a part of CoRgi.

A more specific breakdown of the achievements and contributions made by the research presented in this dissertation is provided below.

8.2.1 Compression

The achievement of excellent compression rates in real time is facilitated. Consequently, an orders of magnitude reduction in bandwidth requirements has been achieved using standard desktop workstations.

A transmission rate of 2 Kbps is achieved. This represents a 1000-fold reduction in the bandwidth usage of a traditional 100x100 uncompressed video stream. The transmission rate is linearly related to the performance of the NCC tracker. The bottleneck, traditionally associated with the network, is moved to the encoder.

8.2.2 Model Acquisition

The most significant contributions afforded by the reconstruction system developed are:

- The development of a cheap generic model acquisition system that is able to generate fully textured representations of any real-world object;
- A novel connection algorithm based very loosely on the marching cubes algorithm. It is able to generate completely closed 3D models;
- The introduction of a method to reduce NCC search space from a 2D problem down to a single evaluation;
- The use of NCC to perform concave surface reconstructions; this is facilitated by the space carving component's mapping process.

The most fundamental contribution offered by this thesis in terms of reconstruction is a method to speed up traditional stereoscopic NCC-based reconstructions.

The brute-force projection approach followed by the visual hull reconstruction system also promises to speed up the traditional NCC-based evaluation considerably because the space carving inherently generates matches that are conducive to NCC evaluations. The NCC-evaluation is reduced from a 2D problem down to a single evaluation, namely whether a specific colour match for a pixel falls above a specific threshold or not.

Additionally, an increase in the bounding volume resolution to sub-pixel levels further implies that sub-pixel depth recovery can be performed.

8.2.3 Encoding

The most important contribution presented in this chapter is the development of a near real-time NCC-based tracking system. The tracking mechanism is completely unobtrusive, avoiding the need to attach markers to the subject's face. A user-assisted feedback loop and history system is employed to overcome tracking problems due to occlusional discontinuities.

A further advantage of the NCC based tracker is that it is very robust. It has been shown that the tracking mechanism is invariant to scaling and in-plane rotations. This is primarily due to the dynamic nature of the feedback loop.

8.2.4 Decoding

A vertex-interpolation deformation system is successfully used for the purposes of expression generation. The system has complexity of magnitude *O(n)* and facilitates real-time deformations. A three-tier expression editing system that facilitates the composition of expressions has been designed and implementation.

Additionally, the following contributions have been made about the decoding process within the framework:

- An expression is defined as a collection of individual facial movements;
- Deformation filters are introduced to the basic VID implementation;
- To support the 3-tier expression editing hierarchy, an algorithm to combine more than one deformation for the purposes of expression generation is defined and implemented.

The proposed use of the VID system has been motivated by the observation that, taken in isolation, the deformations that underlie a specific expression result in structural change that is small with respect to the size of the avatar used. This is valid primarily when realistic facial expressions are modelled. As such, surface continuity and self-intersection issues can be avoided altogether.

It has been shown that the vertex interpolation deformation system is able to facilitate real-time expression generation, and, given its constraints, namely: that it does not observe surface continuity and self-intersection issues, is suited to the purpose of realistic facial expression generation. This is due to the observation that expressions consist of a collection of deformations that do not overly affect the avatar structure.

8.2.5 Expression Parameter Lookup Table

The primary benefit afforded by the concept of an EPLT is the independent mapping that is facilitated between the subject and avatar. This means that only an Expression ID is required to encode expression details.

Furthermore, it is extensible. In addition to supporting the universally identifiable expressions (subject-independent threshold rules for each), new, user specific expression entries can be added, and existing rules can be modified on a per-subject basis.

Finally, the subject is thus afforded freedom of representation in a conference, and therefore not constrained to a specific model.

8.2.6 Implementation with CoRgi

The CoRgi system has facilitated a high level implementation of the system design detailed in Chapter 2.

The following contributions have been made to the CoRgi system:

- Image-based NCC tracking;
- The addition of a three-tier deformation control mechanism to the system for expression generation and management;
- The creation of a CoRgi device, `VideoAvatarTracker`, that combines two very distinct areas of the CoRgi system, namely multimedia and VR, and thus fulfils two functions, namely: (a) tracking fiducials and extracting pose information from a live video stream, and (b) generating VR data as a device.

8.3 Future Work

There are many areas of the research presented here that could benefit from improvements. This section details some of these possible improvements and also introduces possible enhancements to this work through the integration of technology developed here with other systems.

8.3.1 Visual Hull-based NCC Evaluation

The use of brute force visual hull matching underlying an NCC evaluation is very promising. It could lead to the recovery of dense depth maps in a fraction of the time it currently takes.

8.3.2 Facial Tracking System

Currently, the NCC tracker implementation does not fare very well with a high degree of motion. The algorithm can be extended to adopt a more hierarchical approach to matching. This would improve tracking results with respect to the user moving around a lot in the scene. Occlusions are a source of concern as well. The current implementation does nothing to counter the possibility of occlusions.

8.3.3 Sound / Phoneme Extensions

The use of speech and phoneme analysis has been completely avoided during this discussion. The evaluation of speech to accurately model the shape of the mouth is one field that has been discussed by a few authors and a field that can benefit the deaf.

8.3.4 Automatic Face Segmentation and Location

A generic addition to the whole system would be a method whereby the face can be located automatically in the incoming video stream. This could also include automatically locating the positions of facial features such as the eyes and the nose. Face bunch graphs could be used to do this (Kruger *et al.* [35]).

8.3.5 MPEG-4 Compatibility

The generation of MPEG-4 compatible head coding is a very new direction of research and one that warrants pursuit.

8.3.6 Completely Immersive Virtual Conferencing

Casanueva [9] discusses the development of an upper-body tracking system utilising Inverse Kinematics to limit the number of trackers used. Used in combination with that work, the work presented here could lead to a cheap, fast and wholly immersive control mechanism for avatars in virtual environments.

