Three-Dimensional Computer Metaporphosis Through Energy Minimisation

Korfiatis, Ioannis

QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

Department of Computer Science

Research Report No. RR-00-09 ISSN 1470-5559 October 2000

Three-Dimensional Computer Metamorphosis Through Energy Minimisation

Ioannis Korfiatis

THREE-DIMENSIONAL COMPUTER METAMORPHOSIS THROUGH ENERGY MINIMISATION

Ioannis Korfiatis

A Thesis submitted in fulfilment of the requirements
of the degree of doctor of Philosophy
in the University of London

Department of Computer Science
Queen Mary and Westfield College
University of London

2000

Abstract

This dissertation presents a methodology for the metamorphosis of polygonal shapes, using a physically based approach.

A common approach in shape metamorphosis assumes a pair of objects represented as a set of polygons. The vertices of the first object are then displaced over time, to coincide in position with the corresponding vertices of the second object. There are two steps in this approach: first establishing a desirable vertex correspondence and then interpolating the co-ordinates of the corresponding vertices in order to create the intermediate objects.

Correspondence determination is a somewhat ambiguous process, related more to subjective human judgements rather than well-understood, universally applicable principals. The task of correspondence determination has two levels, the syntactic and the semantic levels. In the syntactic level shapes are treated as geometric entities and the metamorphosis is concerned only with geometric properties of elements such as sizes or distances while in the semantic level correspondence is established on the basis of similarity in parts of the two objects.

This dissertation presents a method proposed to solve the correspondence determination problem in the semantic level. This approach assumes that two features of the objects are assumed similar if one needs small changes to transform one to the other. The measure to quantify these "small" or "big" changes is the energy needed to perform them assuming that the objects are made of an elastic material.

Acknowledgements

I would like first to acknowledge the support and guidance of my supervisor Prof. Yakup Paker during the course of this research work.

I am indebted to my friends Dr. Ali V. Sahiner for his helpful criticisms, Turgay Altilar and Tasos Hamosfakidis for their help and support.

This work was supported by a scholarship of the Greek Institute of scholarships (I.K.Y) for which I am grateful. I am also grateful to my parents for their continual backing as well as their financial support.

Dissertation Overview

Chapter 1 presents the background of this research. The applications of computer assisted metamorphosis are discussed and an overview of methods that have been developed is presented.

Chapter 2 discusses a physically-based method used for the blending of two-dimensional polygonal contours. This algorithm will be the basis of the methods described in the following chapters for the metamorphosis of 2D and 3D polygonal shapes.

Chapter 3 presents a method for the morphing between simple two-dimensional polygonal shapes. These shapes consist of a set of domains that can be geometrically ordered on a plane. The method proposed reduces the problem of correspondence determination for these domains to the shortest path problem in a directed graph.

Chapter 4 extends the method described in the previous chapter to the metamorphosis of general two-dimensional polygonal objects. In order to apply the method presented in the third chapter, a new representation of the two-dimensional shapes is considered, this new representation preserves the same geometric information but assigns the shapes a new connectivity structure.

Chapter 5 presents an extension of the 2D method developed to transform a three-dimensional polygonal object to another three-dimensional polygonal object.

Chapter 6 deals with complexity related with the 3D metamorphosis method that is described in chapter 5. There are also some performance measurements presented. Finally at the end of the chapter there is a simplified method presented attempting to solve the vertex correspondence determination problem in real time.

Chapter 7 discusses issues related to the parallelisation of Sederberg's and Greenwood's algorithm and the 3D metamorphosis method.

Finally Chapter 8 concludes the dissertation with reflections on the research outcome and implications for the future.

Table of Contents

List of Figures

List of Tables

Chapter 1

Shape Metamorphosis Techniques

1.1 Introduction

Image and object morphing techniques have gained increasing importance in the last few years having applications in a variety of fields: special effects, animation, design, education, visualisation etc. Given two objects metamorphosis involves producing a sequence of intermediate objects that gradually evolve from one object to another.

Morphing, whether in two or three dimensions, generally consists of two basic phases, the first establishes correspondence between the elements of the two objects and the second interpolates the positions of the corresponding elements in order to create the intermediate objects.

The correspondence determination should be established in such a way as that the resulting animation appears 'natural' and 'pleasing' to the viewer. There is a variety of ideas of what the sequence of the inbetween objects should satisfy to appear natural. However it is rather easy to agree that a morph sequence should satisfy the following criteria to yield a pleasing morph.

Connected and not distorted intermediate shapes: It is evident that algorithms that result into disconnected or heavily distorted intermediate shapes are of little practical use.

No self-intersection occurs during the metamorphosis.

Application to general objects: Most objects both in nature and in computer graphics do not fall in a special category. Even so due to the difficulty of the metamorphosis problem most existing methods put some restrictions to the objects they can handle. The less restrictive is a method to the characteristics of the input objects the more successful it can be considered.

Interaction and control from the user: The main reason to apply computer metamorphosis techniques is to relieve the user of some tedious processes related with animation. It is clear that it is not desirable for the user to specify how every single component of the object should transform during the metamorphosis sequence. However the user is the ultimate judge of the quality of the final result and therefore he/she must be able to guide the metamorphosis by providing as limited input information as possible. Since the source and the target objects can be significantly different and/or misaligned, some user specification is unavoidable.

Establishing correspondences on the semantic level: An important target of algorithms establishing correspondence is finding similar features on the two objects correspond them together and thus preserve them during the animation.

The objective of this thesis is to present a method that solves the correspondence determination problem in the semantic level, which is something which has not been fully addressed by existing methods for correspondence determination. At the same time this method should satisfy the rest of the criteria stated. The method will be applicable to 3D and 2D polygonal objects.

The basic idea that we propose is using a physically based approach. The objects are considered to be created by an elastic material. We assign energy needed to bend and stretch features of the first object to corresponding features of the second. Similar features will need a small amount of energy. Therefore the correspondence of the shapes that minimises the total amount of energy needed (to transform the first object to the second) is the solution to the correspondence determination problem.

In this chapter we will mainly concentrate on metamorphosis techniques that solve the correspondence determination problem and particularly on ones that manage the transformation of polygonal objects but will also present a background of the main approaches used for solving the interpolation problem.

1.2 Metamorphosis techniques

1.2.1 Conventional Metamorphosis Techniques

Metamorphosis between two or more images over time is a useful visual technique often used for entertainment or educational purposes. Traditional filmmaking techniques for this effect include clever cuts (such as a character changing while running through a forest and passing behind several trees) and optical cross-dissolve in which an image is faded out while another is the same time faded in (with makeup change, appliances, or object substitution). Taking the cutting approach to the limit gives us the technique of stop-motion animation in which the subject is progressively transformed and photographed one frame at a time. This process can give the powerful illusion of continuous metamorphosis but requires much skill and it is tedious work. Moreover, stop-motion usually suffers from the problem of visual strobing by not providing the motion blur normally associated with moving film subjects. A motion-controlled variant, called go-motion, in which the frame-by-frame subjects are photographed while moving can provide the proper motion blur to create a more natural effect, but the complexity of the model's motion hardware and the required skills become even greater. Lately in a lot of cases we have a computer helping in these tedious processes with very impressive results.

Computer graphics techniques can be split into two main areas: 2D and 3D morphing. 2D techniques have given more impressive results but they have the disadvantage of not keeping models of objects so a mapping between a part of the image that is not visible in the first frame and visible in the last is not possible. We consider these two approaches more analytically later in this chapter.

1.3 A classification of Computer Metamorphosis Techniques

Computer metamorphosis techniques can be categorised by their method for determining correspondences and their method for assigning trajectories and transformations. In this section we concentrate on methods used to define correspondences, we will briefly discuss methods that are used for interpolation at the end of this chapter.

We present a classification of the computer metamorphosis methods according to a number of different criteria. These criteria are summarised in Table 1.1 and then each topic is developed in more detail.

Table 1.1:Classification of Computer Metamorphosis

1) According to the application
production of commercial or entertainment films
industrial or scientific simulation
2) According to the graphical objects that are transformed
image metamorphosis
contour metamorphosis
2D metamorphosis of polygonal shapes
metamorphosis
of 3D volumetric models
of 3D polyhedral objects
particle system metamorphosis
3) According to the information that is used for correspondence determination

1.4 Classification according to the application

It is possible to distinguish two main applications of computer metamorphosis. The first one is the production of films for entertainment, didactic, or scientific purposes. The second one is the simulation of natural or industrial processes useful in engineering or science.

These two systems have different objectives. A film is first of all a mean of communication. It must transmit a certain message while simulation must solve a specific problem. Consequently in the first system the rendering is very important: the film should appear realistic but the motions can be unrealistic, there are no physical restrictions. In the second system the user expects a solution for a given problem to obey specific natural laws. In Table 1.2, we present a non-exhaustive list of possible applications of computer metamorphosis.

Table 1.2: Applications of Computer Metamorphosis

Films
For art and entertainment.
For advertisements.
For education.
Simulations
Engineering:
Creation of new objects combining characteristics of other available objects.
Industrial design.
Science:
Reproduction of natural phenomena impossible to film.
Medical imaging.
Virtual reality.

1.5Classification according to the graphical objects that are transformed

1.5.1 Image metamorphosis:

This describes a situation where one image is transformed to another. While three-dimensional object metamorphosis is a natural solution when both objects are easily modelled for the computer often the complexity of the objects makes this approach impractical. For example, many applications require transformations between complex objects. In this case, it is often easier to manipulate scanned photographs of the scene using two-dimensional image-processing techniques than to attempt to model and render the details of the object's appearance for the computer.

The simplest method for changing one digital image into another is simply to cross dissolve between them. The colour of each pixel is interpolated over time from the first image value to the corresponding second image value. While this method is more flexible than the traditional optical approach, (simplifying for example different dissolve rates in different image areas) it is still often ineffective for suggesting the actual metamorphosis from one subject to another.

Another method for transforming one image into another is to use a two dimensional particle system to map pixels from one image onto pixels from the second image. As the pixel tiles move over time the first image appears to disintegrate and then reconstruct itself into the second image.

Another transformation method involves image warping so that the original image appears to be mapped onto a regular shape such as plane or cylinder. This technique has the advantage of several real-time implementations for video.

Image metamorphosis has given very impressive results. Examples of methods that have been used so far for this purpose are [WOL88], [WOL89], [BEI92], [SEU94], [SEU95], [SEU96], [HAS97], [HAS98] and [SEU98].

Beier and Neely in [BEI92] introduced a technique for the metamorphosis of one digital image into another. This technique gives the animator high level control of the visual effect by providing natural feature based specification and interaction. The user interaction is based on the drawing of lines. A pair of lines (one defined relative to the source image the other defined relative to the destination image) defines a mapping from one image to the other. Multiple pairs of lines specify more complex transformations, in this case a weighting of the co-ordinate transformations for each line is performed. The blending is guided by user defined corresponding lines. Each intermediate frame of the metamorphosis is computed by creating a new set of line segments by interpolating the lines from their position in the initial image to their position in the final. Both images are distorted towards the position of the lines. This method gives the user fine control over the metamorphosis and has produced very impressive results.

1.5.2 Volume Metamorphosis

A general volume is a collection of scattered voxels, each of which is associated with a set of values. The problem of creating a sequence of in-between volumes transforming from one given volume to another is referred to as volume metamorphosis (or volume morphing). There are several methods in the literature following this approach for the metamorphosis of a 3D object to another [HUG92], [PAY92], [CHE95], [LER95], [CHE96], [YUE96], [COH98].

Approaches using warp and distance field interpolation:

Cohen et. al. in [COH98], present a method for the metamorphosis of two or more objects of general topology. The intermediate objects are constructed by a distance field metamorphosis. The interpolation of the distance field is guided by a warp function controlled by a set of corresponding anchor points. Some rules for defining a smooth least-distorting warp function are given. To reduce the distortion of the intermediate shapes, the warp function is decomposed into a rigid rotational part and

an elastic part. The distance field interpolation method is modified so that the interpolation is done in correlation with the warp function. The method is capable of morphing between objects having a different topological genus where no correspondence between the geometric primitives of the models needs to be established. The desired correspondence is defined by a number of anchor points provided by the animator.

Payne and Toga [PAY92], represent the objects by scalar fields. The field they use is the distance field which represents the distance of a surface as a signed magnitude. The sign designates inside/outside and the magnitude shows the distance from the point to the nearest point on the surface. Since input surfaces are often complex, a high sampling rate is desirable, which makes distance field computation very expensive. Once the distance field has been computed they use interpolation of the distance fields to compute surfaces that are intermediate between the two "keyframe" surfaces. Objects do not have to have a similar shape or identifiable corresponding features and neither object is restricted to a certain number of components. Components of different objects influence each other during averaging if they occupy overlapping regions of space. Since this method does not rely on establishing corresponding points on the key surfaces, it is both general and automatic, but for the same reason, it cannot reliably interpolate similar surfaces. For example the natural interpolation between two orientations of an object is a minimal rotation and translation. Distance fields will not seek out optimal combinations of shape preserving motions as they do not address preserving shape or volume properties. Features on the interpolated objects that might obviously correspond will not blend each other during interpolation unless there is a spatial overlap. Results are highly sensitive to the initial placement of the objects.

Chen et. al. in [CHE95] describe a method for interactive shape metamorphosis. This method consists of morphing the 2D parameter space of a pair of surface models. Beier's technique [BEI92] is used to accomplish the warping. The 2D nature of the process makes interaction easy. The user is presented with both the parametric pre-image and the resulting surface in 3D-space to assist in defining the features to morph. The surface attributes of the source models must be available in the 2D

parameter space so that they may be interpolated. There are also map parameters used, attached to each sample. The surface attributes are interpolated as well as the geometries. The samples map-parameters are also interpolated since they do serve for the construction of the target model from a morphed pre-image, this is done by using the morphed values of the map-parameters at each sample point. Traditional morphing between 2D images operates on colour as a function of 2D pixel co-ordinates; this method operates on colour as a function of 2D parametric co-ordinates.

Lerios et al in [LER95] consider 3D metamorphosis applied to volume-based representations of objects. Their method has two components: first a warping of the two input volumes, then a blending of the resulting warped volumes. The warping component is an extension of Beier and Neely's [BEI92] image warping technique to 3D, it is feature based and allows user control.

Volumetric methods using a signal approach:

Hughes [HUG92] describes a method for smoothly transforming one volumetric model to another. His technique is based on interpolating smoothly between the Fourier transforms of the two volumetric models and then transforming the results back. Since, in some cases, a linear interpolation between the transformed datasets gives unsatisfactory results, a schedule for the interpolation is used in which the high frequencies of the second model are gradually removed, the low frequencies of the first model are interpolated to those of the second and the high frequencies of the second model are gradually added in. In the drawbacks of this method we have the fact that a metamorphosis between two identical objects, one displaced from the other, is not a smooth translation and that high frequencies will be present in certain moments (the keyframes) while being absent the rest of the time.

Seok and Aoki in [AOK96] describe a new morphing technique using the Fourier transform. Two-dimensional or three-dimensional models are Fourier-transformed and the intensities of the transformed patterns are superposed and clipped, resulting in a blending pattern of two models. Morphing by the simple interpolation yields an interesting result between the models, but difficulty can arise if the models are

composed of dominant higher frequency components. They propose a new method, where the morphing of two models is done after dividing a higher-frequency dominant model into two lower-frequency dominant models. The morphing experiment was conducted to show the validity of the proposed methods using simple 2D and 3D models.

Volumetric methods using Implicit functions:

Galin and Akkouche in [GAL96] address the metamorphosis of soft objects built from skeletons. Their approach is split into three steps. The first step consists in an original splitting of the initial and the final shapes with a view to creating a bijective graph of correspondence. In the second step, they assume that the skeletons are convex polygonal shapes, and thus take advantage of the properties of Minkowski sums to characterise the skeletons of intermediate shapes. The Minkowski sum of two sets A and B, denoted by $A \oplus B = \{a+b \mid a \in A, b \in A\}$. They consider the vertices of the objects as vector endpoints and apply the Minkowski sum to the set of vertices of the two objects. Eventually, they characterise the intermediate distance and field function, describe a set of interpolation methods and propose to use a restricted class of parameterised distance and field functions so as to preserve coherence and speed-up computations. They show that they can extend those results to achieve a Bezier like metamorphosis where control points are replaced by control soft objects. Eventually, they point out that matching all components of the initial and the final shapes generates amorphous intermediate shapes based on an overwhelming number of intermediate sub-components. Thus, they propose heuristics with a view to preserving coherence during the transformation and accelerating computations.

1.5.3 Contour metamorphosis

These methods consider the problem for the metamorphosis of closed polygonal contours. [REE81], [SED92], [CHE89], [CHU94], [GUI94], [GOL96], [AOK96] and [COH96] fall into this category.

Chen in [CHE89] mainly interested in industrial design, introduces a technique for shape averaging. This method averages 2D planar polygons or 3D objects represented by a set of coaxial planar contours. The number and the correspondences of the contours is user-specified. Corresponding contours from the two objects are transformed to have a common centre point. The method proceeds by firing rays from the centres of the input shapes. Two points are assumed to be corresponding if they are intersected by the same ray. Each of these pairs of points are interpolated to create intermediate slices, which are combined to create in-between objects using a lofting technique. This method takes into consideration the topology of the objects so it results in connected intermediate shapes. It also gives some control to the user through the selection of slices.

Sederberg and Greenwood in [SED92] address the 2D shape blending problem. Their method is based on a physical model. Imagining that each shape is made of a piece of wire the blend is determined by computing the minimum work required to bend and stretch one wire shape into the other. The user can specify some physical properties of the wire that control the relative difficulty with which it can be bent or stretched. A severe penalty is charged for blends which experience a local self intersection due to the wire bending through an angle of zero degrees. This penalty nearly always prevents the self intersection problem. This method usually finds and interpolates the similar parts of the two shapes.

Goldstein and Gotsman in [GOL96] present an algorithm for morphing between two simple polygons. The two polygons are converted to multiresolution representations. Intermediate representations are generated from the multiresolution representations, from which intermediate polygons are reconstructed. They use a multiresolution shape representation , based on curve evolution schemes. This representation captures the geometric properties of a shape at different levels of detail.

Yuefeng Zhang in [YUE96] presents a new approach for polygon warping. The warping of polygonal shapes is usually separated in two steps. The first establishes a

correspondence between the vertices of two given polygons. The second step interpolates the corresponding vertices to generate vertices of an intermediate polygon. This article presents new approaches to both steps. This new algorithm uses fuzzy techniques to warp polygons that have different locations, orientations, sizes and numbers of vertices. The algorithm is extensible to curved shapes.

1.5.4 Metamorphosis of three dimensional polygonal objects

This approach involves the representation of a pair of three-dimensional objects as a collection of polygons. The vertices of the first object are then displaced over time to coincide in position with the corresponding vertices of the second object with colour and other attributes similarly interpolated. Therefore we can clearly see two steps in this approach first establishing a desirable vertex correspondence and then interpolating the co-ordinates of the corresponding vertices. Correspondence determination is a somewhat ambiguous process, related more to human judgements rather than well-understood, universally applicable definitions. We can distinguish two levels in solving this problem, the syntactic level where shapes are treated as geometric entities and the metamorphosis is concerned only with geometric properties of elements such as sizes or distances and the semantic level where correspondence is established on the basis of similarity in parts of the two objects. At the semantic level shapes are averaged according to the metaphorical properties of their elements. For example in the semantic averaging of two human head shapes, regardless of their sizes or locations the noses of the heads are corresponding and should be averaged together.

In each case the 3D model of the first object is transformed to have the shape and surface properties of the second model and the resulting animation is rendered and recorded. The methods described in [BET89], [KEN91], [KEN92], [HON88], [PAR91], [GAL96], [CAR97], [KAN97], [LAZ97], [STA98] consider the metamorphosis of three dimensional polygonal objects.

Methods using Projection:

Kent in [KEN91], [KEN92] presents a solution to the correspondence problem for Euler-valid, genus 0, polyhedral objects. The first step of his method is the projection of the topology of both models onto the unit sphere. The two topologies are merged by clipping the projected faces of one model to the projected faces of the other, on the sphere. Once the merged topology has been computed it is mapped onto the surface of both original models. This generates two new models that have the same shape as the original but share a common topology. This allows a transformation between the two shapes, to be easily computed, by interpolating the co-ordinates of each pair of corresponding vertices. Since the correspondences between the models are established by their mappings onto the sphere, different mappings result in different transformations. Thus this method gives the user some degree of control over the transformation. The main drawback of this method is that it is restricted to special categories of objects.

Decaudin and Gagalowicz in [DEC94] provide a method directly inspired by the work of Kent. Their method applies to star-shaped obects. They create intermediate shapes by creating a new shape including with them a volume equal to the sum of their volume.

Lazarus and Verust in [LAZ97] extent Kent et al.'s method [KEN91] for cylinder-like objects (objects that are star-shaped around an axis). Given a 3D curve inside each object, two cylindrical meshes are built to approximate the two objects. The two objects are morphed on these cylindrical meshes. The metamorphosis consists of an interpolation of the 3D curves composed by radial interolation of each sampling point of the mesh.

Carmel and Cohen in [CAR97] present an algorithm that builds a correspondence between two arbitrary genus-0 objects and generates a sequence of in-between objects. A warp function deforms the source object and aligns it with the target object. An iterative polygon-evolution algorithm blurs the details of the warped source and target objects into two convex objects with similar shapes that are projected on to two identical circles. Merging the topologies of the projected objects and reconstructing the original objects results in two objects with identical topologies. A two-part

transformation produces the morph sequence. The rigid part moves and rotates the objects to their relative positions. The elastic part establishes the position of each of the vertices forming the in-between object.