8.3.7 Accurate Facial Modelling

In addition to the benefit of low-bandwidth networking, Ekman *et al.* [17] lists the following areas that could benefit from more accurate facial modelling include:

- Medicine: with very accurate facial modelling it is possible to determine certain medical disorders. Many disorders cause specific facial expressions, perceptions or interpretations of facial actions. By classifying the expression a specific primary disorder generates, it is possible to simplify diagnosis by evaluating a patient’s expression.

- Psychiatry: the use of the system developed above could assist in the treatment of victims of child-abuse. Communication with the victim could occur using an avatar representation they relate to, such as Mickey Mouse[4] in the case of child victims.

Accurate, real-time facial modelling thus has many benefits that could improve the living standards of many people. An important improvement to the system could be an improvement on the EPLT paradigm; instead of listing the deformation parameters associated with generating a specific expression, one could simply provide a mapping between each point that is tracked on the subject's face to a point on the avatar. As the point is tracked on the subject's face, so the associated position on the avatar could be deformed to match the position of the tracked point. This will lead to a more realistic and flexible expression generation system. With the NCC-based tracking system in place, this improvement looks very promising.

8.4 In Closing

The question posed in the first chapter has been answered. It has been shown that it is possible to perform replication facial expressions and head pose using standard desktop workstations and with minimal bandwidth requirements. The movement towards completely immersive virtual conferencing is not too far off.

[4] Mickey Mouse refers to a Walt Disney character. He is everybody's friend.

References

[1] Anandan, P. *A computation framework and an algorithm for the measurement of visual motion*. IJCV, 2:283-310, 1989.

[2] Ashley, P. *New 3D Scanner Zaps Sci-Fi Movie Set*. Professional Surveyor, 19(3), April 1999.

[3] Bangay, S., Gain, J., Watkins, G., Watkins, K. *RhoVeR: Building the Second Generation of Parallel/Distributed Virtual Reality Systems*, First Eurographics Workshop on Parallel Graphics & Visualisation, Bristol(UK), September 1996. Available at http://www.cs.ru.ac.za/vrsig/techdocs/PGV.ps.gz

[4] Basu, S., Essa, I., Pentland, A. *Motion Regularisation for Model-based Head Tracking*. 13th Int'l Conference on Pattern Recognition (ICPR '96), Vienna, Austria, August 1996.

[5] Beauchemin, S and Barron, J. *The Computation of Optical Flow*, Computer Science Department, University of Western Ontario, London, Ontario, Canada, 1995.

[6] Birchfield, S. *An Elliptical Head Tracker*. Computer Science Department. Stanford University. Proc: 31st Asilomar Conference on Signals, Systems and Computers, November 1997.

[7] Black, M. and Yacoob, Y. *Tracking and Recognizing Facial Expressiona in Image Sequences, using Local Parameterized Models of Image Motion*. Technical Report CAR-TR-756, Computer Vision Laboratory, Center for Automation Research, University of Maryland, January 1995.

[8] Burford, D. and Blake, E. *Real-Time Facial Animation for Avatars in Collaborative Virtual Environments*, Collaborative Visual Computing Laboratory,

University of Cape Town. Proc: South African Telecommunications Networks Applications Conference, Cape Town, September 1999.

[9] Casanueva, L. *Minimal Motion Capture with Inverse Kinematics for Articulated Human Figure Animation*, MSc Thesis, Computer Science Department, Rhodes University, February 1999.

[10] Carceroni, R. and Brown, C. *Numerical Methods for Model-Based Pose Recovery*. Technical Report 659, University of Rochester, August 1997.

[11] DeCarlo, D., Metaxas, D., Stone, M. *An Anthropometric Face Model Using Variational Techniques*. Proc: Computer Graphics Proceedings, SIGGRAPH 98, 67- 73 (1999).

[12] Deriche, R., Faugeras, O., Luong, Q., Zhang, Z. *A Robust Technique for Matching Two Uncalibrated Images Through the Recovery of the Unknown Epipolar Geometry*. Unite de recherche INRIA, Sophia-Antipolis, Report 2273, May 1994. Available at ftp://ftp.inria.fr

[13] Du, B. and Maeder, A. *Approaches to Video Transmission over GSM Networks*. Appears: Proceedings of South African Institute of Computer Scientists and Information Technologists, November 1999.

[14] Easton, V. and McColl, J. *Statistical Glossary*, 1999. Available at http://www.stats.gla.ac.uk/steps/glossary/paired_data.html

[15] Eisert, P. and Girod, B. *Analyzing Facial Expressions for Virtual Conferencing*. University of Erlangen Proc: IEEE Computer Graphics and Applications 18(5), 70 – 78 (October 1998).

[16] Ekman, P. and Friesen, W. *Unmasking the Face*. Prentice-Hall, 1975.

[17] Ekman, P., Huang, T., Sejnowski, T., Hager, J. *Final Report To NSF of the Planning Workshop on Facial Expression Understanding*. July-August 1992. Available at ftp://helmholtz.sdsc.edu

[18] Eppstein, D. *Quadtree and Hierarchical Space Decomposition*, 1999. Available at http://www.ics.uci.edu/~eppstein/gina/quadtree.html

[19] Escher, M. and Thalmann, N. *Automatic 3D Cloning and Real-Time Animation of a Human Face*. MIRALab, University of Geneva, 1997. Available at http://miralabwww.unige.ch/ARTICLES/FACA97.html

[20] Escher, M., Goto, T., Kshirsagar, S., Zanardi, C., Thalmann, N. *User Interactive MPEG-4 Compatible Facial Animation System*. MIRALab/CUI, University of Geneva, Switzerland, 1999. Available at http://www.stats.gla.ac.uk/steps/glossary/paired_data.html

[21] Essa, I. *Analysis, Interpretation and Synthesis of Facial Expressions.* Doctoral Thesis, Massachusetts Institute of Technology, February 1995.

[22] Essa, I. and Pentland, A. *Facial Expression Recognition using a Dynamic Model and Motion Energy*. Technical Report 307. International Conference on Computer Vision ’95, Cambridge, MA, June 1995.

[23] Ezzat, T. and Poggio, T. *Facial Analysis and Synthesis Using Image-Based Models*. Proc: , *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, Killington, Vermont, October 1996. Available at http://cuneus.ai.mit.edu:8000/publications/icafagr96.pdf

[24] Faugeras, O., Hotz, B., Mathieu, H., Vieville, T., Zhang, Z., Fua, P., Theron, E., Moll, L., Berry, G., Vuillemin, J., Bertin, P., Proy, C. *Real time correlation-based stereo: algorithm, implementations and applications*. Technical Report 2013, Project Robotvis, Inria Research Institute, 1993. Available at ftp://ftp.inria.fr

[25] Fourie, F. *Developing Efficient Graphics Rendering Components*. Honours Thesis, Computer Science Department, Rhodes University, November 1998. Available at http://www.cs.ru.ac.za/vrsig/techdocs/fourie/thesis.ps.gz

[26] Fua, P. and Miccio, C. *Animating Heads from Ordinary Images: A Least Squares Approach*. Computer Graphics Lab (LIG) EPFL, CH-1015. Proc: Optical 3D Measurement Techniques, Zurich, Switzerland 1997.

[27] Gain, J. *Virtual Sculpting: An Investigation of Directly Manipulated Free-Form Deformation in a Virtual Environment.* MSc Thesis, Department of Computer Science, Rhodes University, February 1996, Available at http://www.cs.ru.ac.za/vrsig/techdocs/gain/virtual_sculpting.ps.gz

[28] Goto, T., Escher, M., Zanardi, C., Thalmann, N. *MPEG-4 based animation with face feature tracking*. MIRALab, University of Geneva, 1999. Available at http://www.miralab.unige.ch

[29] Hong, T., Thalmann, N., Thalmann, D. *A General Algorithm For 3-D Shape Interpolation In A Facet-Based Representation.* MIRALab, HEC / IRO Université de Montréal, Canada, 1988.

[30] Horprasert, T., Yacoob, Y., Davis, L. *Computing 3D Head Orientation from a Monocular Image Sequence.* Computer Vision Laboratory, University of Maryland, College Park, 1995.

[31] Hung, D. and Huang, S. *Project Deidre (I) – Modeling Human Facial Expressions*, Cornell Theory Centre, 1995. Available at http://www.tc.cornell.edu/Visualization/contrib/cs490-95to96/hjkim/deidre.html

[32] Kalra, P., Mangili, A., Thalmann, N., Thalmann, D. *Simulation of Facial Muscle Actions Based on Rational Free-Form Deformations*. Computer Graphics, September 1992. Available at http://www.tc.cornell.edu/Visualization/contrib/cs490-95to96/hjkim/kalra92.html

[33] Killian, B. *Providing Efficient Networking for Distributed Virtual Reality Using CoRgi*. Honours Thesis, Computer Science Department, Rhodes University, November 1998.

[34] Koenen, R. *MPEG-4 Overview - (Seoul Version) - Overview of the MPEG-4 Standard*. March 1999. Available at www: http://drogo.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm

[35] Kruger, N. and Potzsch M. *Bunch Graph. Systems Biophysics Research*. Institut für Neuroinformatik (INI), Ruhr-Universität Bochum, Germany, December 1997. Available at http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/computerVision/representation/bunchGraphs/contents.html

[36] Kruger, N. *Representation of Models by Labeled Graphs*. Systems Biophysics Research, Institut für Neuroinformatik (INI), Ruhr-Universität Bochum, Germany, December 1997. Available at http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/computerVision/representation/contents.html

[37] Kutulakos, K. and Seitz, S. *What Do N Photographs Tell Us About 3D Shape?* Technical Report TR680. Rochester University, January 1998. Available at http://www.cs.cmu.edu/~seitz/papers/tr680.pdf

[38] Lacroute, P. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. Ph.D. dissertation, Technical Report CSL-TR-95-678, Stanford University, 1995.

[39] Lee, W., Escher, M., Sannier, G., Thalmann, N. *MPEG-4 Compatible Faces from Orthogonal Photos*. Proc. CA99 (International Conference on Computer Animation), Geneva, Switzerland, 186-194 (May 1999). Available at http://miralabwww.unige.ch/ARTICLES/lee.pdf

[40] Lengagne, R., Tarel, J., Monga, O. *From 2D Images to 3D Face Geometry*. Proc: IEEE Second International Conference on Automatic Face and Gesture Recognition (FG' 96), Killington, USA, October 1996. Available at

http://www-syntim.inria.fr/syntim/textes/fg96-eng.html

[41] Lewis, J. *Automated Lin-Sync: Background and Techniques*. Computer Graphics Laboratory, New York Institute of Technology, 1991. Available at http://www.idion.com/~zilla/Papers/lipsync91/lipsync91.html

[42] Lewis, J. P. *Fast Normalized Cross-Correlation*. Industrial Light & Magic. Available at http://www.yeeyoga.com/~zilla/Papers/nvisionInterface/nip.html

[43] Maurer, T. and Von der Malsburg, C. Tracking and Learning Graphs and Pose on Image Sequences of Faces. Proc: 2nd International Conference on Automatic Face- and Gesture-Recognition, Killington, Vermont, USA, 176-181 (October 1996).

[44] Moghaddam, B. and Pentland, A. *An Automatic System for Model-Based Coding of Faces*. Technical Report No. 317, MIT Media Laboratory Perceptual Computing Section. Proc: IEEE Data Compression Conference, Snowbird, Utah, March 1995.

[45] Niem, W. and Buschmann, R. *Automatic Modelling of 3D Natural Objects from Multiple Views*. Inistitut fur Theoretische Nachrichtentechnik und Informationsverarbeitung. Universitat Hannover, Hannover, Germany, 1994. Available at ftp://ftp.tnt.uni-hannover.de/pub/papers/1994/MONALISA94-WN.ps

[46] Okutomi, M. and Kanade, T. *A Locally Adaptive Window for Signal Matching*. Proc: ICVV, 190-199 (1987).