Geometry based methods:

Hong in [HON88] matches the centroids of the faces of two objects to establish correspondences between them. These correspondences are then used to transform one face into another. When one object has more faces than the other, the extra faces are paired with the closest face in the other object. For carefully selected objects, this technique is effective, but for arbitrary objects, severely distorted shapes occur during the interpolation. Additionally during the transformation, the faces of the in-between objects may break apart and appear to fly randomly from their initial position on one object to their final position on the other.

Kanai et. al. in [KAN97] present a new algorithm for 3D geometric metamorphosis between two objects based on the harmonic map. This algorithm is applicable for polyhedra that are homeomorphic to the three dimensional sphere or the two dimensional disk. In this algorithm, each of the two 3D objects is first embedded to the circular disk on the plane. This embedded model has the same graph structure as its 3D objects. By overlapping those two embedded models, a correspondence can be established between the two objects. Using this correspondence, intermediate objects can be generated. The user has to specify a boundary loop on an object and a vertex on that boundary which control the interpolation. Thanks to the harmonic mapping the correspondence is quite well related to the geometry of the objects. The problem of connecting, in a smooth manner, the two embedded sheets for closed genus-0 messes is not addressed.

Kaul and Rossignac [KAU91] use Minkowski sums for 3D object metamorphosis. They have developed a new animation primitive to support an interactive environment for animation design. They call it Parameterised Interpolating Polyhedron or PIP for short. PIPs are specified by providing their initial and final shapes, which may be any polyhedra and need not have corresponding boundary elements. The faces of the polyhedra have constant orientations and vertices that move on a straight line between

a vertex of the initial shape and a vertex on the final one. This technique transforms pairs of polyhedra by computing the Minkowski sum of versions of the two models. By gradually scaling one model from 100% to 0% while simultaneously scaling the other from 0% to 100% a transformation is obtained. When both the initial and final shapes are convex the resulting Minkowski sum is convex and the algorithm produces PIP faces that define the exact boundary of the intermediate polyhedra. When at least one of the initial polyhedra is not convex, the boundary of their Minkowski sum is always a subset of the faces stored in the PIP representation. These faces, stored in the PIP, may intersect each other, however portions of these faces will lie inside the Minkowski sum. The correct image of the transforming object may be produced by using a hidden surface elimination algorithm. This method uses both geometrical and topological information resulting in intermediate models having connected surfaces and exhibiting small amounts of distortion. On the other hand the user has no control over the transformation.

Methods using a graph approach:

In their 1989 article on shape distortion, Wes Bethel and Sam Uselton [BET89], outlined their 3D morphing system. This system performs a warp between two three-dimensional polygonal objects. The objects have to be polygonal but the principle behind the system can be also applied on b-spline surfaces. The first step is the construction of a B* tree for each object. The nodes of the tree represent faces and the branches adjacency relationship between faces. The user then selects one face and one vertex from that face, for each object, to set the correspondence between the objects. Once the trees and the correspondences have been defined, a union of the two object topologies can be created. The information about this union is stored to a new B* tree. If one of the objects has more faces than the other then extra faces are added to the union tree. Missing faces and vertices are added as needed. If one of the objects has a hole on it then the union will also have a hole. The result is a master object that has all the topological characteristics of each of the input objects. The most difficult part of the processing is assigning new co-ordinate values to the master object. These co-ordinates must be defined both for the initial and final keyframe. The treatment of

convex and star-shaped objects is fairly straightforward but objects with holes present more difficulties. If there is a hole in the initial frame but not in the final one then the hole should not be visible in the initial frame. The hole is really there but it has been geometrically collapsed. The resultant system can semi-automatically match the components of two objects.

Parent in [PAR91] describes a solution to the shape transformation problem for arbitrary 3D polygonal objects. His method involves first breaking the surfaces of the two objects into equal numbers of sheets of connected faces. New vertices are added along the boundaries of these sheets until there is a one-to-one correspondence of the points along the boundary of each pair of sheets. The corresponding pairs of sheets are subdivided by first finding a path of edges on one sheet which connects two vertices on the boundary of the sheet. A path of edges is then found that connects the corresponding two vertices of the other sheet, and vertices are added to the newly constructed path so that a one-to-one correspondence is obtained. These paths split each original pair of sheets into two new pairs of sheets with one-to-one correspondences between the points on their boundaries. The subdivision continues recursively until each sheet contains a single face. The subdivided single-face sheets are then recombined, resulting in two objects which have the same shape as the original objects, but which share a common vertex-edge-face structure. The transformation between the two shapes is computed by interpolating the corresponding vertex positions of the two objects. The main drawback of this algorithm is that often small regions of one model map to large regions of the other, which results in severely distorted in-between shapes.

State et. al. in [STA98] present a new approach for establishing correspondence between two homeomorphic 3D polyhedral models. The user specifies corresponding feature pairs on the polyhedra. Based on these features, their algorithm decomposes the boundary of each polyhedron into the same number of morphing patches. A 2D mapping for each morphing patch is computed in order to merge the topologies of the polyhedra one patch at a time. They create a morph by defining morphing trajectories between the feature pairs and by interpolating them across the merged polyhedron.

The user interface provides high-level control as well as local refinement to improve the morph. The system can also handle non-simple polyhedra that have holes.

Interactive methods:

De Carlo and Gallier in [DEC96] tackle the problem of morphing two polyhedral objects with different topologies. They use a sparse control mesh on each surface in order to define a mapping between the two input objects. This approach involves a heavy user interaction with the user asked to define a rough control mesh on both objects and to associate each face of one object to a face of the other one by one. This implies that the two control meshes have the same number of faces and similar topology.

1.6 Classification according to the information used for correspondence determination.

Another way to classify the algorithms establishing correspondence is what kind of information they use from the original models (geometrical or topological information). The term topology (quite different from the traditional use of the term in mathematics) refers only to the connectivity information while the term geometry is used when referring to a specific instance of the topology for which the relative vertex co-ordinates have been specified. Algorithms that use only the geometrical information usually result in disconnected intermediate models while algorithms using only the topology usually result in severely distorted intermediate models.

1.7 Interpolation

First, we will present some methods of interpolation that have been used in computer animation in general ([WAT92]) and later we will focus in some methods of interpolation particularly used for computer metamorphosis.

1.7.1 General Animation Interpolation Methods

As soon as the correspondence of the components of the two models or images has been established the actual interpolation of their positions can take place. This interpolation can be linear or spline driven.

The linear interpolation produces some undesirable side effects which give the animation a mechanical look, often referred to as the "computer signature" [CAT78]. The characteristic of this type of animation is the lack of smoothness and continuity in motion so the keyframes are clearly visible because of the sudden changes in the direction of motion. Another problem is the distortion that may occur in the situations when the movement has a rotation component. Another disadvantage of the linear interpolation scheme is the discontinuity in the speed of motion when the animator requests different number of in-between frames between successive key frames.

All these serious drawbacks of the linear interpolation resulted in the use of the splines even if that means additional information specified by the animator and more time for the calculation of the motion parameters. By spline-driven animation we mean the explicit specification of the motion characteristics of an object by using cubic splines. The most useful practical system is to have the animator interactively shape the curves and then to view the resulting animation in real time.

One method for spline-driven animation employs a velocity curve in 2D space to specify the movement of the object. The velocity curve is a two-dimensional spline of distance travelled, or arclength against time.

Given a space curve and a velocity curve, the sequence of operations required to find the position along the space curve is done as follows: Starting at time t the distance s is calculated from the velocity curve and then the element moves an amount s along the space curve.

The advantage of this scheme is that the space and the velocity curve are specified independently. Different velocity curves can be tried on the same space curve and

vice versa. Modification of the motion can be made without affecting the space curve. Useful velocity curves can be stored by the animator and used across a wide range of applications.

The velocity curve can be generalised to drive any motion parameter not only arclength. The term 'motion parameter' includes everything that moves in an animation sequence apart from the usual kinetic variables such as position and orientation. Movement could also include colour and transparency, in short anything that is representable by numbers the animator may wish to vary in time.

Steketee and Badler in [STE85] were among the first to recognise the power of specifying a velocity curve which they called a 'kinetic spline' to drive an independently splined motion parameter. Termed the 'double interpolant method', they created the following two interpolations:

1. The 'position' spline: Let the motion parameter to be interpolated be d, d is specified at n key values, $d_0,...,d_{n-1}$. The position spline is constructed by assigning a keyframe number to each key value and interpolating through the resulting tuples $(d_0, 0),...,(d_{n-1}, n-1)$.

2. The kinetic spline: Each keyframe number is assigned a time. The kinetic spline interpolates through the resulting pairs $(0, t_0),...,(n-1, t_{n-1})$.

Calculating a value for the motion parameter proceeds in a manner identical to the above. The kinetic spline provides a keyframe number given a time. The keyframe number is then fed into the position spline to give a value for the motion parameter. They also include a discussion of a slightly higher level of control than just interpolation, concerned with the transition between one motion of an object ending and another beginning.

The cubic B-spline is the most commonly used because its second derivative continuity guarantees smooth motion. Furthermore the local control property of the B-spline enables the animator to make small adjustments to the animation without the adjustments affecting the entire sequence.

Following the development of Kochanek [KOC84] we will consider now the Hermite basis. A Hermite cubic spline segment interpolates two of its control points. The remaining two specifications are not positions but vectors that determine the tangent of the curve at the interpolant points. By varying the magnitude of the tangents we can change the shape of the curve. Therefore the basis is not affine invariant.

The basis functions are:

$b_0(u)=2u^3-3u^2+1$
$b_1(u)=-2u^3+3u^2$
$b_2(u)=u^3-2u^2+u$
$b_3(u)=u^3-u^2$

A keyframing system supplies n key positions $P_0,...,P_{n-1}$ but Hermite interpolation requires the tangents $T_0,...,T_{n-1}$ to be specified in order to make up the curve consisting of n-1 segments $(P_0, P_1, T_0, T_1),...,(P_{n-2}, P_{n-1}, T_{n-2}, T_{n-1})$. Kochanek suggests making the tangent T_i at P_i a weighted sum of the source chord P_i-P_{i-1} and the destination chord $P_{i+1}-P_i$. The parameterisation of this weighting into tension, bias and continuity is done to allow the animator to fine-tune the animation, either locally or globally, without changing the key positions. The tension parameter t controls how sharply the curve bends at a key position by controlling the magnitude of the tangent at that position:

$$T_i=\left(\frac{1-t}{2}\right)\cdot((P_{i+1}-P_i)+(P_i-P_{i-1}))$$

The default, t=0, is the Catmull-Rom spline. Increasing the tension reduces the magnitude of the tangent vector down to zero at t=1, where the curve is at its tightest. For t>1 the curve will loop at the key position. Decreasing the tension causes the curve to slacken, or balloon out, at the key position.

The bias parameter b controls the relative weighting of the source and the destination chord, which dictates the direction of the tangent at a key position.

Assuming t=0, we have:

$$T_i=(\frac{1+b}{2})\cdot(P_i-P_{i-1})+(\frac{1-b}{2})\cdot(P_{i+1}-P_i)$$

The bias parameter simulates the traditional animation effect of following through an action as 'overshooting' the key position (b=1) or exaggerating a movement by 'undershooting' a key (b=-1).

1.7.2 Other interpolation methods

Here we discuss methods that have been proposed specifically for the interpolation step of computer metamorphosis. These methods try to avoid some of the undesirable effects of linear interpolation. The first one applies to the morphing of closed polygonal contours and attempts to cancel the distortion that may occur in situations when the movement has a rotation component the second extends this approach to the metamorphosis of three-dimensional polygonal objects while the last method tackles the self-intersection problem occuring during the morphing of two-dimensional polygonal shapes.

Sederberg et. al. in [SED93] present an algorithm for determining the paths along which corresponding vertices travel in a 2D polygon shape blending. The polygons are described in terms of the lengths of their edges and the angles at their vertices. The intermediate polygons in the shape blend are computed by interpolating the respective vertex angles and edge lengths. A problem with this approach is that the resulting polygon will not close in general. To overcome this, Sederberg considers the open polygon as a piece of wire and defines physical rules for bending and stretching. The adjustments to angles and/or edges are computed by determining the equilibrium shape when the two open polygon vertices are forced to coincide.

Yue Man Sun. et. al. in [YUE95] present an extension of the intrinsic shape interpolation scheme for 2D polygons [SED93]. Rather than considering a polyhedron as a set of independent points or faces, their solution treats a polyhedron as a graph

representing the interrelations between faces. Intrinsic shape parameters, such as dihedral angles and edge lengths that interrelate the vertices and faces in the two graphs, are used for interpolation. This approach produces more satisfactory results than the linear or cubic curve paths would, and is translation and rotation invariant.

Shapira et. al. in [SHA95] present a method for automatic polygon blending that takes into account polygon interiors as well as boundaries. This approach is based on a new polygon representation scheme, the star skeleton. A star polygon is a polygon for which there exists a point, the star point, for which all other parts are visible. The star skeleton consists of two parts. First the decomposition of the polygon into star-shaped pieces, and a special star point called the star origin, second a planar graph, the skeleton, that joins the star origins. This approach interpolates shapes by first interpolating between their skeletons and then unfolding the star pieces from the interpolated skeletons.

1.8 Conclusions

This chapter presented the background of this dissertation and discussed methods that have been proposed for the metamorphosis of three-dimensional objects.

Summarising these methods we could say that methods that use only geometric information of the original models like [HON88] result in disconnected shapes while methods that are purely topological like [BET89] can lead to heavily distorted intermediate objects.

In general most methods proposed for determining the correspondences fall into two categories. They are either automatic where the sole criteria for two features corresponding is their relative position in space, or rely on extensive guidance from the user which for non trivial objects can become a very tedious and time consuming task.

The first category consists of the methods presented in [HUG92], [PAY92], [KEN91], [CHE89], [KAN97], [KAU91]. For these methods the process is entirely automatic, it relies heavily on the initial orientation of the objects. There is a very low level of control of the animation. Even the slightest change of orientation results into a completely different morph.

The second category consists of the methods presented in [LER95], [CHE95], [DEC96], [GAL96], [STA98]. These methods rely heavily on user interaction.

The methods presented in [CAR97] and [COH98] appear to have found the right balance between user interaction and automation. The user specifies a limited number of features and the rest of the correspondences are calculated automatically. [CAR97] mainly addresses the 2D case, the extension to three-dimensional objects seems far from straightforward. The work of [COH98] has given very impressive results, it applies to volumetric models of general topology but it is somehow limited to the types of transformation it can create (for example it is not easy to give a rotational effect in the animation).

Looking at the results, it seems that the volume based approach is reaching maturity with the work of Cohen et. al in [COH98], while the polygonal approach could still benefit from further improvement.

The use of polygonal representation has several advantages. The corresponding data structures are more compact than the storage of voxelized objects. Many practical and efficient algorithms are available to visualize objects represented by polygonal representation. It is also very easy to attach properties such as colour, normal, or texture to polygonal representations. Therefore a method that successfully solves the correspondence determination problem for polygonal objects is highly desirable.

This dissertation presents a method for the metamorphosis of 3D polygonal objects, this method uses minimal user interaction but at the same time succeeds establishing correspondences on the basis of the object's similarities.

Chapter 2

Energy Minimisation for Computer Metamorphosis

2.1 Introduction

A common approach in shape metamorphosis involves the polygonal representation of a pair of objects. The vertices of the first object are then displaced over time to coincide in position with the corresponding vertices of the second object. There are two steps in this approach: first establishing a desirable vertex correspondence and then interpolating the co-ordinates of the corresponding vertices.

Most of the methods for correspondence determination in the semantic level that have been proposed rely on user interaction or on information implied by the object's modelling representation. The former approach can be extremely tedious for complex objects, effectively forcing the user to specify correspondences for all the object vertices, while the later limits the metamorphosis to objects of a certain category, objects where semantic information is incorporated into their representation during the modelling step.

The next section presents an algorithm attempting to solve the correspondence determination problem of two planar polygonal contours in the semantic level. This approach assumes that two features of the objects are similar if one needs small changes to transform one to the other. The measure to quantify these "small" or "big" changes is the energy needed to perform them assuming that the objects are made of a piece of wire. This is the basis of Sederberg's and Greenwood's algorithm [SED92].

In this chapter we will discuss this algorithm and propose some possible extensions to it, while in the following chapters we will attempt to extend this approach for the metamorphosis of two-dimensional and three-dimensional polygonal objects.

2.2 Sederberg's and Greenwood's Algorithm overview

Sederberg's and Greenwood's algorithm, addresses the 2D shape-blending problem and is based on a physical model. Imagining that each shape is made of a piece of wire, then the metamorphosis is determined by computing the minimum work required to bend and stretch one wire shape into another. The user can specify some physical properties of the wire which controls the relative difficulty with which it can be bent or stretched. A severe penalty is charged for animations that experience a local self-intersection due to the wire bending through an angle of zero degrees. This penalty nearly always prevents the self- intersection problem.

Given two polygons P_0 and P_1 with the same number of vertices, shape blending is accomplished by performing a linear or spline interpolation between the corresponding vertices of the two polygons. Let P_k^i denote vertices.

$$P_0=[P_0^0, P_1^0,..., P_n^0]; \qquad P_1=[P_0^1, P_1^1,..., P_n^1]$$

Assuming linear interpolation, intermediate polygons of the metamorphosis can be defined as:

$$P(t) = uP_0+tP_1 = [uP_0^0 + tP_0^1, uP_1^0 + tP_1^1,..., uP_n^0 + tP_n^1] = [P_0(t), P_1(t),..., P_n(t)]$$

Where $t \in [0,1]$ and $u=1-t$.

Typically two polygons that are morphed do not initially have the same number of vertices and even if they do, defining a one to one correspondence, according to the order of the vertices along the contour, will not usually generate a pleasing

metamorphosis. Therefore the principal task in shape blending is that of adding vertices to each polygon so that both polygons end up with the same number of vertices, and the resulting vertex correspondences produce a desired metamorphosis.

The approach that is proposed is modelling the polygons as pieces of wire made of some idealised material. The best shape blend is one that requires the least work to deform P_0 to P_1 through bending and stretching.

Whenever we define that two consecutive vertices P^0_i, P^0_{i+1} of the first contour correspond with two successive vertices P^1_j, P^1_{j+1} of the second, we also define that edge $P^0_iP^0_{i+1}$ of the first contour should be stretched to become edge $P^1_jP^1_{j+1}$. In the same manner, whenever we define that three consecutive vertices P^0_i, P^0_{i+1}, P^0_{i+2} of the first contour correspond with three successive vertices P^1_j, P^1_{j+1}, P^1_{j+2} of the second, we also define that angle $P^0_iP^0_{i+1}P^0_{i+2}$ of the first contour should be bent to become angle $P^1_jP^1_{j+1}P^1_{i+2}$ of the second. Therefore stretching work is computed for each line segment (that is, each adjacent pair of vertices) whereas bending work is computed for each adjacent pair of line segments (that is, for each set of three adjacent vertices).

Stretching work:

A force F will stretch an actual wire of length L_0 an amount _ given by:

$$\delta = \frac{FL_0}{AE}$$

where A is the cross sectional area and E is the modulus of elasticity, a constant of the material.

The work expended in stretching a real piece of wire an amount δ is:

$$W = \frac{\delta^2 AE}{2L_0} \qquad (2.1)$$

AE is a constant for the wire so it can be replaced with a user defined constant k_s the 'stretching stiffness constant'.

If L_0 is the initial length of a section of wire and L_1 is its final length equation 2.1 will compute different values if the initial and the final shapes are swapped. Furthermore, if an edge collapses to a single vertex equation 2.1 requires infinite work (since $L_0=0$). The exponent 2 in equation 2.1 assumes the wire is linearly elastic. If excessive stretching occurs, less work is required to elongate the wire because it undergoes plastic deformation. In this case an exponent $e_s \cong 1$ more closely expresses the work expended.

These three considerations motivate the following modification to equation 2.1 [SED92]:

$$W_s = k_s \frac{(L_1 - L_0)^{e_s}}{(1-c_s)\min(L_0, L_1) + c_s \max(L_0, L_1)} \qquad (2.2)$$

Where k_s , e_s , c_s are user defined constants.

Bending work:

The authors of [SED92] point out some undesirable situations of blends. One of them occurs when an angle goes to zero (in Figure 2.1 we present the drawing of an incorrect animation, the angle at the circled vertex passes through zero), resulting in two edges meeting at that vertex and passing over one another. When this occurs at least part of the shape is turning 'inside out'.

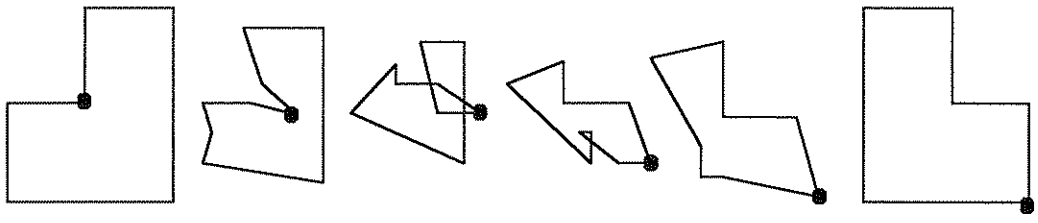


Figure 2.1: Simple example 'solution' 1

Another example of ill-behaved intermediate angles is ones not changing monotonically during the animation. Consider the drawing presented in Figure 2.2

here the terminal angles at the circled vertex are both 90°, yet those angles in the intermediate frames exceed 130°.