[47] Oliver, N., Pentland, A., Berard, F. *LAFTER: Lips and Face Real Time Tracker with Facial Expression Recognition*. Technical Report No. 396, MIT Media Laboratory. Proc: Computer Vision and Pattern Recognition Conference, CVPR ’97, 1997.

[48] Panagou, S. and Bangay, S. *The Development of a Generic Framework for the Implementation of a Cheap, Component-Based Virtual Video-Conferencing*

System. Proc: SACJ (South African Computer Journal) Special Issue: SAICSIT-99 (South African Institute of Computer Scientists and Information Technologists) 24, 185-193 (November 1999). Available at http://www.cs.ru.ac.za/vrsig/techdocs.html

[49] Parke, F. *Parameterized Models for Facial Animation*. New York Institute of Technology. Proc: IEEE Computer Graphics and Applications 2(9), 61 – 68 (November 1982).

[50] Pelachaud, C., Badler, N., Steedman, M. *Generating Facial Expressions for Speech*. Department of Computer Science and Information Science, University of Pennsylvania, Philadelphia, 1995.

[51] Pelachaud, C., Badler, N., Steedman, M. *Linguistic Issues in Facial Animation*. Department of Computer Science and Information Science, University of Pennsylvania, Philadelphia, 1991.

[52] Preston, L. *The Use of Virtual Reality in the Reduction of Stress*. Honours Thesis, Computer Science Department, Rhodes University, November 1998.

[53] POV-Team. *POV-Ray(tm) User's Documentation 3.0*. 1997. Available at http://mirror.det.mun.ca/doc/povray-manual/povray.htm

[54] Rekimoto, J. *A Vision-Based Head Tracker for Fish Tank Virtual Reality – VR without Head Gear*. Technical Report SCSL-TR-95-004, Sony Computer Science Laboratory Inc. Tokyo, Japan, February 1995.

[55] Roach, L. and Jacobs, G. *3D Laser Scanning Speeds 3D Visualization of Existing Plants and Extends 3D Visualization into New CPI Applications*. Proc: CE Exposition & Conference, Houston, June 1999.

[56] Rummel, R. *Understanding Correlation*. Honolulu, Department of Political Science, University of Hawaii, 1976. Available at http://www2.hawaii.edu/~rummel/UC.HTM

[57] Schroeder, M., Martin, K., Lorensen, B. *The Visualisation Toolkit: An Object Oriented Approach to 3D Graphics*. 146-152, Prentice Hall, 1996.

[58] Seitz, S., and Dyer, C. *Photorealistic Scene Reconstruction by Voxel Coloring*. Proc: Computer Vision and Pattern Recognition Conf., 1067-1073 (1997). Available at http://www.cs.cmu.edu/~seitz/vcolor.html

[59] Singh, A. *Optic Flow Computation: A Unified Perspective*. IEEE Computer Science Society Press, 1992.

[60] Sobottka, K. and Pitas, I. *A Novel Method for Automatic Face Segmentation, Facial Feature Extraction and Tracking*. Department of Informatics, University of Thessaloniki, Greece. Proc: First International Conference on Audio- and Video-based Biometric Person Authentication (AVBPA'97), Crans-Montana, Switzerland, 77-84 (March 1997),. Available at http://poseidon.csd.auth.gr:80/papers/PUBLISHED/CONFERENCE/Sobottka97a/Sobottka97a.ps.Z

[61] Terzopoulos, D. and Szeliski, R. *Tracking with Kalman Snakes*. Appears: Rakn Prize Funds Mini-Symposium on Active Vision, Grasmere, England, September 1991. Available at http://www.stanford.edu/~jeremyj/cs328/ksnakes.html

[62] Thalmann, N., Cazedevals, A., Thalmann, D. *Modelling Facial Communication Between an Animator and a Synthetic Actor in Real Time*. MIRALab, CUI, University of Geneva, Switzerland, 1999.

[63] Thalmann, N., Kalra, P., Escher, M. *Face to Virtual Face*. MIRALab, CUI, University of Geneva, Switzerland, 1998. Available at www: http://Matrix Software, 1998. Available at http://www.miralab.unige.ch/ARTICLES/ieeet.html

[64] Thalmann, N. and Thalmann, D. *Synthetic Actors in Computer Generated 3D Films*. Springer-Verlag, 1989.

[65] Unknown, *Film Gloss: A Complete Film and Movie Glossary – Blue Screen Processing*. Matrix Software, 1998. Available at http://www.allmovie.com/mov_Glossary.html

[66] Van den Bergh, F. and Lalioti, V. *Software Chroma-Keying in an Immersive Virtual Environment*. University of Pretoria. Proc: SACJ (South African Computer Journal) Special Issue: SAICSIT-99 (South African Institute of Computer Scientists and Information Technologists), 24, 155-162 (November 1999).

[67] Vishnevsky, E. *Color Conversion Algorithms*, 1999. Available at http://www.cs.rit.edu/~ncs/color/t_convert.html.

[68] Woo, M., Neider, J., Davis, T. *OpenGL Programming Guide*. Second Edition : The Official Guide to Learning OpenGL, version 1.1 Second Edition, Addison Wesley Developer's Press, 1997.

[69] Zhang, Z. *A Flexible New Technique for Camera Calibration*. Technical Report MSR-TR-98-71, Microsoft Research, March 1999. Available at http://research.microsoft.com/~zhang/Papers/TR98-71.pdf

[70] Zhang, Z. *Determining the Epipolar Geometry and its Uncertainty: A Review*. Unite de recherche INRIA, Sophia-Antipolis, Report 2927, July 1996. Available at ftp://ftp.inria.fr/

Appendix A - Tracker Parameter Results

This appendix is intended to tabulate the measurements obtained with the NCC tracker over a twenty second time period. The groupings are with respect to the **Search Window Parameter**. Within each table, a sub-grouping according to the frame size has been provided, with average measurements provided at the end of each frame size grouping (average frames per second [average total frames processed]). All measurements have been performed using a 333 MHz Intel Pentium II, running RedHat Linux 6.1 with 128 Mb ram, and unless otherwise specified, have been taken over a 20 second time period.

Number of Corrections is indicative of how robust the tracker is with a given set of parameters. "-" indicates that the tracker was too slow with the given parameters. Measurements were thus avoided.

Search Window Parameter = 64

	Frame Size	Correlation Window Size	Frames Per Second	Number of Corrections
	256x256	6x6	16.10	25
	256x256	10x10	8.85	9
	256x256	14x14	5.35	5
	256x256	18x18	3.50	9
	256x256	21x21	2.45	4
Average			**7.25 [145.0]**	
	192x192	6x6	30.95	30
	192x192	10x10	17.10	24
	192x192	14x14	9.95	13
	192x192	18x18	6.40	8
	192x192	21x21	4.85	12
Average			**13.85 [277.0]**	
	128x128	6x6	97.30	50
	128x128	10x10	41.05	40
	128x128	14x14	22.65	43
	128x128	18x18	14.90	27
	128x128	21x21	10.20	4
Average			**37.22 [744.4]**	

Search Window Parameter = 32

	Frame Size	Correlation Window Size	Frames Per Second	Number of Corrections
	256x256	6x6	6.35	8
	256x256	10x10	2.85	1
	256x256	14x14	1.50	0
	256x256	18x18	3.50	0
	256x256	21x21	2.45	0
Average			**3.33 [66.6]**	
	192x192	6x6	11.95	14
	192x192	10x10	5.05	9
	192x192	14x14	2.75	2
	192x192	18x18	1.70	5
	192x192	21x21	1.15	1
Average			**4.52 [90.4]**	
	128x128	6x6	27.95	34
	128x128	10x10	11.55	22
	128x128	14x14	6.30	16
	128x128	18x18	3.90	8
	128x128	21x21	2.75	-
Average			**10.49 [209.8]**	

Search Window Parameter = 28

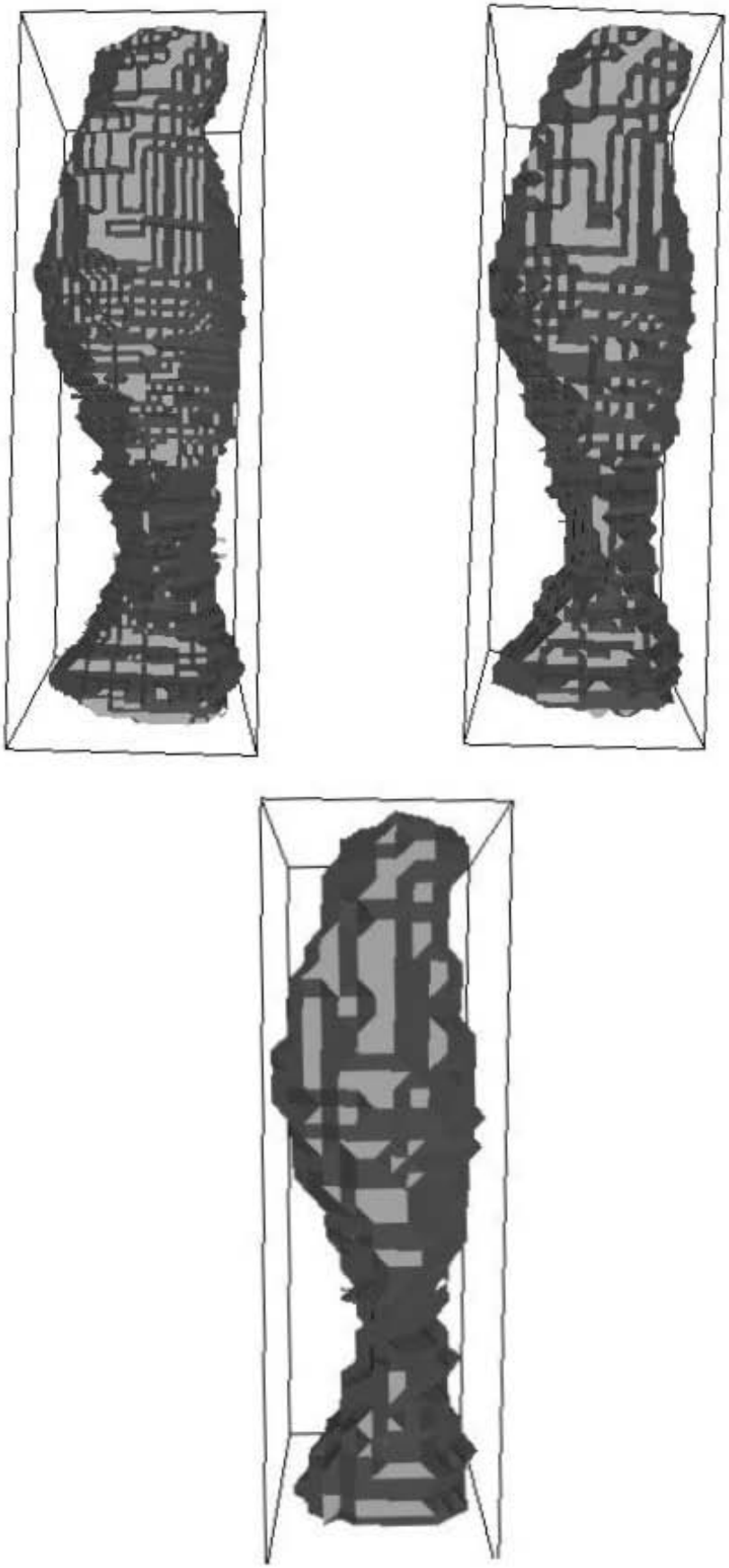
	Frame Size	Correlation Window Size	Frames Per Second	Number of Corrections
	256x256	6x6	5.30	5
	256x256	10x10	2.30	0
	256x256	14x14	1.25	2
	256x256	18x18	3.50	3
	256x256	21x21	2.45	3
Average			**2.96 [59.2]**	
	192x192	6x6	10.20	21
	192x192	10x10	4.55	11
	192x192	14x14	2.45	3
	192x192	18x18	11.55	6
	192x192	21x21	1.05	2
Average			**3.96 [79.2]**	
	128x128	6x6	24.85	34
	128x128	10x10	9.95	22
	128x128	14x14	5.25	16
	128x128	18x18	3.25	8
	128x128	21x21	2.20	-
Average			**9.1 [45.5]**	