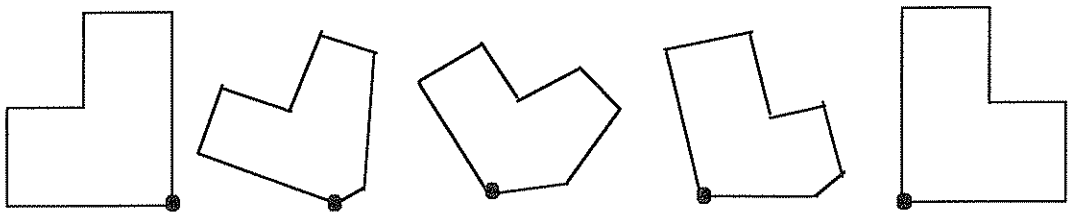


Figure 2.2: Simple example 'solution' 2

Work that causes bending is defined as [SED92]:

$$W_b = \begin{cases} k_b \cdot (\Delta\theta + m_b \cdot \Delta\theta^*)^{e_b} & \text{if the angle never goes to zero} \\ k_b \cdot (\Delta\theta + m_b \cdot \Delta\theta^*)^{e_b} + p_b & \text{if the angle does go to zero} \end{cases} \qquad (2.3)$$

where $\Delta\theta$ is the change of the angle between two consecutive edges by bending, $\Delta\theta^*$ shows how much this angle deviates from monotonicity during the animation and m_b, k_b, p_b, e_b are user defined constants. The constant m_b penalises angles that do not change monotonically, p_b penalises angles that go to zero (causing local intersection), k_b indicates bending stiffness and e_b is an exponent that simulates the effect of plastic deformation due to excessive bending.

Least work solution:

The authors of [SED92] impose the condition that vertices can only be inserted at existing vertices. Otherwise it becomes a non-linear constrained optimisation problem whose solution is very expensive and whose global optimality is difficult to verify.

Given two polygons P^0 with m vertices and P^1 with n vertices all vertex correspondences can be represented by an $m \leftrightarrow n$ matrix. The columns of the matrix represent vertices of the first polygon and the rows of the matrix vertices of the second. A correspondence between the i'th vertex of the first polygon and the j'th

vertex of the second polygon is represented on the matrix as a '1' at the junction of column i and row j. Furthermore [i, j] is only allowed to be a correspondence if only [i-1, j], [i, j-1] or [i-1, j-1] is also a correspondence, otherwise intermediate polygons in the shape transformation would split apart. Given that [0, 0]=[m, n] is a valid correspondence, a complete solution can be represented on the matrix as a string of '1' starting at [0, 0] and ending at [m, n] with each subsequent dot positioned one step 'east', 'south' or 'southeast' from the preceding dot.

The basic correspondence strategy proceeds by considering polygon fragments consisting of vertices $P_0^0,\ldots, P_i^0$ of the polygon P^0 and vertices $P_1^1,\ldots, P_j^1$ of the polygon P^1. These polygon fragments are denoted as $P^0(i)$ and $P^1(j)$. The minimum work required to transform P(i) into P(j) can be computed according to the work equations (2.2) and (2.3). Therefore it is possible to find the path that connects [0, 0] with [i, j] using the minimum amount of work denoted W(i, j). This is accomplished observing that W(i, j) must equal one of the three predecessor values W(i-1, j), W(i, j-1), W(i-1, j-1) plus the incremental work involved connecting that predecessor with [i, j]. By keeping information about the predecessor of [i, j] it is possible to backtrack the path to [0, 0].

The algorithm for computing the least work solution amounts to setting W(0, 0)=0 and from the work equations computing W(i, j) i=0,...,m; j=0,...,n. Then W(m, n) is the optimal least work solution and the path which leads to this least work solution can be backtracked by following the stored pointers. A dot [k, h] in the path denotes a correspondence of the k'th vertex of the first polygon with the h'th vertex of the second. Therefore at this point by having the path it is possible to determine the vertex correspondence of the two polygons.

The least work solution can be determined by visiting each junction only once hence the complexity of this algorithm is O(mn).

The above discussion assumes that the first vertex of the first polygon corresponds to the first vertex of the second but in the same way it is possible to compute a globally minimum work solution for any initial correspondence.

Figure 2.3 presents a computer-generated animation obtained by the application of Sederberg's and Greenwood's algorithm. The correspondence of the vertices preserves the similar features of the two shapes.

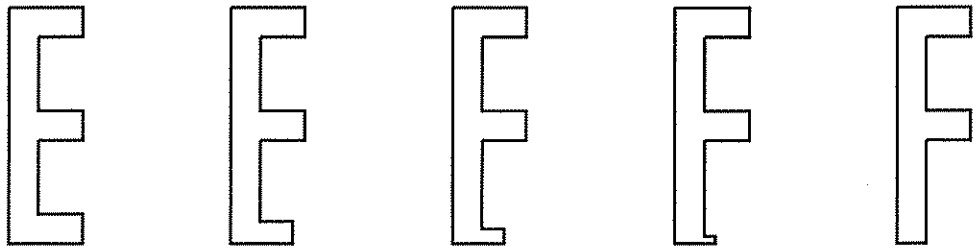


Figure 2.3: An animation obtained by Sederberg's and Greenwood's algorithm

2.3Defining more pairs of corresponding vertices on the two contours

Let's assume that k pairs of corresponding vertices are defined on the two contours (Figure 2.4). These vertices will separate each initial contour to k sub-contours. We denote as m_i, n_i the number of vertices of the i'th sub-contour of the first and second contour respectively.

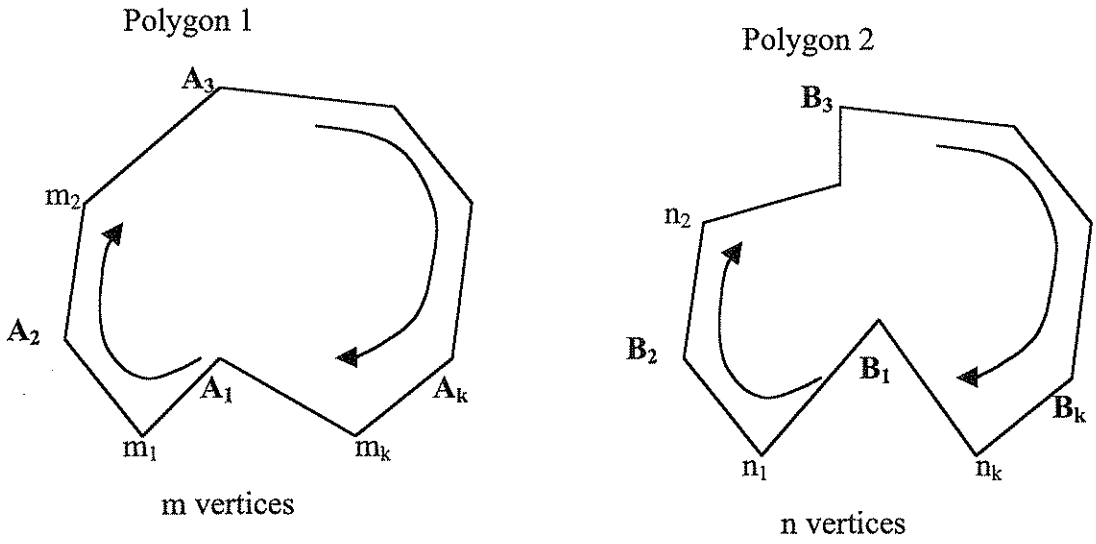


Figure 2.4: k pairs of corresponding vertices have been defined on the two contours.

In this case sub-contour A_1A_2 corresponds to sub-contour B_1B_2, sub-contour A_2A_3 to sub-contour B_2B_3 and so on. These pairs of corresponding sub-contours can be treated separately. The number of operations needed to calculate the vertex correspondences of the first pair of mapped sub-contours is in the order of $m_1 \cdot n_1$ similarly the number of operations needed for the second pair is in the order of $m_2 \cdot n_2$ and so on. Therefore the total number of operations needed is:

$$\sum_{i=1}^{k} (m_i \, ?n_i)$$

If we assume that the k corresponding vertices divide the contour into sub-contours with the same number of vertices then $m_1=m_2=\ldots=m_k=m/k$. In this case the total number of operations is in the order of:

$$\sum_{i=1}^{k} (m_i \, ?n_i) = \frac{m}{k} ?_1 + \frac{m}{k} ?n_2 + \ldots + \frac{m}{k} ?n_k = \frac{m \, ?n}{k}$$

Therefore by providing k pair of corresponding vertices on the two polygons and assuming that these vertices are evenly distributed around the two contours, the amount of computation is reduced by a factor of k.

2.4 Allowing varying stiffness for different parts of the wire

The original Sederberg's and Greenwood's work model assumes that each wire has uniform stiffness. There was an extension of the original algorithm implemented that allows the user to specify that some portions of the wire are more stiff than others, suggesting a relative difficulty in altering those portions. In the case that the same polygonal contours are used repeatedly, it is worth incorporating the information about the wire stiffness in the modelling information of the contours. In Figure 2.5 we present a computer-generated animation obtained with the application of Sederberg's and Greenwood's algorithm to a pair of polygonal contours.

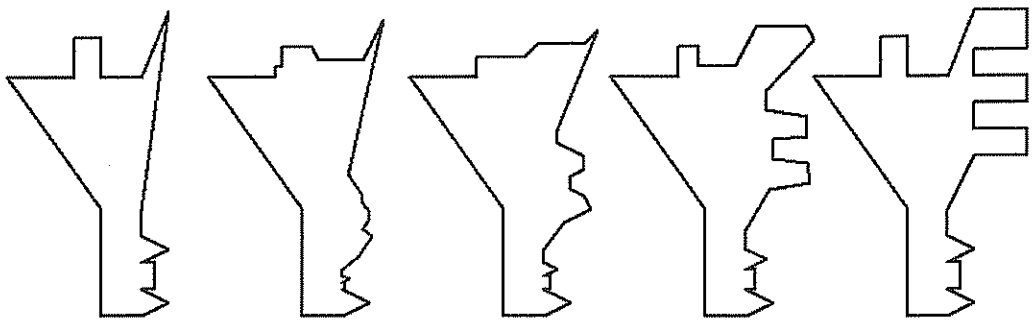
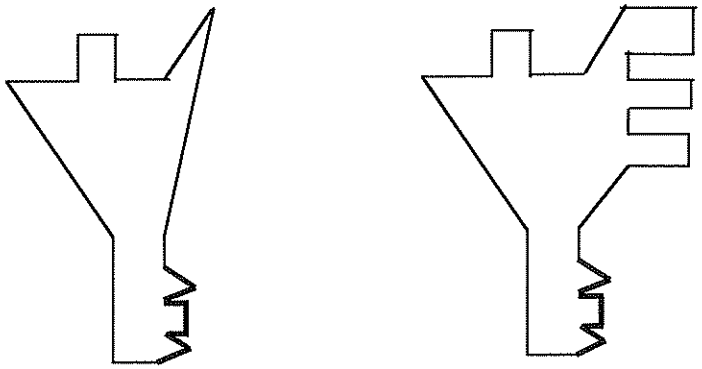
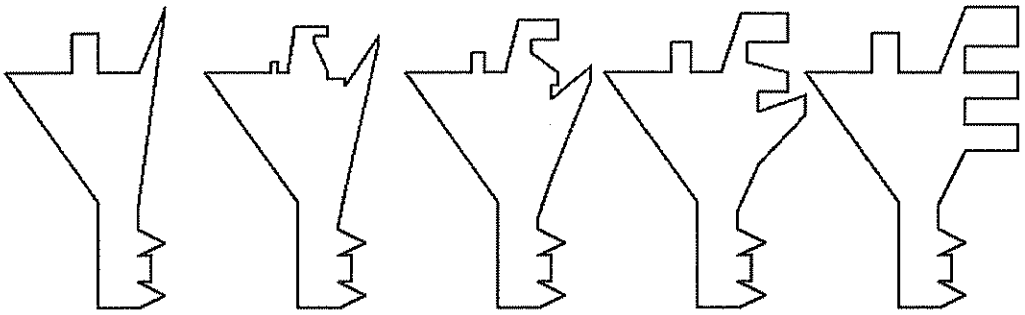


Figure 2.5: Morphing two contours using the Sederberg's and Greenwood's Algorithm

If we specify that parts of the wire are more difficult to bend and stretch (the highlighted parts of the contours in the drawing of Figure 2.6a) we obtain the computer-generated animation of Figure 2.6b, This animation preserves the identical parts of the two contours.



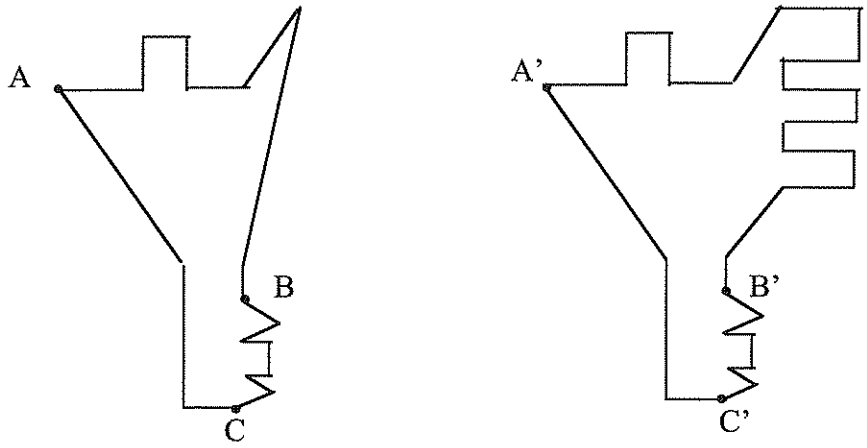
a) Specifying different degrees of stiffness for different parts of the wire



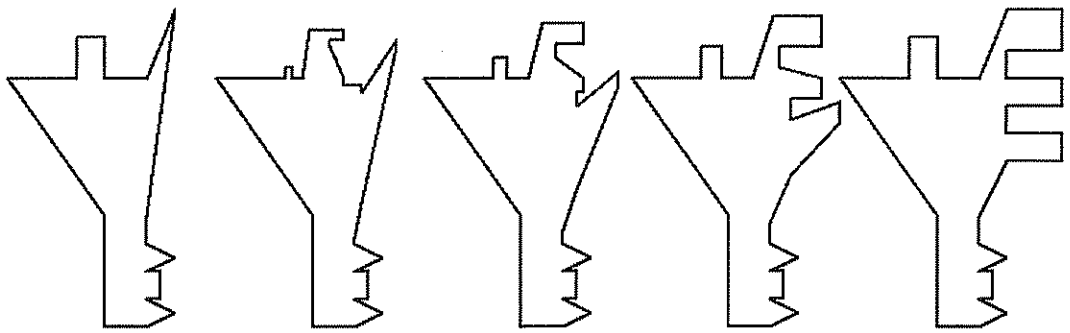
b) The animation sequence

Figure 2.6: Varying the wire stiffness

Even though it was demonstrated through examples that this can lead into quite interesting animations we believe that is a lot more intuitive and that a higher level of control is obtained by defining multiple pairs of corresponding vertices on the two contours. In Figure 2.7b we present the same animation obtained by specifying three pairs of corresponding vertices on the two contours (the highlighted vertices in the drawings in Figure 2.7a).



a) Specifying three pairs of corresponding vertices



b) The animation sequence

Figure 2.7: Specifying more than one pair of corresponding vertices

2.5 Conclusions

This chapter presented a physically based algorithm [SED92], developed for the morphing of planar polygonal contours. Experiments have demonstrated that this method gives very satisfactory results, avoiding self-intersections and identifying similarities between the two terminal contours. This algorithm will be the basis of the methods described in the following chapters for the metamorphosis of 2D and 3D polygonal shapes.

Chapter 3

Metamorphosis of Simple Two-dimensional Polygonal Objects.

3.1 Introduction

A two-dimensional polygonal shape consists of a number of polygons so that every non-contour edge belongs to exactly two polygons and no vertex of the shape is inside a polygon. In this chapter we will consider situations of morphing between some special categories of two-dimensional shapes, these shapes consist of a set of independent domains that can be geometrically ordered in space. The method proposed reduces the problem of correspondence determination for these domains to a shortest path problem in a directed graph. At the end of this chapter we outline how a similar methodology could be used for the metamorphosis of special categories of three-dimensional polygonal objects. In the next chapter the proposed method will be extended to more general two-dimensional polygonal shapes.

3.2 Basic concepts from Graph theory

Here we present the basic concepts of Graph theory and the terminology which will be used in this thesis [MCH90], [MIN88].

Graphs are mathematical objects that can be used to model networks, data structures, process scheduling, computations, and a variety of other systems where the relations between the objects in the system play a dominant role.

Definition 3.1: A *graph* G(V, E) consists of a set V of elements called *vertices* and a set E of unordered pairs of members of V called *edges*. An edge connecting a pair of vertices u and v is denoted by (u, v). The cardinality of V, denoted |V|, is called the *order of G*, while the cardinality of E, denoted |E|, is called the *size of G*.

Definition 3.2: A vertex u in V is adjacent to a vertex v in V if (u, v) is an edge in E. The vertices u and v are called the endpoints of the edge (u, v) and edge (u, v) is *incident with* the vertices u and v.

Definition 3.3: The degree of a vertex v, denoted by *deg(v)* is the number of edges incident with v.

Definition 3.4: A *path* from a vertex u to a vertex v in G is a sequence of edges (u, e_1), $(e_1, e_2),\ldots,(e_k, v)$ where all the vertices and edges in the sequence are distinct.

Two vertices are connected if and only if there is a path from one to the other. A graph G is *connected* if and only if every two vertices in G are connected. A graph which remains connected even if we remove any one of its edges is *strongly connected.*

Definition 3.5: A graph G where directions have been imposed on the edges of a graph, interpreting the edges as ordered rather than unordered pairs of vertices is called a *directed graph.*

If G is a directed graph and (u, v) is an edge of G, the edge (u, v) is *incident into* v, and *incident from* u.

Definition 3.6: Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are *isomorphic* if there is an one to one correspondence M between V_1 and V_2 such that u and v are adjacent in G_1 if and only if the corresponding vertices $M(u)$ and $M(v)$ are adjacent in G_2.

Definition 3.7: A Graph G(V, E) is called a *planar graph* if it can be drawn or embedded in the plane in such a way that the edges of the embedding intersect only at the vertices of G.

3.3Morphing of shapes consisting of parts that can be geometrically ordered.

Let's consider a two-dimensional polygonal shape such as every non-contour edge of it belongs to exactly two polygons and no vertex of the shape is inside a polygon. This shape can be described by a planar graph where the nodes of the graph correspond to the vertices of the polygons and the edges of the graph to the edges of the polygons.

Definition 3.8: The vertices of the graph, having a degree greater than two, are those where three or more polygonal contours meet. We call these *primary vertices*.

Definition 3.9: A path connecting two primary vertices without containing any other primary vertex will be referred as a *simple path.*

We will consider the problem of morphing two two-dimensional polygonal shapes such as two of the contour vertices belong to all polygons and all the other non-contour vertices belong to two polygons. These shapes consist of two primary vertices connected by a set of polylines, when each polyline is a simple path (Figure 3.1). The only common vertices that two polylines have, are the primary vertices. Since the polylines do not intersect, they can be ordered according to their geometric position on the plane. The position of a polyline in this ordering will be used as an index for referencing it.

Suppose that the first shape consists of m polylines and the second shape of n polylines (Figure 3.1). These polylines can be ordered from 1 to m for the first shape and from 1 to n for the second. It is presumed that the primary vertices should

correspond so that vertex A of the first shape corresponds to vertex A' of the second shape and vertex B of the first shape corresponds to vertex B' of the second shape.

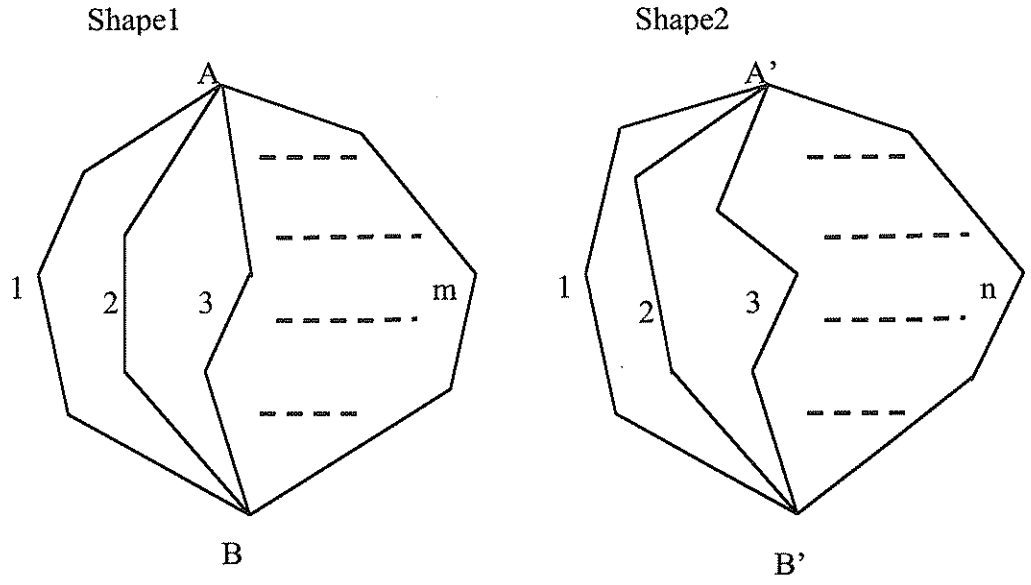


Figure 3.1: Two shapes consisting of two primary vertices and a set of primary paths joining them

This section presents a method for determining how the polylines should correspond so that the shape transformation is accomplished with least work. Sederberg's and Greenwood's algorithm will be used to determine the minimum energy needed to transform a polyline to another and the correspondence of the polyline vertices that results in this minimum energy transformation.