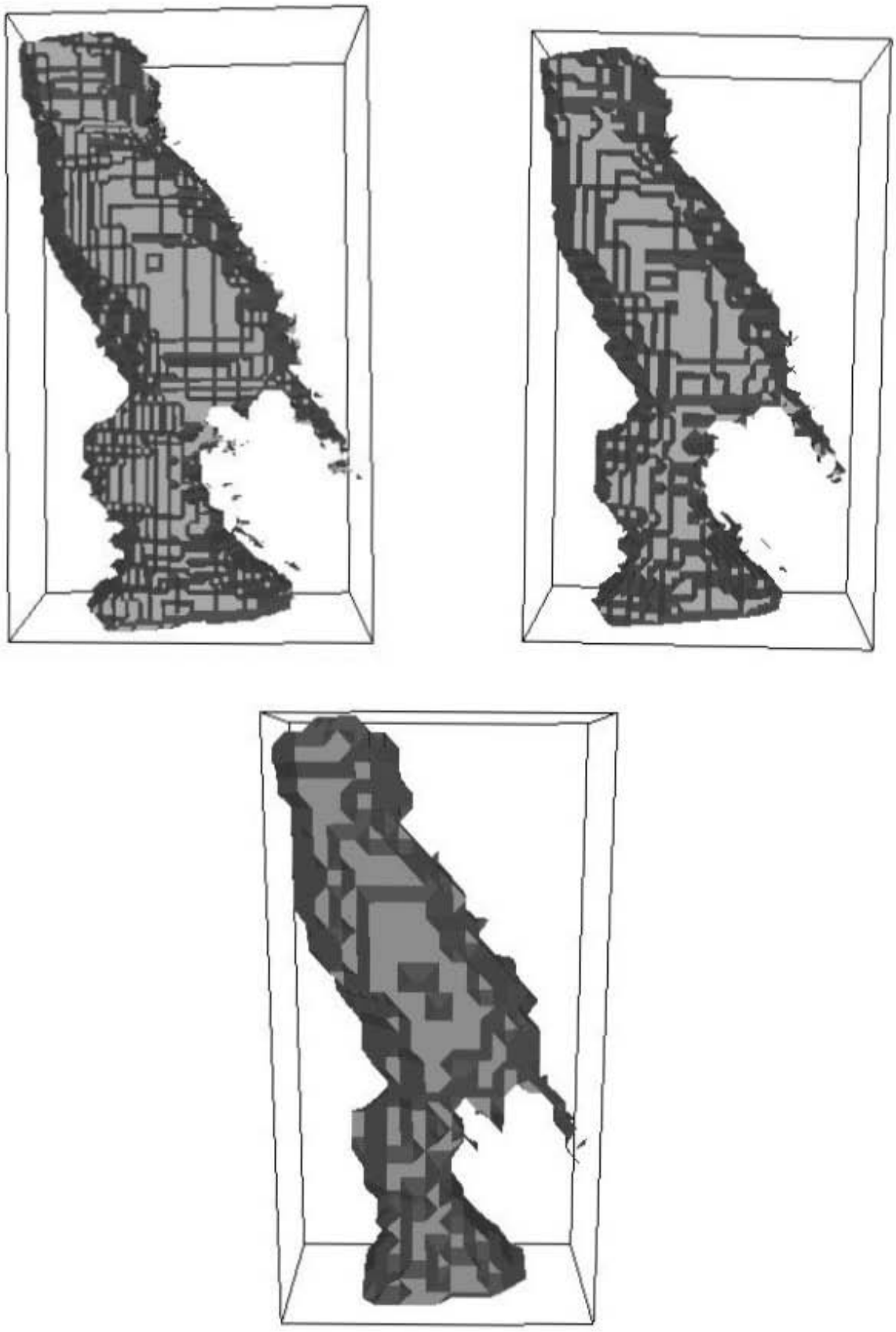
Search Window Parameter = 24

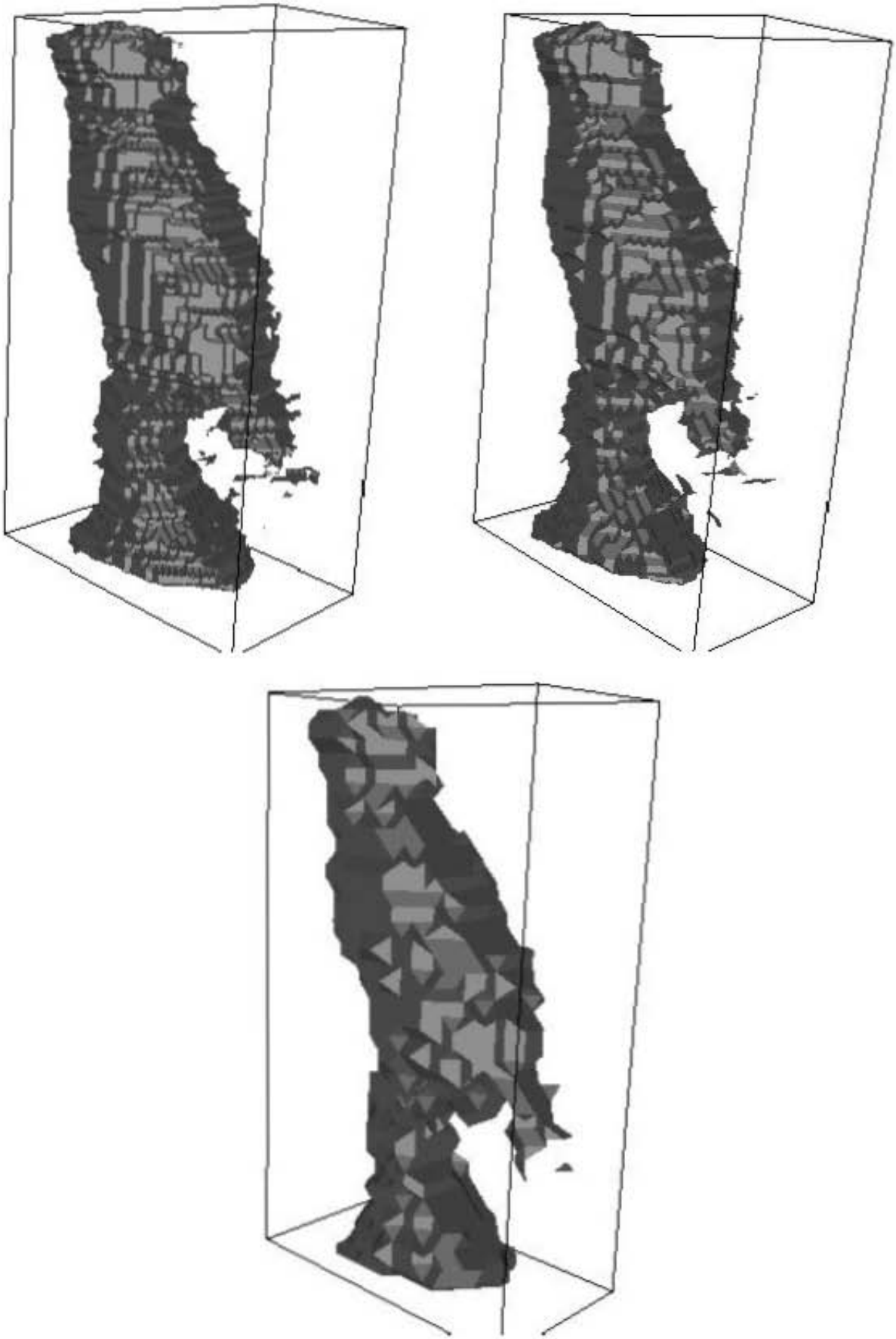
	Frame Size	Correlation Window Size	Frames Per Second	Number of Corrections
	256x256	6x6	4.10	12
	256x256	10x10	1.70	0
	256x256	14x14	0.95	0
	256x256	18x18	0.75	3
	256x256	21x21	0.40	3
Average			**1.58 [31.6]**	
	192x192	6x6	7.70	-
	192x192	10x10	3.10	-
	192x192	14x14	1.65	-
	192x192	18x18	1.05	-
	192x192	21x21	0.70	-
Average			**2.84 [56.8]**	
	128x128	6x6	20.15	-
	128x128	10x10	8.20	-
	128x128	14x14	4.20	-
	128x128	18x18	2.60	-
	128x128	21x21	1.80	-
Average			**7.39 [147.8]**	

Appendix B - Multi-resolution Budgie Reconstruction

This section illustrates multi-resolution reconstructions of a budgie without texturing. The first image in each triplet below represents a 150x150x150 reconstruction, followed by a 100x100x100 reconstruction, and then finally a 50x50x50 reconstruction.







Appendix C - Expression Editing

The development of an expression editing system is discussed in this section. It is based on a three-tier expression editing hierarchy.

C.1 Description

The goal of this application is to simplify the development of an FACS-like database of expressions. The application allows the user to deform a custom 3D model and build up an expression by combining a number of basic deformations, which can then be saved to disk for later expression replication. Once a list of appropriate deformations have been selected that will generate a specific expression, the parameters can be saved for later use. **Figure C.1** illustrates the main user interface of the application. In addition to the deformation control panel, the Viewer Control panel allows the object to be rotated and zoomed in upon using the radius slider. It can conceptually be decomposed into three levels of functionality, namely:

- Single deformation management;
- Multiple deformation management; and lastly
- Expression management.

Each of these three levels of functionality gradually builds on top of each others, with the single deformation management system acting as the building block for the remaining levels.

C.2 Single Deformation Management

The basic functionality provided is the ability to control single deformations. This implies that the user can deform the model using either a global deformation or a deformation filter.

The currently loaded deformation filters are displayed in the `Deformation Filters` list box. The selection of any deformation filter from the list box means that the deformation parameters associated with that filter are set as the **active parameters** for the current deformation. The active parameters refer to the data present in each of the input boxes on the Main Panel. Thus, selecting any deformation filter means that the currently active parameters are overwritten by the parameters associated with the newly selected filter.

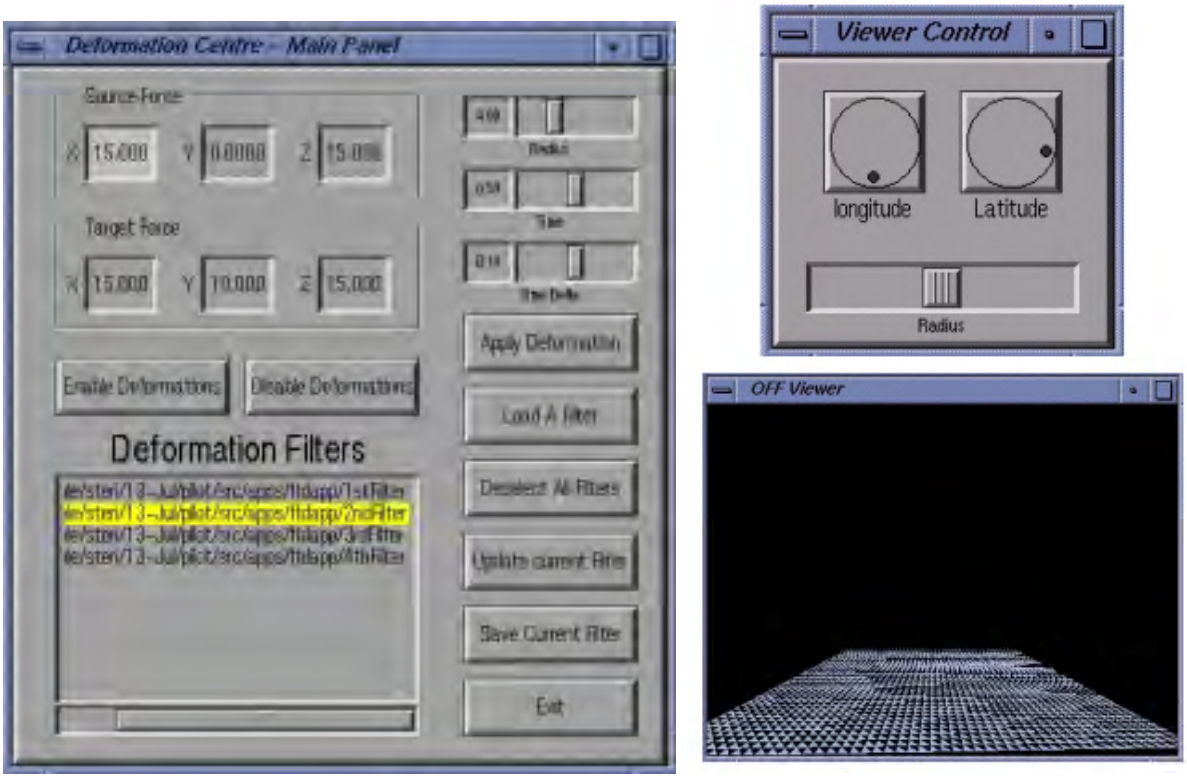


Figure C.1 The expression editor's main interface.

Any of the deformation parameters may be altered individually. It may be desirable that the animation occur quickly. The `TimeDelta` parameter, or even the `EndTime` parameter can be altered appropriately. To allow the user easily and quickly to specify the appropriate forces relative to the custom model, two cubes are positioned around the object. These cubes represent the source and target force parameters respectively and are coloured differently (green for the source and red for the target). As the positions of the source and target forces are changed, so the cubes move to their new respective locations. Once the deformation parameters have been set, the user can view the object being deformed. This process is repeated until the correct deformation parameters have been chosen.

The `OFF Viewer` in **Figure C.1** is the rendering window displaying the object being deformed, while the Viewer Panel facilitates rudimentary rotation of the object being deformed.

C.3 Multiple Deformation Management

Multiple deformations are composed of a number of single deformations (either global deformations or deformation filters) that are executed in a specific sequence, thereby affecting

Figure C.2 The Animation Panel.

the object topology in a number of interesting ways. The creation/management of multiple deformations maps very nicely to the video-editing paradigm. The implementation of a system along these lines involves the addition/deletion of individual deformation filters/global deformations to/from a playlist.

Each playlist represents an expression. The interface panel for an implementation based on the idea of expression playlist editing is illustrated in **Figure C.2**. This illustrates the part of the final expression editing application that facilitates control over multiple deformations and, ultimately, expressions. The `Animation Panel` allows deformation entries to be added and removed from the `Animation Queue`. The playback controls at the bottom of the panel (`Play`, `Stop`) provide control over the animation process. In this example, there are currently two filters (`2ndFilter` and `3rdFilter`) in the `Animation Queue`.

The playlist is composed of an ordered list of deformation filter names (`GLOBAL` for global deformations and `<filter name>` for deformation filters) and the deformation parameters for each. As entries are added to the playlist, the name uniquely identifying the parameter and the deformation parameters for that entry are added to the playlist. In addition to the normal parameters, however, the `time delta` parameter is included as well.

An expression is generated by combining a number of deformation filters. A fine-tuning process is applied to each entry in the playlist. This means that the deformation parameters for each entry can be modified until the best results are returned for the overall expression are achieved.

Fully supporting the video-editing paradigm implies the inclusion of playback controls (`Play` and `Stop`). These controls facilitate the fine-tuning process; the effects of playlist on the object can be visualised in this way.

Each playlist entry contains the following information:

- Name of the deformation entry (`GLOBAL` for a global deformation or a filter name for a deformation filter);
- 3D position of the source force; (`SourceForce`)
- 3D position of the target force; (`TargetForce`)
- Radius of influence; (`Radius`)
- End time; (`EndTime`); specifies the time at which the deformation will stop.Is represented as a value between 0 and 1.
- Time delta; (`TimeDelta`)
- Time direction (`Direction`); specifies whether the deformation is proceeding from start to end or the other way round;
- Current time;

The inclusion of a time direction value specifies whether `TimeDelta` is a positive or negative component to be added to or subtracted from the `CurrTime`. `Direction` can be either `–1` or `1` depending on whether the deformation is getting bigger or smaller. Initially, `CurrTime` represents the start time, typically 0.

C.3 Avatar Expression Management

The final issue to be dealt with in terms of the expression editing system is that of avatar expression management. This translates to a tool that allows the current list of defined expressions to be evaluated and tested. An example is illustrated in **Figure C.3**. As an expression is selected from the drop-down **list of expressions** so the avatar is deformed to the selected expression. The selection of `smile.exp` thus causes the avatar to assume a smile.

Figure C.3 The Expression Panel.

The previous section has described the composition of expressions from individual expressions. The functionality described here ties together all expressions associated with an avatar. The result is that all expressions belonging to a specific avatar are grouped together and can be tested at a high level.

A list of all expressions belonging to an avatar is provided. The selection of any expression from the list forces the generation of the expression according to the deformations defined for the selection. Traversing the list of expressions by selecting each one and viewing the deformation effects on the model facilitates a quick classification of the correctness of the expression generation. **Figure C.3** also illustrates the resulting smiling face that is generated when `smile.exp` is selected.

At an implementation level, the process of expression management is important because it means that a high-level scripting system is required to handle the selection of the different expressions from the drop-down list. If the user selects a frown from the list, all the internal deformation parameters must be changed to allow the correct deformations to take place. A result of this is a high level of control over the expression generation process, and the provision of facilities for later expression management in terms of the higher level decoding process.

Appendix D - CDROM

The accompanying CD-ROM contains, amongst other things, the source tree for the model acquisition system, as well as a snapshot of the CoRgi source tree. See the `README` file in the root directory of the CD-ROM for more details about the contents.