A *valid solution* to the correspondence problem of the polylines is one that for every polyline of one shape finds one or more corresponding polylines of the other and this correspondence leads to an animation sequence without self-intersections. When the order of the polylines is not maintained during the animation at least one polyline crosses another one. Therefore in a valid solution it is expected that the order of polylines is preserved. In other words, it is expected that if the i'th polyline of the first shape corresponds to the j'th polyline of the second shape then there is no polyline of the first shape with index greater than i corresponding to a polyline of the second shape with index less than j and there is no polyline of the first shape with index less than i corresponding to a polyline of the second shape with index greater than j.

The contours of the two shapes should correspond otherwise self-intersections will occur during the animation sequence. Therefore, we assume that the first and m'th polylines of the first shape correspond to the first and n'th polylines of the second shape respectively.

3.4Expressing the polyline correspondence problem as the shortest path problem in a directed graph.

We construct a directed graph G containing mn vertices denoted [i, j] i=1,...,n, j=1,...,m. The vertex [i, j] represents a correspondence of the i'th polyline of the first shape with the j'th polyline of the second shape. The edges of G are defined so that each vertex [i, j] (with i>1 and j>1) has three edges incident into it namely ([i-1, j-1], [i, j]), ([i, j-1], [i, j]) and ([i-1, j], [i, j]). This leads to a directed graph having vertex [1, 1] as a source and vertex [m, n] as a sink; these vertices will be denoted *s* and *t* respectively. The vertices *s*, *t* represent the correspondences for the contour polylines of the two shapes. We will refer to graph G as the correspondence graph.

Figure 3.3 illustrates the graph constructed for the two shapes of Figure 3.2

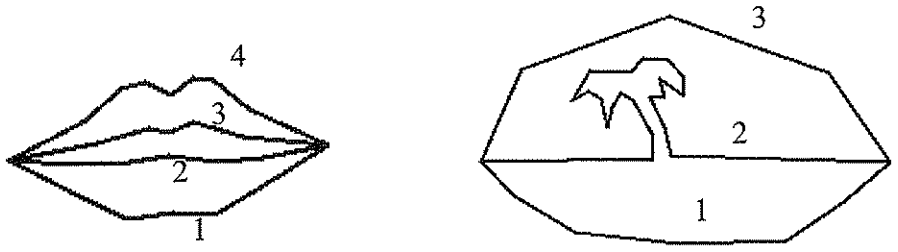


Figure 3.2: Two shapes, the first one consists of four polylines and the second of three.

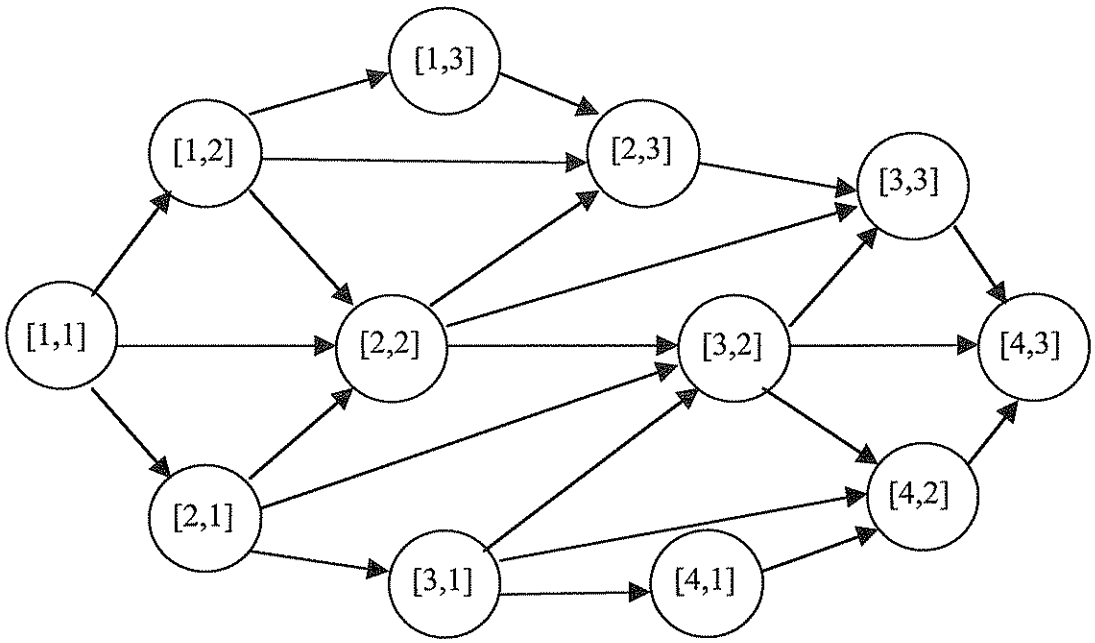


Figure 3.3: The graph constructed for the metamorphosis of a shape consisting of four polylines transformed to a shape consisting of three.

***Lemma 3.1*:** if vertices $v=[i_v, j_v]$ and $p=[i_p, j_p]$ of the correspondence graph G belong in the same path from vertex *s* to vertex *t* and vertex *v* precedes vertex *p* in the path then $i_v \leq i_p$ and $j_v \leq j_p$.

We will use induction.

The above lemma is true if the distance of the two vertices on the path is 1. Vertex *p* is adjacent to vertex *v*. The only such vertices are $[i_v+1, j_v+1]$, $[i_v, j_v+1]$ and $[i_v+1, j_v]$. Therefore vertex *p* must be one of these vertices, which means that i_p is either i_v or i_v+1 and j_p is either j_v or j_v+1. Hence $i_v \leq i_p$ and $j_v \leq j_p$ holds.

Suppose that *Lemma 3.1* is true if the distance between two vertices in the path is d. We will prove that the above lemma is true when the distance between two vertices in the path is d+1.

Let g=[i_g, j_g] be the vertex immediately preceding *p* in the path, therefore $i_g \leq i_p$ and $j_g \leq j_p$. The distance of vertices *v*, *g* in the path is d, therefore from our assumption, $i_v \leq i_g$ and $j_v \leq j_g$. Which leads to $i_v \leq i_p$ and $j_v \leq j_p$.

Theorem 3.1: There is a one-to-one mapping between the set of the paths in the correspondence graph G, from vertex *s* to vertex *t,* and the set of all valid solutions to the correspondence problem of the polylines.

Proof: In order to prove *Theorem 3.1* must be demonstrate that:

Lemma 3.2: Every path of G from vertex *s* to vertex *t* represents a valid solution to the correspondence problem of the polylines.
Lemma 3.3: For every valid correspondence of the polylines, there is a unique path from vertex *s* to vertex *t* in G representing it.

In order for *Lemma 3.2* to hold, it must be shown that
Lemma 3.4*: Each path from vertex [1, 1] to vertex [m, n] in G contains vertices representing a correspondence for every polyline of the two shapes.*
Lemma 3.5*: This set of correspondences for the polylines preserves the order of the polylines during the animation sequence.*

First, Lemma 3.4 will be proven:

Let us assume that there is a path P from vertex *s* to vertex *t* in G that does not imply any corresponding polyline for a number of polylines of the two shapes. Let i be one of these polylines with the smallest index (without loss of generality it can be assumed that this polyline belongs to the first shape). Index i cannot be equal to 1 since the path P contains vertex *s*=[1, 1] (thus implying a corresponding polyline for polyline 1). There is at least one corresponding polyline for polyline i-1. Let *v*=[i-1, k] be the last vertex in the path, representing a correspondence of polyline i-1. Vertex *v* cannot be the last vertex of the path since $i-1<i\leq m$ and the last vertex of the path is vertex *t*=[m, n]. Therefore, there must be an edge leaving vertex *v* incident to another vertex *q* of the path. Vertex *q* cannot have the first element of its ordered pair equal to

i (since it was assumed that there is no vertex belonging in path P implying a correspondence for polyline i of the first shape). Vertex q cannot have the first element of its ordered pair equal to i-1 either (since v is the last vertex of the path, representing a correspondence for polyline i-1). This contradicts the definition of graph G, since the only vertices that are adjacent from the vertex v=[i-1, k] are the vertices [i-1, k+1], [i, k] and [i, k+1]. The contradiction occurred from our assumption that there is a path in G from vertex s to vertex t that does not imply a correspondence for a number of polylines. Therefore each path in G, from vertex [1, 1] to vertex [m, n], contains vertices representing a correspondence for every polyline of the two shapes.

Now we will prove *Lemma 3.5*:
Let us assume that the order of polylines in the two shapes is not maintained when correspondences are established. In this case there exist i, j, k, h so that i<j and k>h and the i'th polyline of the first shape corresponds to the k'th polyline of the second shape while the j'th polyline of the first shape corresponds to the h'th polyline of the second shape.

We will show that vertices [i, k], [j, h] with i<j and k>h cannot belong to the same path from vertex s to vertex t.

If vertices [i, k], [j, h] belong to the same path either vertex [i, k] precedes vertex [j, h] in the path or the other way around. Without loss of generality we can assume that vertex [i, k] precedes vertex [j, h]. From Lemma 1 it is i≤j and k≤h, which contradicts our assumption that k>h. Similarly it can be shown that vertex [j, h] cannot precede vertex [i, k]. Therefore vertices [i, k], [j, h] with i<j and k>h cannot belong to the same path from vertex s to vertex t. This ensures that the order of the polylines is maintained throughout the blend thus avoiding the intersection of polylines.

To prove *Lemma 3.3* we must show that for every valid solution to the correspondence problem of the polylines there is a specific path representing it and this path is unique.

Let's S be a valid solution to the correspondence problem of the polylines, this solution will contain a set of ordered pairs (i, j) representing a correspondence of the i'th polyline of the first shape with the j'th polyline of the second shape. We order these pairs using their first element as the primary index and the second element as a secondary index. Let (x, y), (w, z) be two consecutive pairs in this ordering we will prove that i) $0 \leq w-x \leq 1$ and ii) $0 \leq z-y \leq 1$.

It is obvious that i) holds from the way the pairs are ordered.
In order to prove ii) we first show that $0 \leq z-y$. Let us assume that $z-y<0$. In this case $y<z$ and since $x<w$ this contradicts our assumption that S is a valid solution for the correspondence of the polylines.

Now we will show that $z-y \leq 1$. Let's assume that $z-y>1$ then there exist a polyline of the second shape having an index k so that $y<k<z$. let r be the index of one of its corresponding polylines. The pair (r, k) belongs in S. We will consider three cases for the value of r; r is equal to x or w, r is less than x and r is greater than w. We will show that in all those cases there is a contradiction occurring.

1) r is equal to either x or w: In this case the pair (r, k) should be between pairs (x, y), (w, z) in S which contradicts the hypothesis that these pairs are consecutive.

2) $r<x$: It is also $k>y$, therefore the pairs (r, k), (x, y) cannot belong to the same valid solution for the correspondence of the polylines. This contradicts our assumption that S is a valid solution for the correspondence of the polylines.

.

3) $r>w$:It is also $k<z$, therefore the pairs (w,z), (r, k) cannot belong to the same valid solution for the correspondence of the polylines. Which contradicts our assumption that S is a valid solution for the correspondence of the polylines.

Therefore there is a contradiction occurring for every value of r. The contradiction occurred from our assumption that $z-y>1$ therefore it is $z-y \leq 1$.

Since there is an edge in G from every vertex $v=[v_1, v_2]$ to all the vertices $p=[p_1, p_2]$ such that $0 \leq p_1-v_1 \leq 1$ and $0 \leq p_2-v_2 \leq 1$ it is obvious that there is a path in G from vertex

s=[1, 1] to vertex t=[m, n] traversing all the vertices representing the pairs of every valid solution S.

Now we will show that this path is unique. Suppose that there exist two different paths P_1, P_2 representing the same solution to the correspondence problem of the polylines. These paths must pass from the same set of vertices but traverse this set of vertices in a different order.

Let us find the first place that the two paths differ. Suppose that the i'th vertex of path P_1 is vertex v=[i_v, j_v] while the i'th vertex of path P_2 is another vertex p=[i_p, j_p]. Since path P_1 must also contain vertex p, p must follow v in P_1 hence:

$i_v \leq i_p$ and $j_v \leq j_p$ (3.3)

In the same manner path P_2 must contain vertex v, therefore v, must follow p in P_2 therefore:

$i_p \leq i_v$ and $j_p \leq j_v$ (3.4).

From (3.3) and (3.4) it follows that $i_p=i_v$ and $j_p=j_v$, indicating that vertex p is the same with vertex v which contradicts the assumption that vertices p and v differ. The contradiction occurred from the hypothesis that there exist two different paths representing the same solution for the correspondence of the polyliines, therefore the path is unique.

3.4.1 Representation of the energy as edge length

We assign lengths to the edges of graph G. The length assigned to the edge ([i, j], [k, h]) is equal to the energy needed to transform polyline k to polyline h (as given by Sederberg's and Greenwood's algorithm) plus the energy needed to change the angle between polylines i, k to the angle between the polylines j, h (In Figure 3.4 the angles _, _). Every edge of G has a length equal to the energy needed to transform a part of the first shape to a part of the second shape. The length of a path from vertex [1, 1] to vertex [m, n] equals to the energy needed to perform the complete transformation of the first shape to the second shape according to its equivalent correspondence solution for the polylines. Therefore *the minimum energy*

transformation is accomplished by finding the solution that is represented by the shortest path from vertex [1, 1] to vertex [m, n] in G.

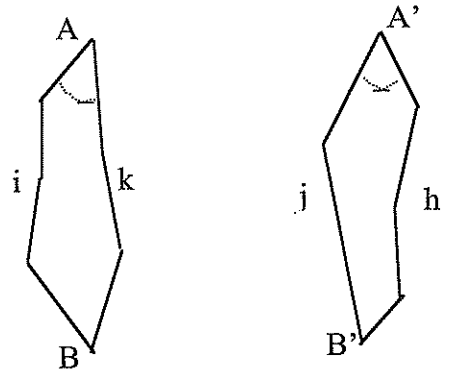


Figure 3.4: Angle between two polylines

Figure 3.5 presents the computer generated animation sequence obtained with the application of this method.

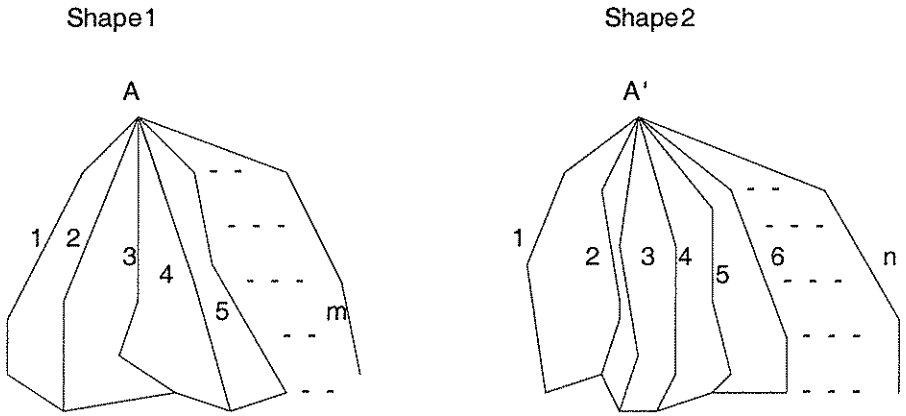
Figure 3.5: An animation example. The initial shape consists of four polylines while the final shape of three

3.5 Extension to another category of polygonal shapes

The method described in the previous section can be extended to morph a similar category of shapes consisting of a contour and a set of simple paths connecting a certain vertex of the contour to other contour vertices. Figure 3.6 presents a pair of shapes of this category.

Suppose that we want to morph the shapes of Figure 3.6. We assume that the primary vertices A, A' should correspond. Again the same directed graph G is constructed but

in this case the length of edge ([i, j], [k, h]) in G is equal to the sum of the energy needed to transform the k'th polyline of the first shape to the h'th polyline of the second shape (In Figure 3.5, the energy needed to change polyline AC to polyline A'E), the energy needed to change the simple path joining the i'th polyline to the k'th polyline in the first shape, to the simple path joining the j'th polyline to the h'th polyline in the second shape (In Figure 3.7, the energy needed to change polyline BC to polyline DE) and the energy needed to change the angle that polylines i, k form in A to the angle that polylines j, h form in A' (In Figure 3.7 the energy needed to change angle _ to _). The shortest path in G from vertex [1, 1] to vertex [m, n] represents the solution to the correspondence problem of the polylines. Note that the correspondence of the simple paths joining the polylines is implied by the correspondence of the polylines.



3

Figure 3.6: A pair of shapes.

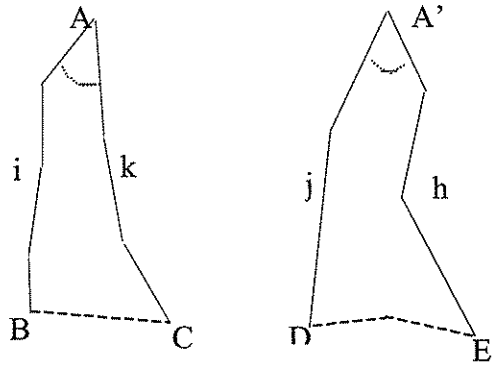


Figure 3.7: Two consecutive polylines and the angle between them.

3.6 Implementation details

Initially the length of the edges of the graph G is calculated by applying Sederberg's and Greenwood's method. When we have the lengths of all the edges of G we compute the shortest path from vertex [1, 1] to vertex [m, n]. Suppose that we want to find the shortest path from vertex [1, 1] to vertex [i, j], denoting L([i, j]) the length of this path. This is easily accomplished by the simple observation that this shortest path must pass through one of the predecessor vertices of vertex [i, j] namely [i-1, j], [i, j-1] or [i-1, j-1], its length will be equal to the length of the shortest path to one of these predecessors (L([i-1, j]), L([i-1, j-1]) or L([i, j-1]) plus the length of the edge connecting this predecessor vertex to the vertex [i, j]. Therefore

L([i, j])=min{(L([i-1, j])+length([i-1, j], [i, j]), L([i, j-1])+length([i, j-1], [i, j]), L([i-1, j-1])+length([i-1, j-1], [i, j])}

When the value L([i, j]) has been determined it is stored in a mn matrix and a pointer is kept to the predecessor vertex of the path that resulted in this value.

The algorithm for computing the least work solution for the polyline correspondence amounts to setting L([1, 1])=0 and computing L([i, j]) i=1,...,m; j=1,...,n. L([m, n]) is then the length of the shortest path from vertex [1, 1] to vertex [m, n] in G which is equal to the minimum energy for the transformation of the first shape to the second shape. The correspondence of the polylines that results in this minimum energy transformation is determined by backtracking these stored pointers. Knowing the correspondence of the polylines, Sederberg's and Greenwood's algorithm gives us the correspondence of the vertices which is already computed.

The algorithm for filling the matrix is presented in Figure 3.8. The procedure *SEDERBERG* calculates the energy needed to morph a pair of polylines, while the procedure *ANGLE* calculates the energy needed in order to change the angle that two polylines of the first shape form to the angle formed by their corresponding polylines of the second shape. The matrix *Previous* contains the pointers used to backtrack the corresponence of the polylines when the matrix has been filled.

Algorithm 3.1 Calculation of correspondence between two sets of polylines

Input: Two sets of simple paths (polylines)

Output: The value of the minimum energy needed to transform the first set to the second, and an m by n matrix *Previous*. Every element of *Previous[i, j]* will contain two fields *Previous[i, j].col*, *Previous[i, j].row* these two fields act as pointers to another element of *Previous*.

```
for i ← 1 to m do
        for j ← 1 to n do
                Edge[i, j]=SEDERBERG(i, j);

Work[1, 1] ← 0;
for i ← 2 to m do
begin
        morph_energy ← Edge(i, 1);
        Work[i, 1] ← Work[i-1, 1]+ANGLE(i-1, 1, i, 1)+ morph_energy;
        Previous[i, 1].col ← i-1;
        Previous[i, 1].row ← 1;
end;

for j ← 2 to n do
begin
        morph_energy ← Edge(1, j);
        Work[1, j] ← Work[1, j-1]+ANGLE(1, j-1, 1, j)+ morph_energy;
        Previous[1, j].col ← 1;
        Previous[1, j].row ← j-1;
end;

for i ← 2 to m do
        for j ← 2 to n do
        begin
                morph_energy ← Edge(i, j);
                Energy1 ← Work[i, j-1]+ANGLE(i, j-1, i, j)+ morph_energy;
                Energy2 ← Work[i-1, j-1]+ANGLE(i-1, j-1, i, j)+ morph_energy;
                Energy3 ← Work[i, j-1]+ANGLE(i, j-1, i, j)+ morph_energy;

                if (Energy1<Energy2)AND(Energy1<Energy3) then
                begin
                        Work[i, j] ← Energy1;
                        Previous[i, j].col ← i;
                        Previous[i, j].row ← j-1;
                end;
                else
                        if (Energy1<Energy2)AND(Energy1<Energy3) then
                        begin
```

```
                Work[i, j] ← Energy2;
                Previous[i, j].col ← i-1;
                Previous[i, j].row ← j-1;
        end;
        else
                begin
                        Work[i, j] ← Energy3;
                        Previous[i, j].col ← i-1;
                        Previous[i, j].row ← j;
                end;
end;
```

Figure 3.8: Algorithm for filling the matrix

Figure 3.9 presents the algorithm used for backtracking the pointers in order to find the shortest path in G.

Algorithm 3.2 Calculation of correspondence between polylines

Input: The matrix Previous returned by algorithm 3.1

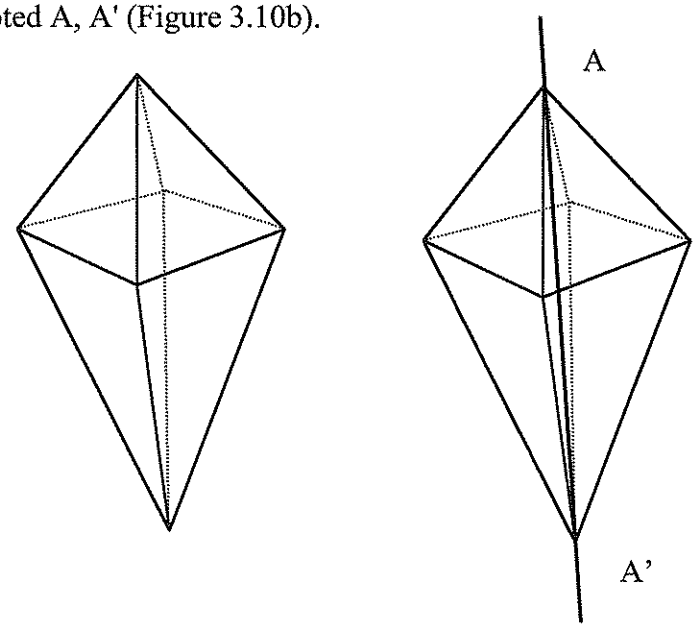
Output: A list *Solution* containing (i, j) pairs representing the correspondence of the i_{th} polyline of the first polyline set to the j_{th} polyline of the second polyline set.

```
x ← m;
y ← n;
Solution ← add(x, y);
while (x!=0)OR(y!=0) do
begin
        x ← Previous[x, y].col;
        y ← Previous[x, y].row;
        Solution ← add(x, y);
end;
```

Figure 3.9: Pseudocode for backtracking the stored pointers

3.7Application of a similar methodology to three-dimensional polygonal objects

This section describes how this approach can be extended to handle the metamorphosis of certain categories of three-dimensional objects. For the time being convex objects will be considered. Let's consider a convex object like the one presented in Figure 3.10. Its intersection with a straight line results in a single pair of points denoted A, A' (Figure 3.10b).



(a) (b)

Figure 3.10: A convex object and its intersection with a straight line

Let's consider a plane that contains this line (Figure 3.11a). This plane will intersect this object resulting in a contour containing the points A, A' or two sub-contours AA' (moving clockwise), AA' (moving anti-clockwise). If the line is used as an axis for the rotation of the plane more sub-contours are created (Figure 3.11b).

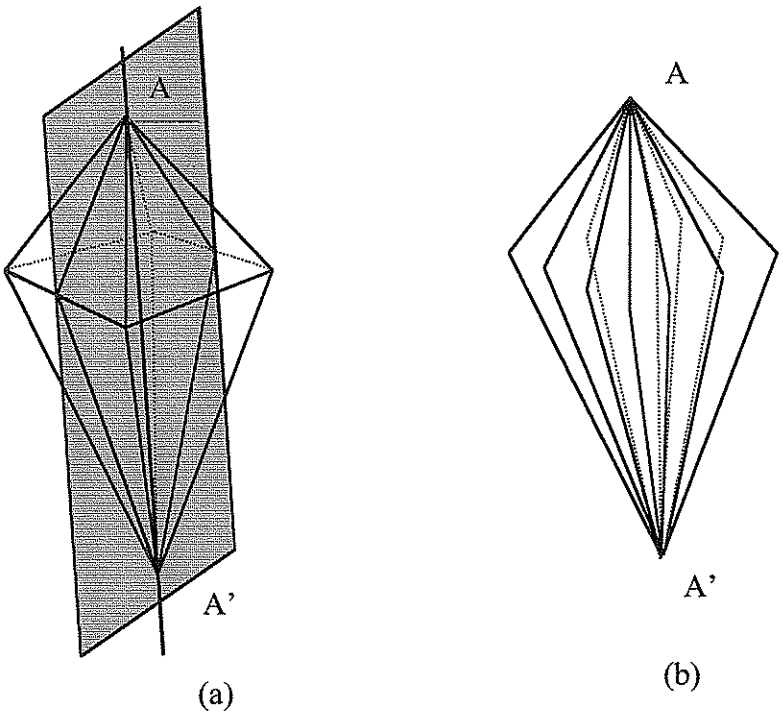


Figure 3.11: Intersection of the object with a set of rotating planes

Depending on the increment of the rotation a different number of sub-contours is created. All this sub-contours have just two common points, namely the points A, A'. The problem of morphing a set of sub-contours (describing an object) to another (describing another object) is similar to the problem described in the previous section for the polyline sets and since all the points of one sub-contour are on the same plane it is possible to use a similar methodology. This will give us a correspondence of the sub-contour sets and their vertices. The interpolation of these corresponding vertices will result to intermediate sub-contour sets. If we triangulate the surface between every two successive intermediate sub-contours we can create the intermediate objects of the metamorphosis. We are not going to describe this method in any more detail since a similar (and more powerful) method is described extensively in chapters 5 and 6.

Of course in general the intersection of a rotating plane with an object can result into more than one contour. But still the method can be applied to many non-convex objects, objects that they are star-shaped across the intersecting line.

3.8 Complexity

Let us consider the situation where the first shape consists of m and the second shape of n polylines. The algorithm operates in two steps, the first step is the filling of the matrix and the second is the backtracking of the stored pointers. The filling of the matrix (Figure 3.8) requires the calculation of the energy needed for the morphing between every pair i, j of polylines (where i is a polyline of the first shape and j a polyline of the second shape). Let v_i, v_j denote the number of vertices of the polylines i and j respectively. Since we are using Sederberg's and Greenwood's method for the calculation of energy the number of the necessary operations is on the order of $v_i \cdot v_j$.

Let $|V_1|$, $|V_2|$ denote the total number of vertices of the first and the second shape respectively while v_{1i}, v_{2i} denote the number of vertices in the i'th polyline of the first and the second shape respectively. It is:

$$\sum_{i=1}^{n} v_{1i} = |V_1| + n \cup |V_1|$$

and

$$\sum_{j=1}^{m} v_{2j} = |V_2| + m \cup |V_2|$$

The total number of necessary operations is in the order of:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} (v_{1i} ?v_{2j}) = \sum_{i=1}^{n} ((|V_1| + n) ?v_{2j}) = (|V_1| + n) ?(|V_2| + m) \cup |V_1| ?|V_2|$$

Therefore the complexity of this step is $O(|V_1| \_ |V_2|)$.

When we have filled the matrix its entry [m, n] contains the length of the shortest path. What is actually important is the correspondence of the polylines that resulted in this shortest path. The correspondence of the polylines can be defined by backtracking the pointers that have been stored for each matrix

junction, during the filling of the matrix. Starting from the junction [m, n] we follow the stored pointers till we reach junction [1, 1] (Figure 3.9). At each step being on a junction [i, j] we follow the stored pointer that leads either 'north' to junction [i-1, j] or 'west' to junction [i, j-1] or 'northwest' to junction [i-1, j-1]. Clearly the maximum number of steps occurs when there are no 'northwest' steps: in that case we have to do m steps in the north direction and n steps in the west direction for a total of m+n steps There are at most m+n junctions visited, therefore the complexity of the algorithm is O(m+n).

3.9 Conclusion

This chapter presented a methodology for the morphing of shapes that consist of a set of distinct parts that can be geometrically ordered in space. The problem of determining the correspondence of these parts was reduced to the problem of computing the shortest path of a directed graph. The categories of shapes that were studied in this chapter are very specialised and probably of little practical importance but served the purpose of creating the framework for the application of this method. In the following chapters this methodology will be applied to more general shapes where their separation into parts is not as straightforward.

Chapter 4

A Method for the Metamorphosis of General Two-Dimensional Polygonal Objects

4.1 Introduction

This chapter extends the method described in the previous chapter to the metamorphosis of more general two-dimensional polygonal objects. In order to apply the method described in Chapter 3, a new representation of the shapes is considered, this new representation preserves the same geometric information but assigns a new connectivity structure to the shapes. Several animations obtained with the application of this method are presented. Since the resulting animations are not immune to problems of self-intersection some possible solutions are discussed.

4.2 Tree structured shapes

Initially we will consider morphing between shapes with graph representation having the property such that you can remove a set of consecutive contour edges and the resulting graph becomes a tree. We will refer to these shapes as 'tree structured'. This category of shapes is probably of little practical importance but will help as an intermediate case between the shapes studied in Chapter 3 and the most general shapes that we will consider later in this chapter. A shape of this category can be

represented as a set of paths starting from the root and ending at the leaves of the tree. Starting from the root of the tree and moving anti-clockwise across the contour we can order these paths based on the order that we encounter the endpoint of each path on the contour. The problem of morphing between two 'tree structured' shapes can be reduced to the problem of morphing between the two set of paths representing them. This problem can be treated as discussed in Section 3.5 by considering the paths leading to the leaves of the tree as simple paths. Figure 4.1 presents a tree-structured shape and Figure 4.2 the new representation that is built from it. Note that in the new representation of the shape the vertex B has been replicated as B_1, B_2, B_3, B_4 are treated like different vertices even though they have the same geometric information.

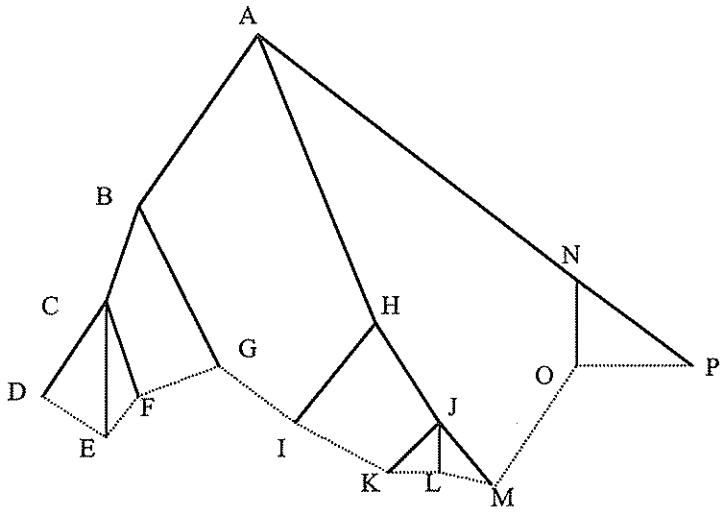


Figure 4.1: A Tree-structured shape

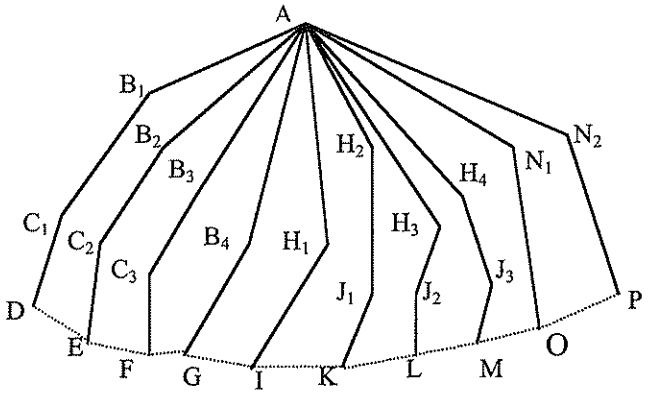
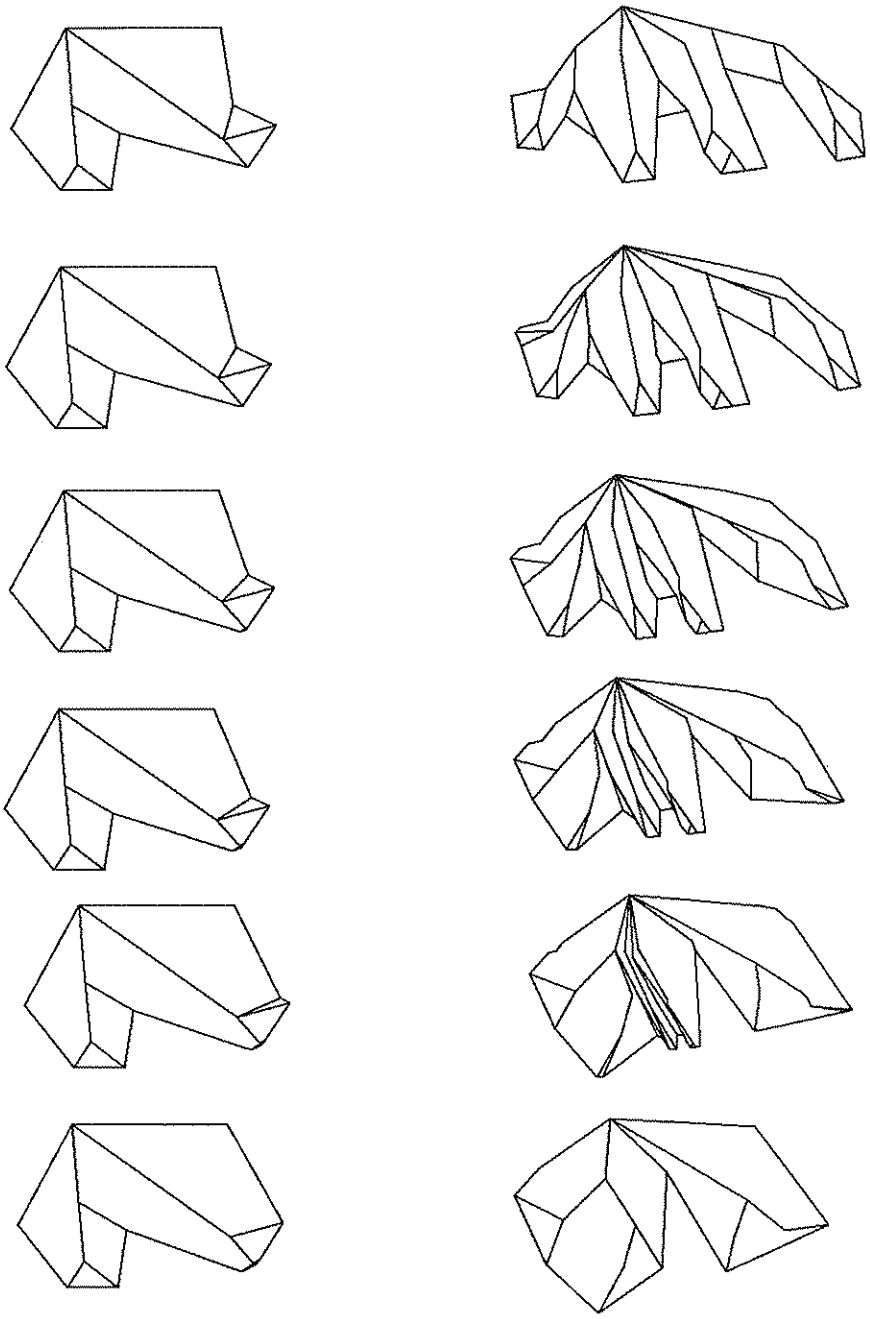


Figure 4.2: The new representation we consider for the shape of Figure 4.1.

In Figure 4.3, two computer generated animations of 'tree-shaped' objects are shown. The first one is a metamorphosis of two shapes of similar topology, while the second presents the metamorphosis of two objects with very dissimilar shapes. In column one note how the similar parts of the two shapes are maintained throughout the metamorphosis.



(a) Animation 1 (b) Animation 2

Figure 4.3: Two animations of Tree-structured shapes

4.3 General two dimensional polygonal objects

This section deals with the metamorphosis between general two-dimensional polygonal shapes. We assume that the user has defined a pair of corresponding primary vertices one on the contour of each shape.

In general, shapes do not belong to any of the simple categories that were discussed so far. The method employed for more general shapes is treating them as instances of simpler shapes. For each shape a new topology is created, to represent it. This new representation consists of a set of paths each starting from the vertex of the contour with a given correspondence and ending on another contour vertex. The method proceeds by first assigning directions to the edges of the graph and then following the directed edges to find the paths.

There is more than one way of obtaining a directed graph. We follow the assumption that the application of the same rules for the direction of the two graphs exploits their structural similarities. The method that is used for the direction of the graph is a breadth first search. Starting from a point of the contour with a given correspondence a branch of the search will terminate when it reaches a point of the contour. The points of the contour are the leaves of the tree that is built for the shape. This preserves the correspondence of the two contours, thus avoiding crossovers during the transformation.

Figure 4.4 illustrates the two steps for the representation of a shape as a set of paths that can then be treated with the method described in Chapter 3. The first step is making the graph of the shape directed and the second is the creation of a new graph representation.

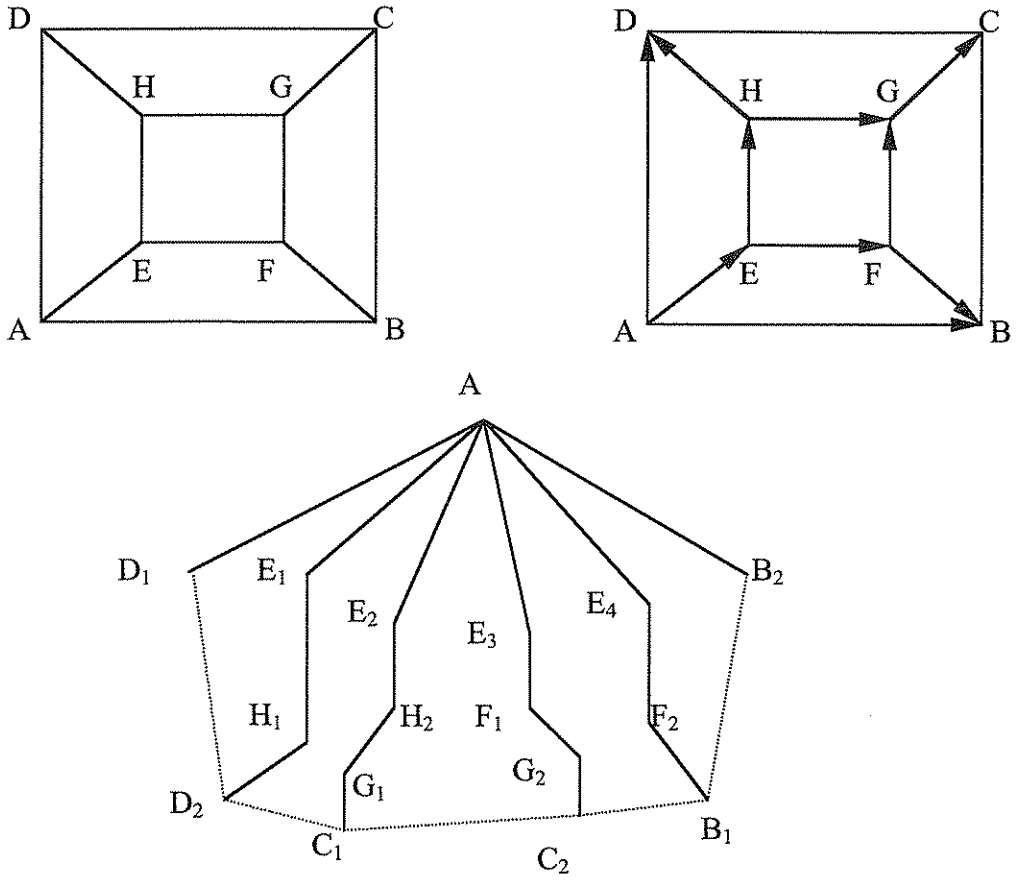


Figure 4.4: Representing the shape as a set of paths

As soon as this new representation has been created for the two shapes it is possible to apply the algorithm, that was described in section 3.5, on these set of paths to interpolate them. Next section describes in more detail how this method operates.

4.4 Algorithm for finding a directed graph.

Consider the problem of morphing between two two-dimensional polygonal shapes. We assume that a user has defined a pair of corresponding primary contour vertices, one for each shape. The first step for the metamorphosis method is finding the contours of the two terminal shapes. We will calculate the order of the contour vertices in a clockwise order.

Let v be the vertex of the shape for which a correspondence has been defined, we select an edge (v, x) incident from v and visit x (Figure 4.5c). For all unexplored edges (x, w) we calculate the right hand screw angle _ that the edges (v, x), (x, w) form. The edge (x, w) that creates the greatest angle $\__{max}$ (Figure 4.5d) is followed leading to vertex w. In general, suppose h is the most recently visited vertex. If (k, h) is the edge we followed to reach h we calculate the right hand screw angle _ that the edges (k, h), (h, m) form for all unexplored edges (h, m). The edge (h, m') that creates the greatest angle $\__{max}$ is followed leading to vertex m'. If we have not yet returned to the initial vertex v, we start the same procedure anew starting at vertex m'. This procedure results either to a polygon of the shape (Figure 4.5d) moving anti-clockwise or to the contour of the shape (Figure 4.5f) moving clockwise. We can determine if the polygon calculated is the contour of the shape by adding all the intermediate angles _ ($\_=\pi-\__{max}$). If the polygon calculated is the contour of the shape the sum of all _ angles is 2π otherwise the sum is -2π and the procedure described should be repeated for a different initial edge (v, x).

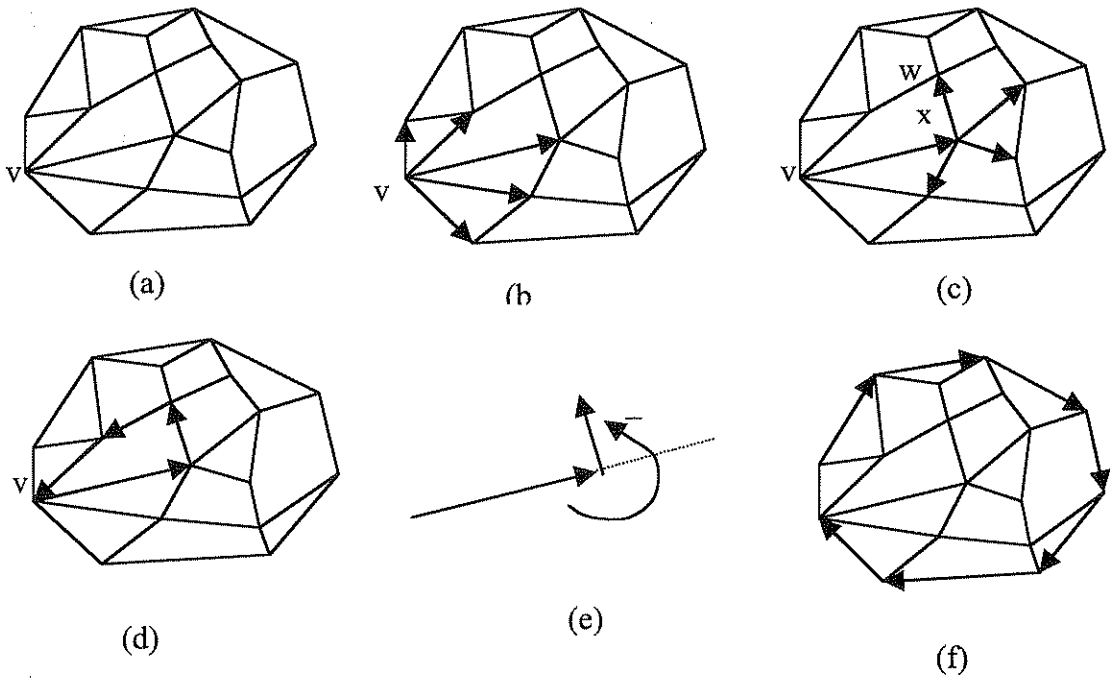


Figure 4.5: Calculating the contour of a shape

As soon as the contour of the shapes has been calculated we proceed to direct the graph of each shape

Let $v, c_1,\ldots,c_n$ be the contour of the shape. We start again from the vertex v. For all the vertices b_i adjacent to v we calculate the right hand screw angle that the edges (b_i, v), (c_n, v) form. We store the vertices b_i in a list C(v) ordered according to this angle. We insert all non contour vertices of C(v) in a list L_1, keeping track that they were reached from vertex v. Let x be the first vertex of L_1, for all unexplored edges (x, n_i) we calculate the right hand screw angle $\_i$ that the edges (v, x), (x, n_i) form. We store the vertices n_i in a list C(x) ordered according to their respective angle $\_i$. All non contour vertices of C(x) are inserted in a list L_2 keeping track that these vertices were reached from x. We repeat the same procedure for the next vertex of L_1 and so on till the list is exhausted. Then we move to the list L_2 and visit its first vertex. In general suppose that we are going through the vertices of a list L_i and h is the most recently visited vertex. Let k be the vertex of L_{i-1} that led us to h, for all unexplored edges (h, m_i) we calculate the right hand screw angle $\_i$ that the edges (k, h), (h, m_i) form. We store the vertices m_i in a list C(h) ordered according to their respective angles $\_i$. All non contour vertices of C(h) not already in L_{i+1} are inserted in L_{i+1}, we also keep track that these vertices were reached from vertex h. Then we visit the next vertex of L_i and follow the same procedure. This is repeated till there are no more vertices in L_i. Then we move to the list $L_{i+1}$and start the same procedure anew starting from the first vertex of L_{i+1}. This procedure is repeated till there are no more unexplored edges. This results in a set of lists C that effectively describe the directed graph.

Algorithm 4.1 Directing the graph

Input: A graph G(V, E) presented as adjacency list L(v) for v∈ V. A list of the vertices *Contour* belonging to the contour of the shape.

Output: A set of descendancy lists C(v) for v∈ V that effectively describe G as a directed graph

The following figure (Figure 4.6) presents the algorithm used for directing G.

```
procedure DIRECT(previous, v)
begin
        Mark v "old";
        for each vertex w on L[v] do
                if w is marked "new" then
                        begin
                        CALCULATE_ANGLE(previous, v, w);
                        INSERT(C(v), w);
                        For each vertex x in C(v) then
                        if w not in contour then
                                DIRECT(v, w);
                end
end
```

Figure 4.6: Algorithm for the directing the graph.

Figure 4.7 presents the algorithm used for calculating the set of paths

Algorithm 4.2 Finding the paths that will describe the shape.

Input: A directed graph G(V, E) represented by descendancy lists C(v) , for $v \in$ V.

Output: A set of paths starting from the vertex of the shape with a given correspondence and ending on the contour of the shape.

```
procedure PATHS_FIND(v, path)
list    temp;
list    p;
begin
        temp ← path;
        ADD(temp, v);

        if v is a contour node then
        begin
```

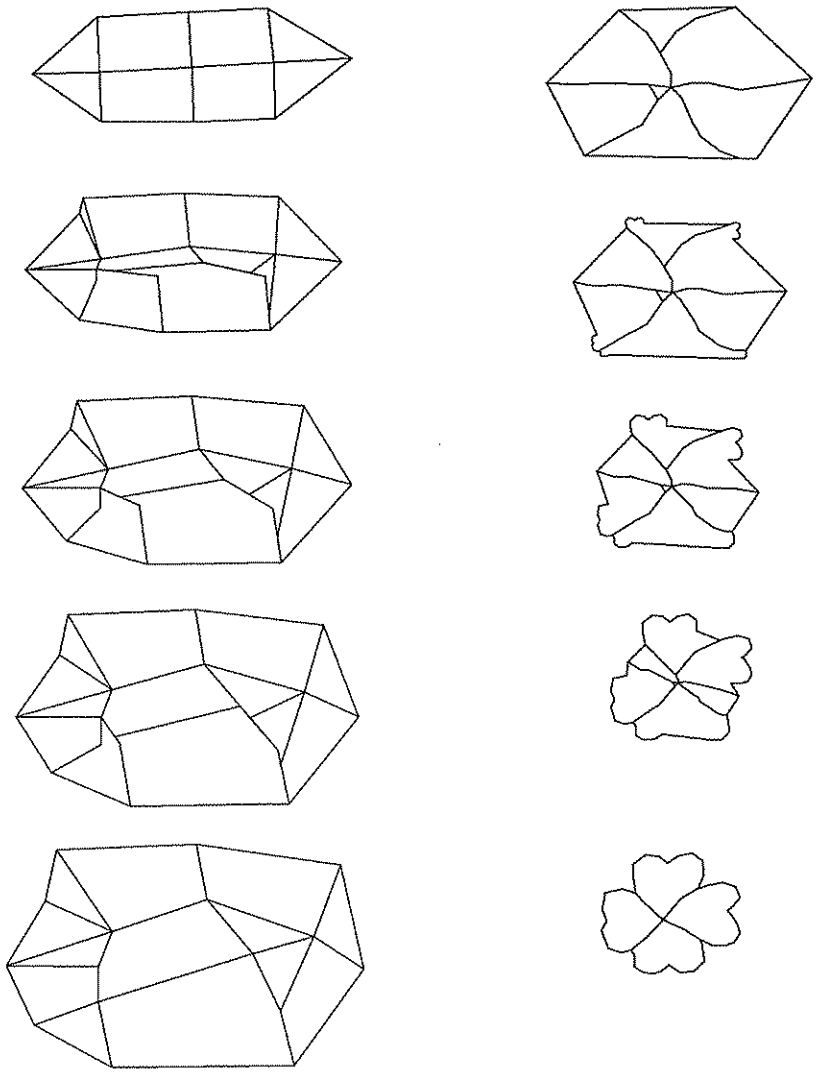
```
            numpaths ℜ numpaths+1;
            Paths[numpaths] ℜ temp;
        end

        if (C(v) is not empty) then
        begin
            PATHS_FIND(p->element, temp);
            while (C(v) is not empty) do
            begin
                C(v) ℜ p;
                p ℜ p->next;
                If (p not  empty)
                    PATHS_FIND(p->element, temp);
            end
        end
end
```

Figure 4.7: Algorithm for finding the paths

As soon as the paths are calculated for both shapes we can apply the method described in Section 3.5 to calculate the correspondence of the paths and their vertices. Having the correspondence of the paths we can interpolate their corresponding vertices to create the intermediate shapes.

Figure 4.8 illustrates two computer generated animation sequences obtained with the application of this method.



(a) Sequence 1 (b) Sequence 2

Figure 4.8: Two animation sequences.

4.5 Correction of the final animation sequence

Even though the described method keeps the domains invariant there are situations like the computer generated animation presented in Figure 4.9. Here the vertex correspondence leads to overlapping during the animation.

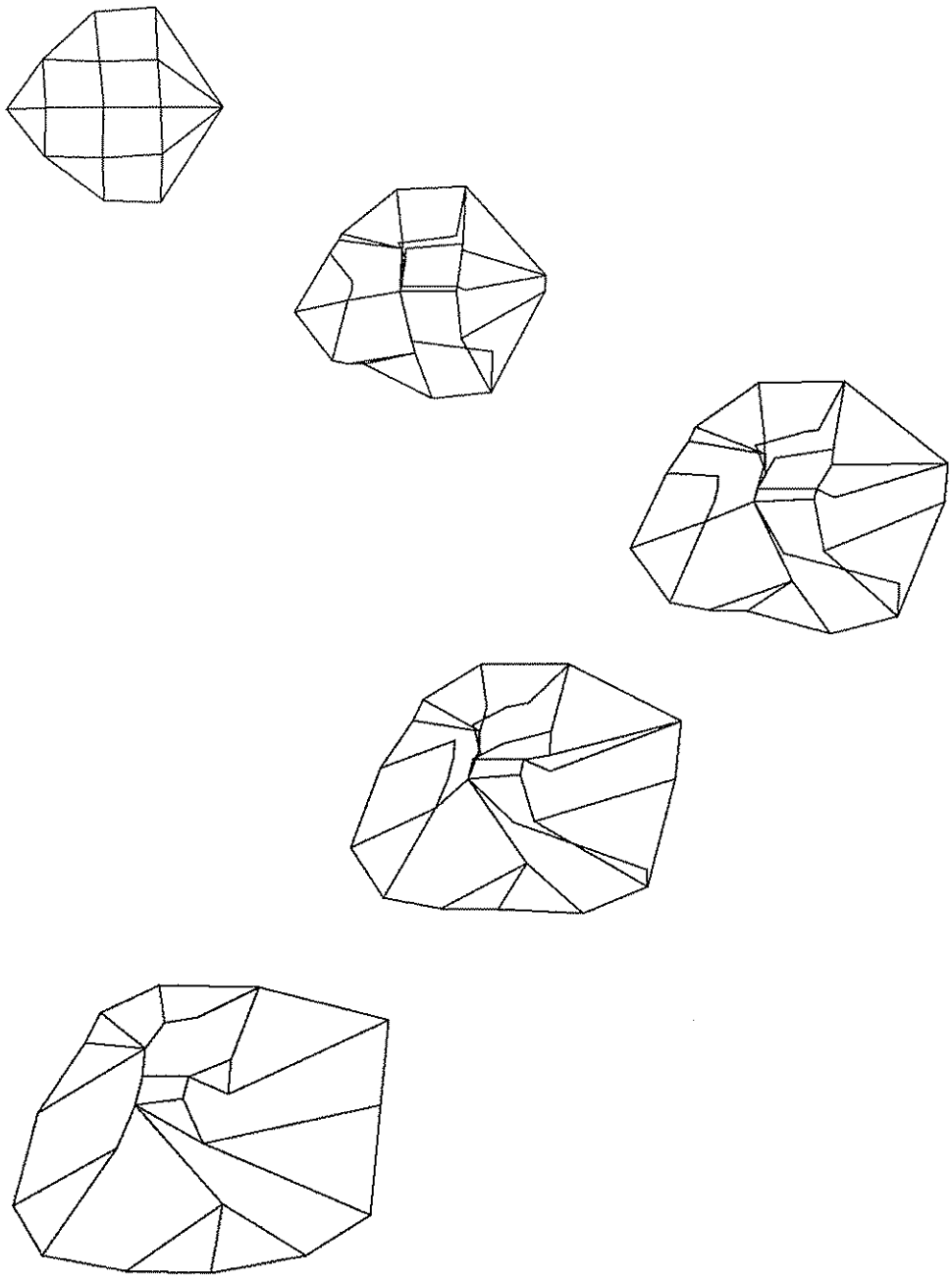


Figure 4.9: A situation of overlapping

Figure 4.10 illustrates a possible situation of overlapping.

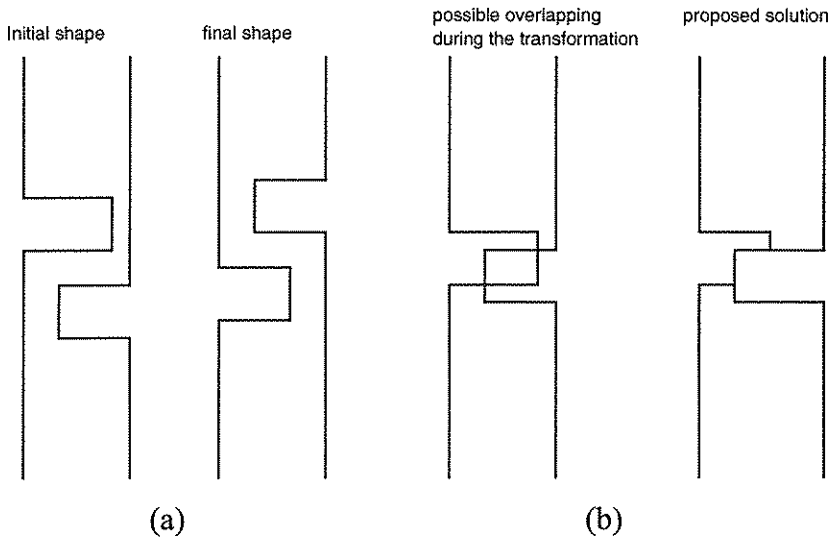


Figure 4.10: A possible situation of overlapping

To avoid this kind of undesirable effects, domains are considered being on different layers, so that a domain of less depth hides the intersecting parts of a higher depth domain. For each polyline with endpoints AB we consider the polygon formed by it and the sub-contour AB moving anti-clockwise (Figure 4.11). Depth values are assigned to these polygons equal to the index of their respective polyline and a depth buffer technique is employed to hide overlapping when it occurs (Figure 4.11b). This way an improved final animation sequence is achieved.

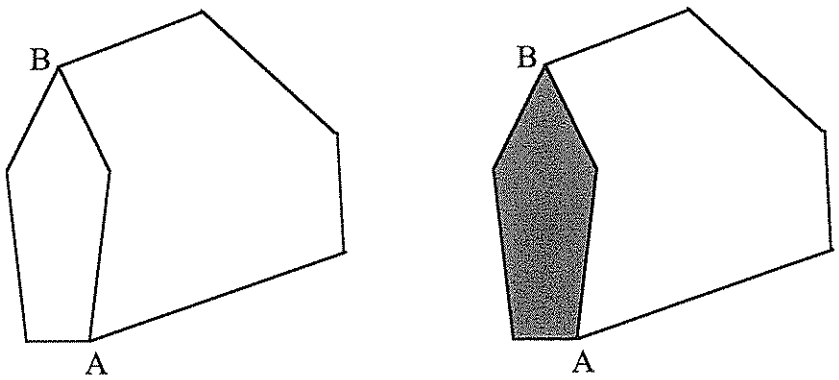


Figure 4.11: The polygon we consider for each polyline

Another problem occurs when the aliases of the same vertex correspond to different vertices of the same path. This can lead to overlapping during the metamorphosis or to shapes that lose their connectivity. To overcome this problem we allow a vertex to slide during the interpolation step from one position of the path to another not following the linear interpolation track. This keeps the connectivity of the original models and cancels overlapping.

Let us consider the metamorphosis of the shapes shown in Figure 4.12:

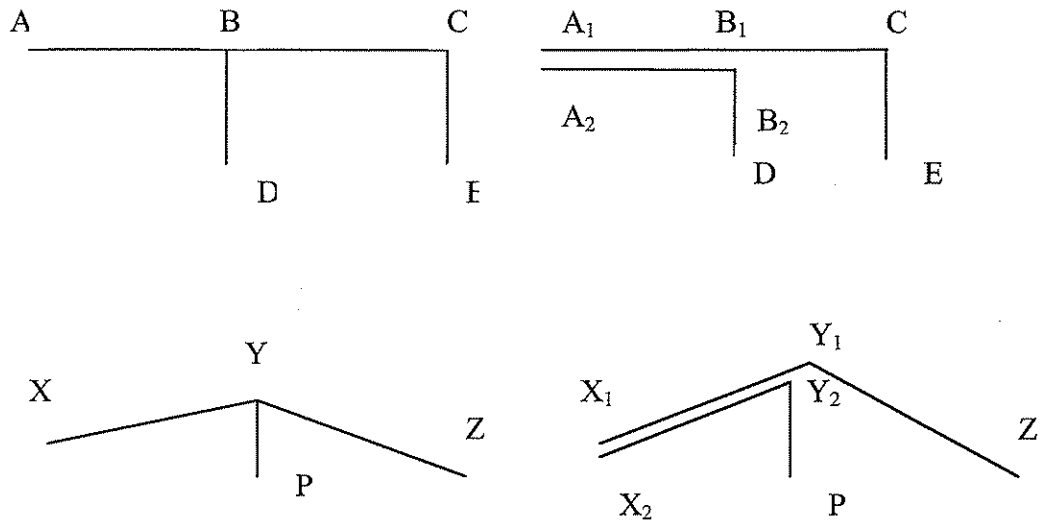


Figure 4.12: Two pairs of corresponding paths

Let's assume that the calculated correspondences are:

(A_1, X_1), (B_1, X_1), (C, Y_1), (E, Z) for the first pair of paths

(A_2, X_2), (B_2, Y_2), (D, P) for the second pair of paths

The resulting animation is presented in the following drawing (Figure 4.13).

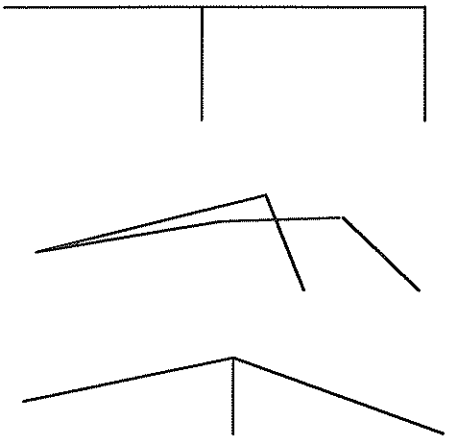


Figure 4.13: Two shapes and an intermediate frame of their blending

Again we assign priorities on the different paths so that a point slides from its initial to its final position across the path. This does not change the correspondence of the vertices but alters the trajectory of a vertex. Figure 4.14 presents the drawing of an intermediate shape of the resulting animation sequence.

Figure 4.14: The corrected intermediate frame

4.6 Conclusions

The examples that have been used so far show that the algorithm presented tends to associate regions of the two shapes, which look alike when the shapes consist of a small number of polygons. As the number of polygons increases the method seems prone to self-intersections. The heuristics proposed for correcting the self intersections when they occur are successful only when the self-intersections are not very severe.

This method does not seem easily extensible to 3D. Most 3D objects consist of hundreds if not thousands of polygons. Application of this method to non-trivial three-dimensional objects will result in a high number of self-intersections.

Another problem is the representation of the intermediate objects. The intermediate shapes of the 2D case were created from a set of paths that were giving the impression of a polygonal shape, this cannot be successful for the 3D case since in order of the intermediate objects to be rendered they need to be in polygonal representation.

Chapter 5

Metamorphosis of Three-Dimensional Polygonal Objects

5.1 Introduction

This chapter presents a method for the metamorphosis of three-dimensional polygonal objects. This method is an extension of the method developed in Chapter 3, for two-dimensional polygonal shapes and it applies to 3D dimensional objects in the case where there exists at least one axis such that the intersection of the object with any plane perpendicular to this axis results in at most one contour. The method applies to but is not limited to convex objects and objects that are star-shaped around an axis.

A common approach in shape metamorphosis involves a pair of objects represented as a collection of polygons. The vertices of the first polygonal object are then displaced, over time, to coincide in position with the corresponding vertices of the second object. Like the two-dimensional case there are two steps in this approach: first establishing a desirable vertex correspondence and then interpolating the co-ordinates of the corresponding vertices to get the intermediate objects.

The approach, presented in this chapter, is based on the fact that features of the two objects are assumed similar if small changes are needed to transform one to the other. The measure used to quantify these 'small' or 'big' changes is the energy needed to perform them, assuming that the objects are made of wire. This is the basis of Sederberg's and Greenwood's algorithm [SED92] already introduced. This approach

is extended to three-dimensional polygonal objects. The proposed solution uses a simplified representation of the original models.

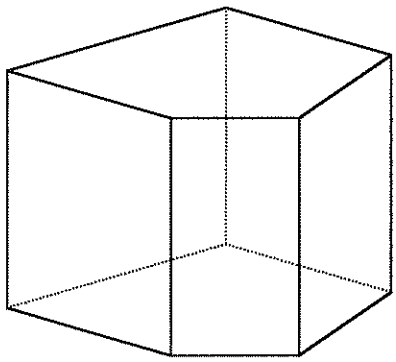
Kent in [KEN92], states the following criteria for judging a solution to a shape transformation problem:

1) Is face connectivity maintained for all intermediate shapes?
2) Are the intermediate shapes distorted?
3) What restrictions exist on the original models?

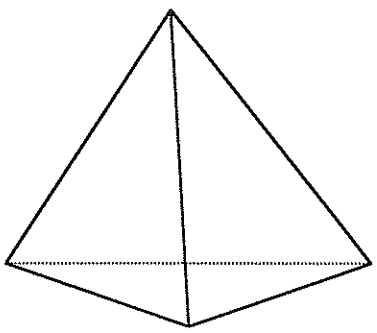
Some of the methods existing in the literature give very interesting results according to these criteria, but little attention has been given for correspondence determination in the semantic level. The method proposed in this chapter tries to recognise similar features of the two objects that are morphed and maintain those features throughout the blend.

5.2Approximation of 3D objects with a set of parallel planar contours.

Let's consider the problem of morphing two three-dimensional objects, made up by a set of polygons (Figure 5.1). In most practical cases the complexities of these objects make it difficult to find an exact solution with respect to energy minimisation. Therefore, a simplified way of representing the objects is required.



Object1



Object2

Figure 5.1: Two polygonal objects.

Consider the intersection of an object with offsets of a given plane. This results in a set of planar contours on parallel planes (Figure 5.2). Depending on the number of contours, it is possible to reconstruct the original model to a reasonable precision using one of the existing algorithms for surface reconstruction from planar contours.

Levin in [LEV86] addressed the problem of the reconstruction of an n-dimensional model from a sequence of its (n-1)-dimensional cross-sections. It presented and analysed the distance field interpolation method.

A procedure is said to be of order k if for data values taken from a C^j-continuous function, the error in the approximation is $O(h^{j+1})$ $(0 \leq j \leq k)$. For equidistant cross-section levels $x_i = x_1 + (i-1)h$ $(1 \leq i \leq n)$

He proves that:
Theorem 5.1 Let B be a closed 3D domain whose boundary _B is composed of a finite number of surfaces which are C^k-continuous and mutually disconnected and let $x = x_i$ $(1 <= i <= l)$ be the planes which are locally tangent to _B. Let δ be any positive constant. For each $h>0$ consider any point (x^h, y_1^h, y_2^h) with:

$$\min_{1 \leq i \leq l} | x^h - x_i | ? \delta$$

which is a point of disagreement between B and the reconstructed domain. Then

$dist[(x^h, y_1^h, y_2^h), \_B] = O(h^{k+1})$ as h_0

Hence for h small enough (x^h, y_1^h, y_2^h) is close enough to _B so that $d(x; y_1, y_2)$ is C^k continuous as of function of x in $(x^h - mh, x^h + mh)$

For the time being we will consider situations where each intersection of the object results in a single contour.

Let us consider the problem of morphing these two sets of planar contours with least work. Since the contours do not intersect they can be ordered according to their geometric position in space with respect to a perpendicular axis to the intersecting planes. The position of a planar contour in this ordering will be used as an index.

Suppose that m contours, not necessarily equidistant are used to describe the first object and similarly n contours are used for the second object. It is possible to order the contours of the first object from 1 to m and of the second object from 1 to n (Figure 5.2).

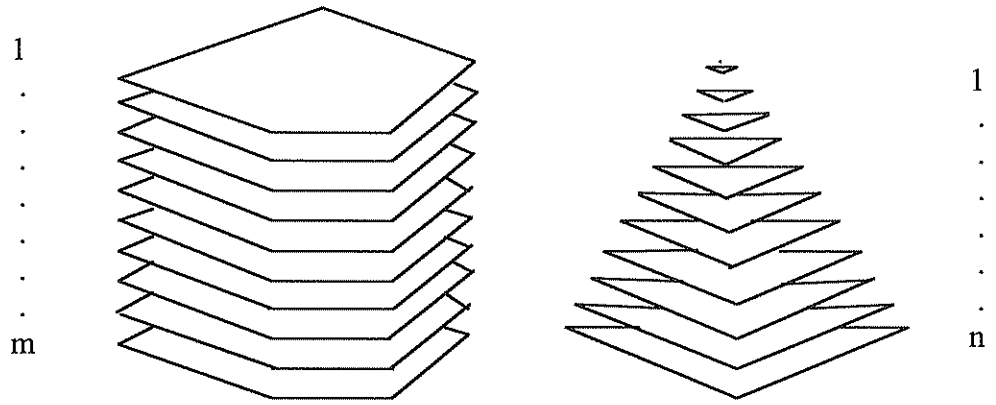


Figure 5.2: The objects approximated with a set of parallel planar contours

The next section presents a method for determining how the contours of the first object should correspond to the contours of the second object so that the object transformation is accomplished with least work. Sederberg's and Greenwood's algorithm will be used to determine the minimum energy needed to transform one contour to another and the correspondence of the contour vertices that results in this minimum energy transformation.

A *valid solution* to the contour correspondence problem described above needs to satisfy the following:

for every contour of the first object there is one or more corresponding contour(s) of the second object, and this leads to an animation sequence without crossovers.

Crossovers occur when one contour crosses another one, or two contours are interchanged. To avoid crossovers it is expected that if the i'th contour of the first object corresponds to the j'th contour of the second object then there is no contour of the first object with index greater than i corresponding to a contour of the second object with index less than j and similarly there is no contour of the first object with index less than i corresponding to a contour of the second object with index greater than j.

We assume that the extreme contours correspond together, so that the first and m'th contours of the first object correspond to the first and n'th contours of the second object respectively.

5.3The contour correspondence problem is expressed as the shortest path problem in a directed graph.

The correspondence problem of the contours bears a lot of similarities to the correspondence problem of the polylines described in Chapter 3.

Initially a directed graph G is constructed as described in section 3.4. This graph contains mn vertices denoted [i, j] i=1,...,n and j=1,...,m. The vertex [i, j] represents a correspondence of the i'th contour of the first object with the j'th contour of the second object. We assign lengths to the edges of graph G, so that edge ([i, j], [k, h]) has a length equal to the energy needed to transform the k'th contour of the first object to the h'th contour of the second object (as given by Sederberg's and Greenwood's algorithm).

As the problem of the correspondence of the contours was stated, it is equivalent to the problem of the correspondence of the polylines discussed in chapter 3. Using similar arguments as the ones presented in section 3.4 we can demonstrate that there is one-to-one mapping between the set of the paths in G from vertex s to vertex t and the set of all valid solutions to the correspondence problem of the contours.

Furthermore we can show that the shortest of those paths represents the solution of the correspondence problem of the contours with least energy.
The implementation of the method proceeds as follows: Initially Sederberg's and Greenwood's method is used for the calculation of the length of the edges of graph G. When the length of all the edges of G has been calculated it is possible to compute the shortest path from vertex [1, 1] to vertex [m, n]. The method used is described in section 3.4.

Sederberg's and Greenwood's method needs a pair of corresponding anchor points in order to be applied on a pair of contours. Therefore we need an anchor vertex on each contour of the shapes. These can be either user-defined or automatically computed. In all the examples presented in this chapter these anchor points were automatically computed by the intersection of the contours with a plane containing the axis used for slicing.

Having the correspondence of the contours and their vertices we can interpolate their positions and compute the intermediate contour sets for the desired number of frames. Then a surface reconstruction method from these sets of planar contours gives us the intermediate object for each frame of the animation sequence.

5.4 Examples

Figure 5.3 presents the objects that will be used for the animations of this chapter.
Table 5.1 presents the number of polygons for each object.

Object	**Number of polygons**
Banana	256
Glove	2191
Teapot	2518
Plane	686
Face	4357
Head	1882

Table 5.1: Number of polygons for each object

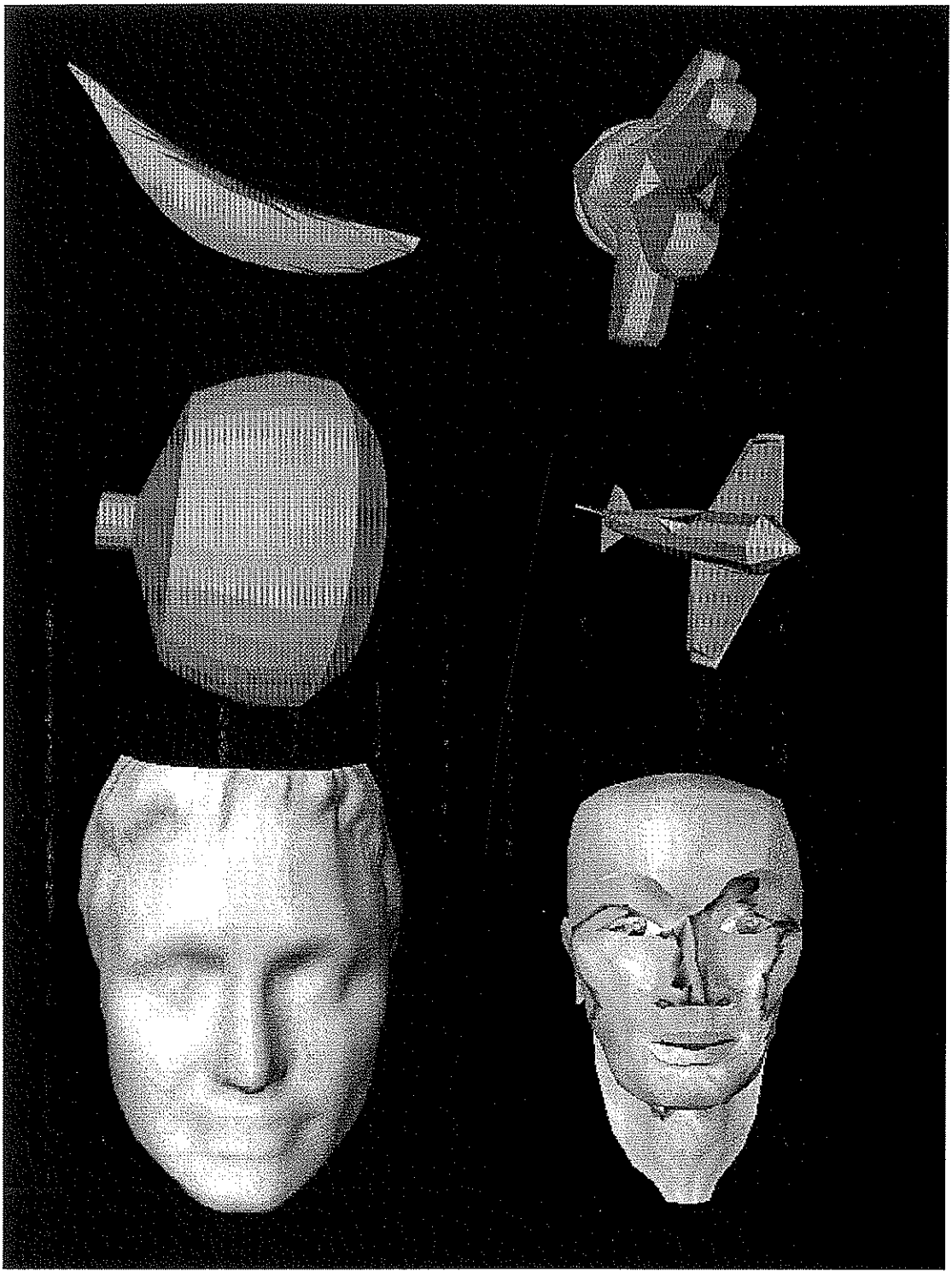


Figure 5.3: The original objects 'banana', 'glove', 'teapot', 'plane', 'face', 'head'

Figures 5.4 and 5.5 present two animation sequences obtained by the application of this method. In Figure 5.4 one a pot is transformed to an aeroplane while in Figure 5.5 a banana is transformed to a glove. In both Figures 5.4, 5.5 the first column contains sets of contours that approximate the object during the metamorphosis. In the first and last line of this column there is a set of contours that approximate the initial and the final object respectively. These contours are obtained by the intersection of the original object (Figure 5.3) with a set of equidistant parallel planar planes. The orientation and the number of planes are user-defined and this is a way that the user can influence the resulting animation sequence.

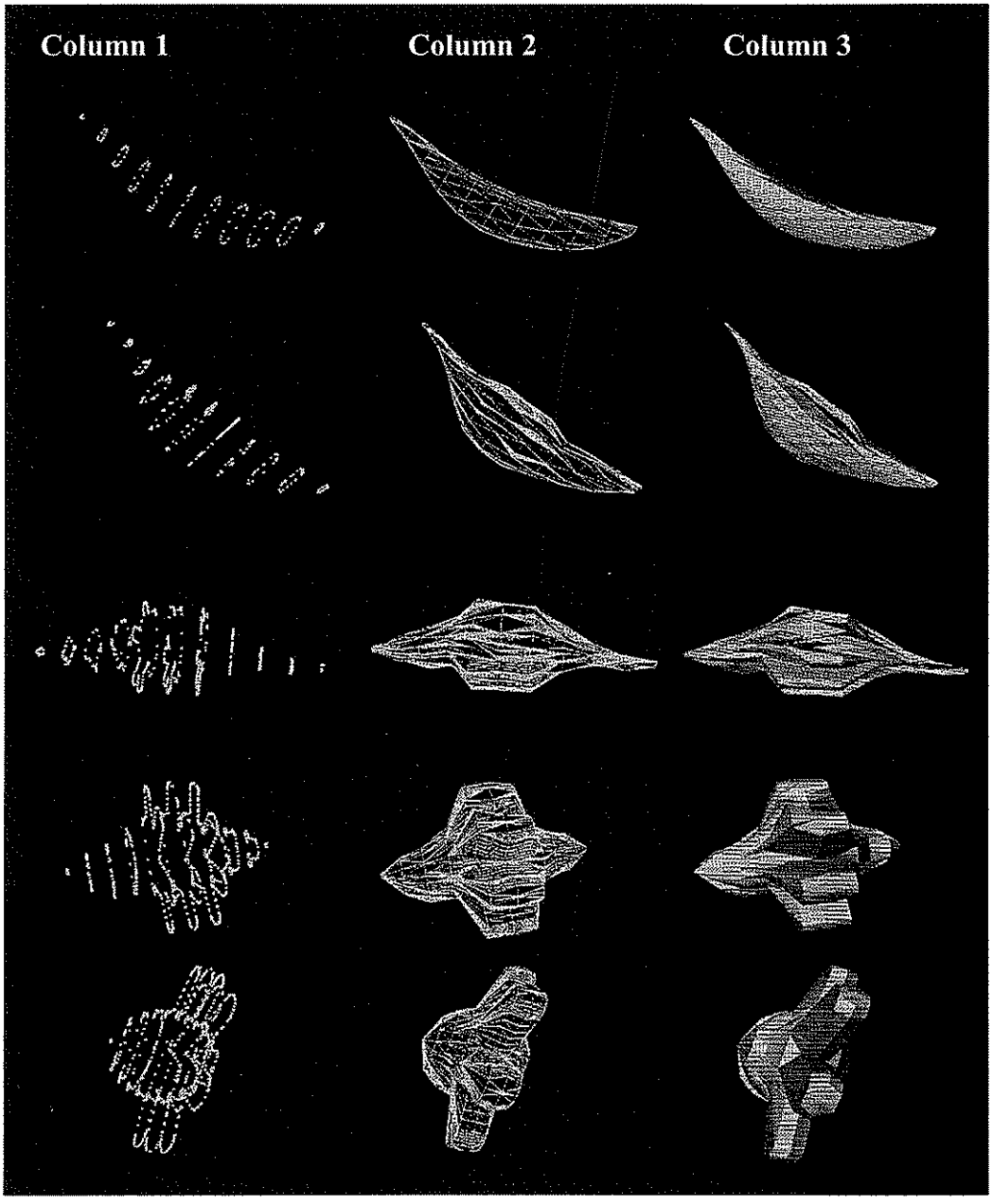


Figure 5.4: Metamorphosis from 'banana' to 'glove'

The second column of Figures 5.4, 5.5 displays the reconstructed wireframe objects from the sets of parallel planar contours of column 1. The third column of Figures 5.4 and 5.5 contains the same objects of the second column but rendered. These examples were rendered using faceted shading and neutral colours to better illustrate the topological structure of the intermediate models.

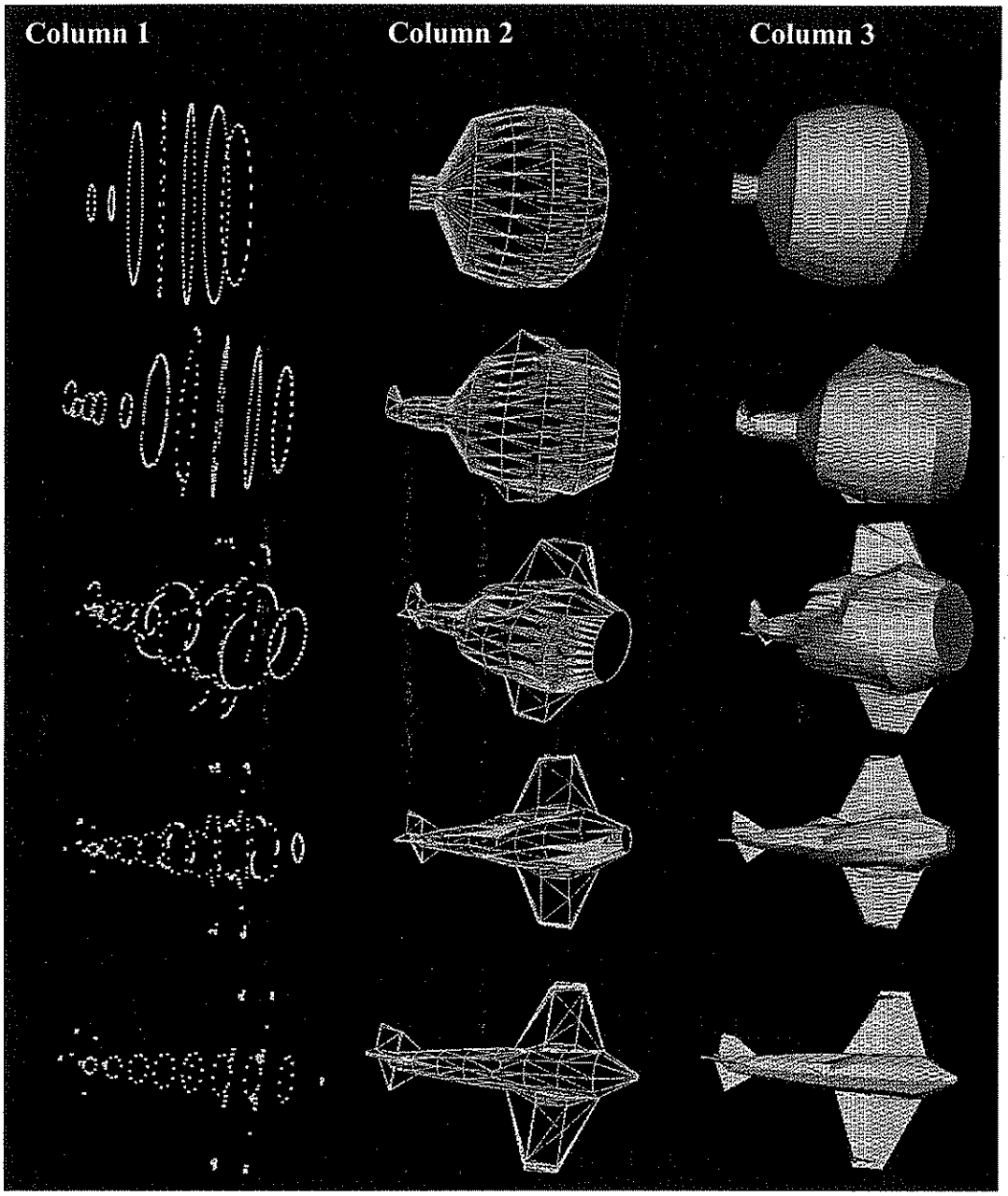


Figure 5.5: Metamorphosis from 'teapot' to 'biplane'

In the first example (Figure 5.4), the two terminal objects are approximated by a different number of contours (seven for the first object and ten for the second) while in the second example (Figure 5.5) both terminal objects are approximated by ten contours.

By applying the metamorphosis method to these two sets of contours it is possible to find a correspondence for the contours and their vertices. Once the correspondence of the contours and their vertices has been established, the intermediate contour sets are computed by interpolating between each pair of corresponding vertex locations. The preservation of the contour ordering and the usage of Sederberg's and Greenwood's method for determining the vertex correspondence minimises the situations where any kind of self-intersection occurs during the animation sequence.

The objects that are used in these examples have very dissimilar shapes, none of them is convex and the objects 'banana', 'glove' are not star-shaped. These examples demonstrate clearly that the proposed method can be applied to generic polygonal objects. A limitation of the proposed method is that it can not handle situations where the intersection of the object and a plane results into multiple contours during the slicing step. Once we have found the correspondence of the contours and have calculated the intermediate contour sets we apply an algorithm for surface reconstruction from planar contours [FUC82] to get the intermediate objects. This algorithm calculates a triangulated surface between every two successive contours. Since all faces in the resulting intermediate objects are triangles they remain planar during the animation.

Since the method uses both topological and geometric information of the original models, it avoids undesirable effects such as faces flying apart during the metamorphosis or uneven, distorted transformations (i.e. large parts of one object changing to small parts of another). Both examples display lack of distortion of the intermediate objects.

Even though this method produces even and not distorted transformations it allows parts of an object to extend or compress so that they can be mapped to similar parts of

the other object. Let's consider the contour sets of Figure 5.6, when object A is morphed to object B parts of the shape will extend through the creation of new contours. In Figure 5.4 the two terminal objects are approximated by a different number of contours. The lid of the teapot, which is quite similar with the rear body of the plane, is extended through the generation of three more contours.

When object C of Figure 5.6 is morphed to object A part of the object will be deflated or squeezed.

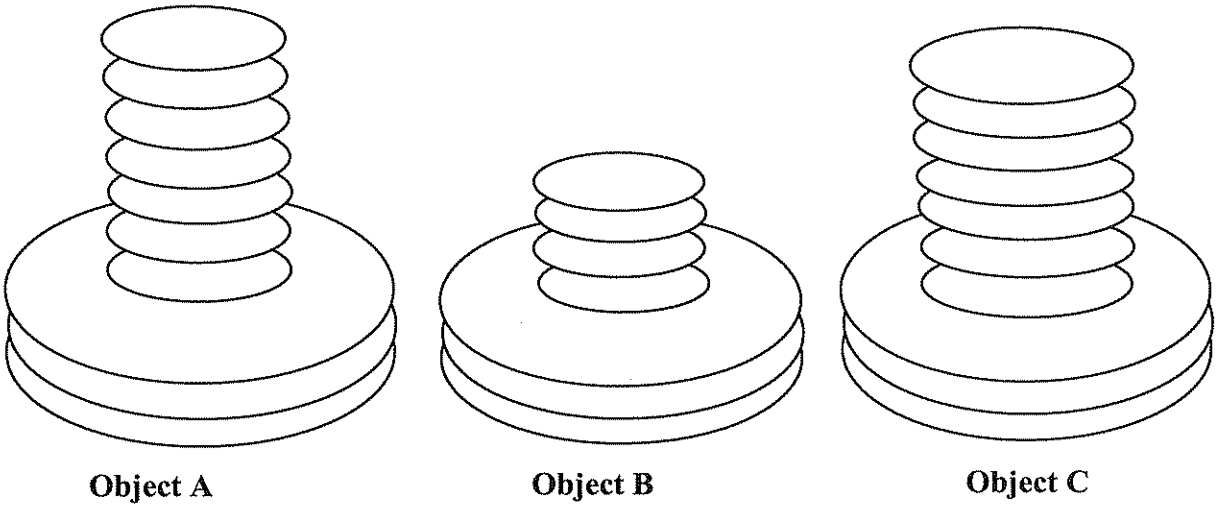


Figure 5.6: Morphing between three objects

Other transformations that are possible are scaling either locally or for the whole object. As successive contours deform it is possible to have effects that appear as local or global deformations on the 3D level.

The anchor points on the contours can be used to provide a rotational component to the animation (Figure 5.7). this can be provided by the user adding an additional overhead in the amount of user interaction required but also giving the possibility to the animator to guide the animation to transformations he requires like part of the object twisting.

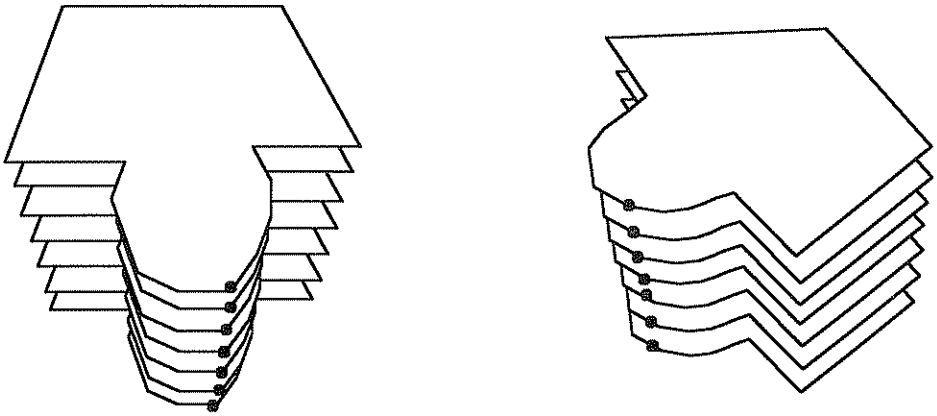


Figure 5.7: Adding a rotational effect to the metamorphosis

The described method allows some user control over the transformation through mechanisms such as selecting the orientation of the original objects, defining the axis for the slicing of the objects and defining the number of slices for each object. The following figure (Figure 5.8) displays that the original orientation of the initial objects influences the resulting animation sequence.

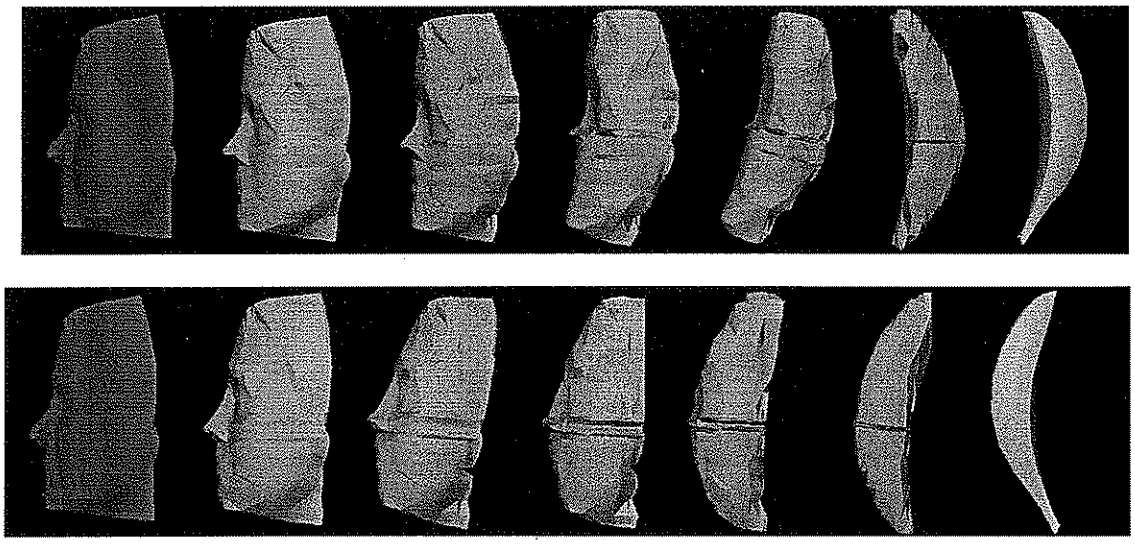


Figure 5.8: Two different animations depending on the orientation of the original objects

When there is more than one possible axis for slicing the object there is an obvious question of what the preferable axis should be. Obviously this mainly depends on what the animator wants to achieve but there are some general based on the work of this Thesis.

An axis with a specific semantics (such as from the front to the back of a car) usually gives good results probably because the viewer can associate more with changes happening across this axis.

An axis that minimises the relative displacement of the contours on the parallel planes. The less the relative displacement of the contours the closer they are and the more precise is the reconstruction of the object from the planar contours.

Choosing the axis as the longest dimension of the object seems giving better results

In general, an axis which the object is symmetric around is a good candidate to be used for slicing.

5.5 Methods for selecting the number and spacing of slices

It is clear that in order to represent an object adequately the number of slices and their spacing depends on the complexity of the original. For example a cylinder that is sliced across its axis can be described (without loss of information necessary to describe the object) by just two contours; (ie the base and the top of the cylinder). On the other hand, a more complex shape having fine details and drastic shape changes, along the axis used for slicing, would need many more slices in order to be represented with a reasonable precision.

Several approaches have been followed for determining the number and the position of slices. The first one assumes that slices are equally spaced across the axis of slicing. The number of contours used is user defined and depends on the complexities

of the original models. This was the method used for the examples presented in Figures 5.4 and 5.5. This approach demands minimum user interaction but does not guarantee the preservation of the important characteristics of the two objects since drastic changes in object's shape occurring between two successive contours will not be present in the intermediate objects. Consequently this approach demands a very large number of slices to guarantee that no important characteristics of the two shapes are lost. This gives an additional overhead not only to the time needed to slice, transform, interpolate and reconstruct the objects but also creates intermediate objects of high complexity.

A second approach allows the user to break the objects into parts across the axis used for slicing and define the number of contours for each part (Figure 5.7). This approach gives a higher level of control over the transformation with the drawback of an increased level of user interaction.

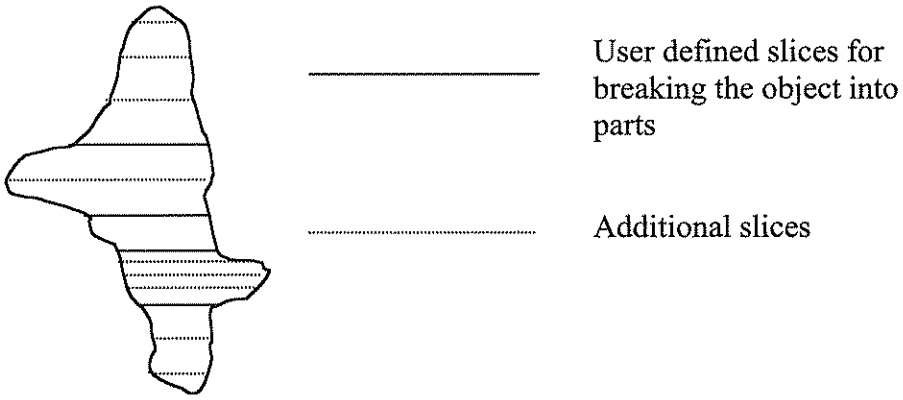


Figure 5.9: Breaking the object into parts across the axis used for slicing

Finally in the next section we present a fully automated method for finding out the number and the position of slices.

5.6 An automated contour slicing algorithm

This section presents a method that automatically calculates the characteristic contours necessary to describe an object. The method operates by examining a number of densely spaced contours across the axis of slicing, but keeping only a

contour if it differs considerably from the previously selected contour. But how do we decide that two contours differ enough and on what basis? One obvious answer is that the measure to quantify the amount of difference between two contours is the energy needed to change one to the other as defined by Sederberg's and Greenwood's algorithm. This is a naïve approach as we can see in Figure 5.10. In this case all the slices that are calculated are identical (Figure 5.10b) and only the two extreme contours will be used for the approximation of the object. This will result to the reconstructed shape of Figure 5.10c which differs considerably from the original.

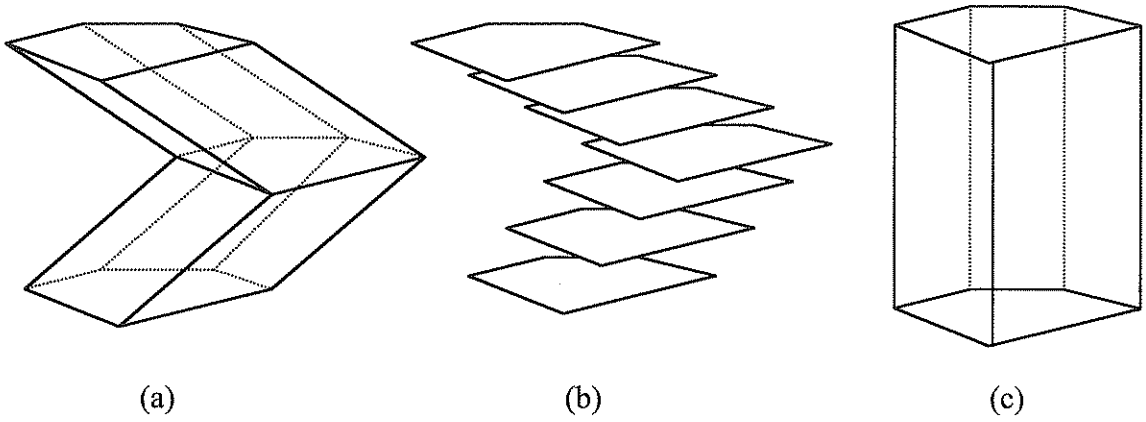


Figure 5.10: Possible anomaly on the contour selection

This happens because Sederberg's and Greenwood's algorithm does not take into consideration the relative displacement of the two contours. To overcome this problem we modified the original method by adding an energy term related to the angle that the centres of gravity of three successive contours form. We assume that the centres of gravity of successive contours are connected by the same elastic material that the contours are made of. The method for selecting a number of characteristic contours to describe the object proceeds as follows: Moving across the axis used for slicing we calculate n uniformly spaced contours. The two extreme contours 1, n are included by definition to the selected contours. The contours 2,..,n-1 are provisional since only the most characteristic of them will be selected to approximate the object. For each provisional contour we calculate two energy terms: the first term is the energy needed to morph it to the previous selected contour and the second term is the energy needed to align the 'last selected contour', the 'current provisional contour' and the 'next provisional contour' (for the Figure 5.11, the

second term is the energy needed to bend angle _ to π). If the sum of these two energy terms is greater than a user defined threshold the current contour is included in the set of the contours that will be used to approximate the object and it becomes the 'last selected contour' otherwise it is discarded. Following this procedure for all provisional contours, from contour 2 to contour n-1, we obtain a set of selected parallel planar contours that will be used to approximate the object. In Figure 5.12 we present the algorithm used for the selection of these characteristic contours.

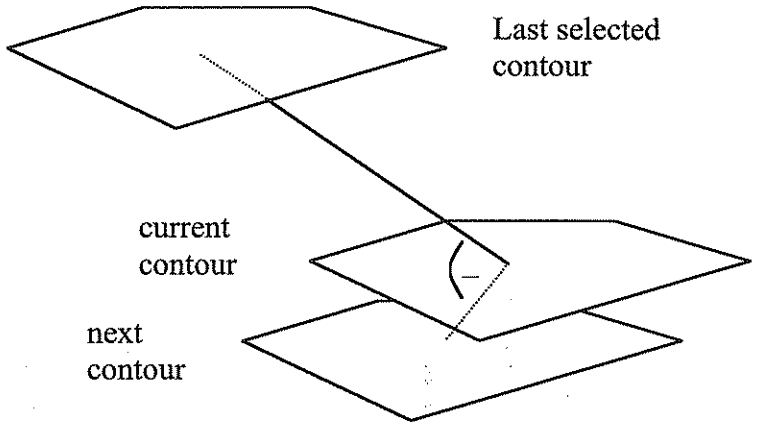


Figure 5.11: Angle formed by three successive contours

Algorithm 5.1 Calculating the most characteristic contours

Input: A set of contours *Contour[N]* created by the intersection of the object with N parallel planes, a user defined threshold.

Output: A set of contours *List_of _Contours* containing the most characteristic contours of *Contour[N].*

```
Last_Selected ← Contour(1)
Current ← Contour(2)
for j ← 2 to N-1 do
begin
        Next ← Contour(j+1);
        Energy ← Energy_to_transform(Last_Selected, Current)+
        +Energy_to_straighten(center(Last_Selected), center(Current), center(Next));
        if Energy>Threshold then
        begin
                ADD(List_of _Contours, Current);
```

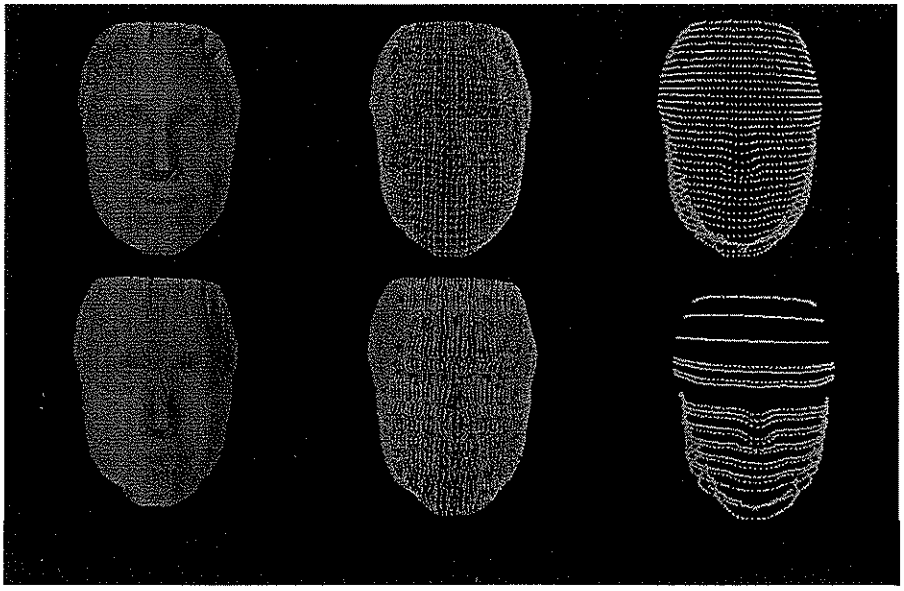
```
        Last_Selected ℜ Current;
    end
    Current ℜ Next
end
ADD(List_of _Contours, Current)
```

Figure 5.12: Algorithm for the selection of contours

When a branching occurs we consider that the shape change is substantial by definition.

Better results are obtained by using a high sampling rate (testing a big number of slices) so that we can guarantee that no important object characteristics are lost.

Figures 5.13, 5.14 present some examples of slicing obtained by the application of this method. The first Figure 5.13 presents the approximation of the object 'face', while Figure 5.14 display a sequence of crude to more precise approximations of the same object depending on the threshold used. In Figure 5.14 the first column displays the selected contours, the second the reconstructed wireframe model and the third the fully rendered object. It is easy to notice that the selected contours are more densely spaced where the changes are more rapid.



Initial Object

Approximated with 23 contours

Figure 5.13: An example of an object approximated by a set of contours

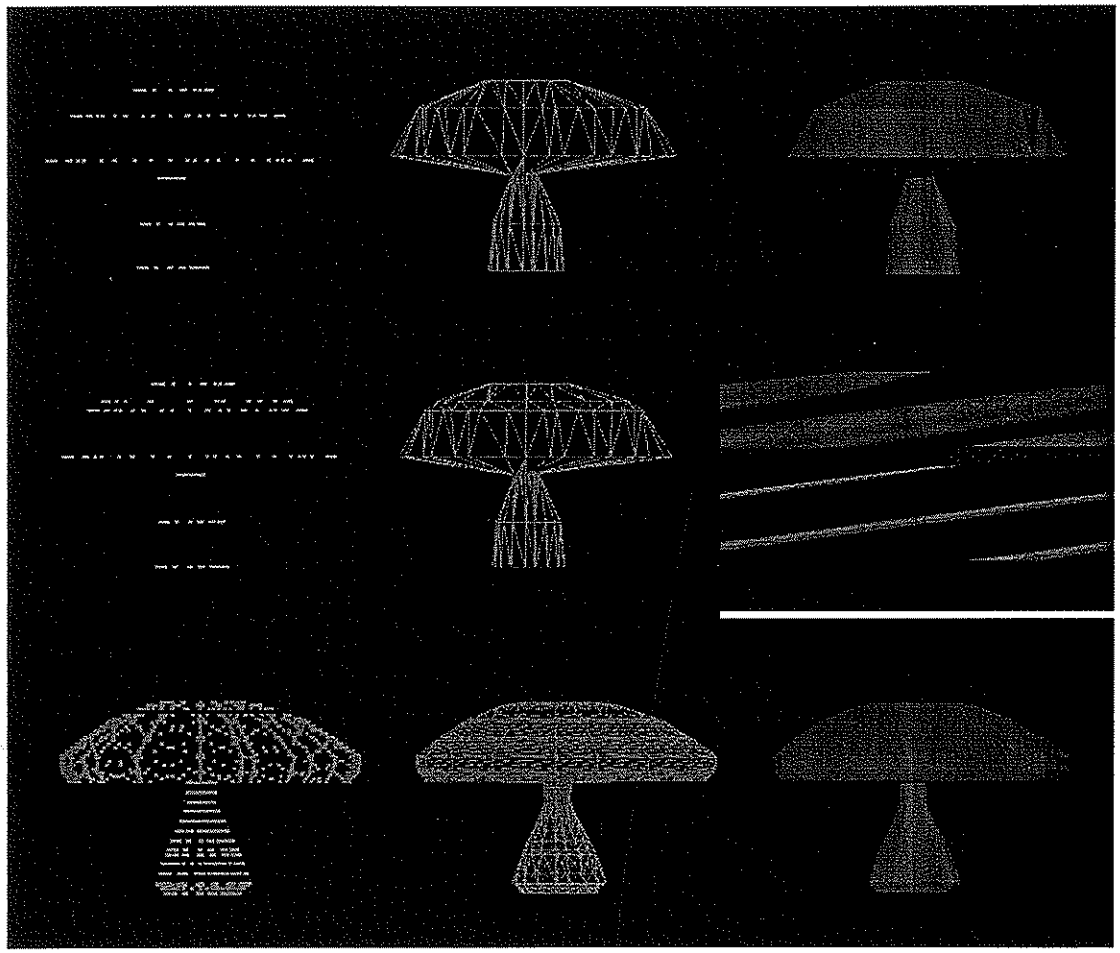


Figure 5.14: The same object represented in different levels of precision

The contour representation of a 3D polygonal object seems to share the same problems as the related problem of how to reconstruct a surface from a set of parallel planar contours. When the cross-sections are not dense they may exhibit significant differences in the geometry and the topology of the boundary contours making the reconstruction difficult and imprecise. When branching occurs ambiguities arise about how the successive contours should tile. But the simplification method has a certain advantage over the surface reconstruction problem, when we simplify the objects we also have the original models so even if there exist situations of multiple branching we do have the information as to how these multiple branches connect to each other. Therefore a possible extension of the simplified representation would be to incorporate this connectivity information to the contour representation, which will

allow this method to be applied in the simplification and subsequent reconstruction of general 3D polygonal objects.

Figure 5.15 presents an animation from the object 'face' approximated by 23 contours to the object 'head' approximated by 29 contours. These two set of contours were obtained by applying the automated method for the selection of contours. We can notice that the similar characteristics of the two objects correspond to each other. The neck of the first object is extended through the generation of additional contours.

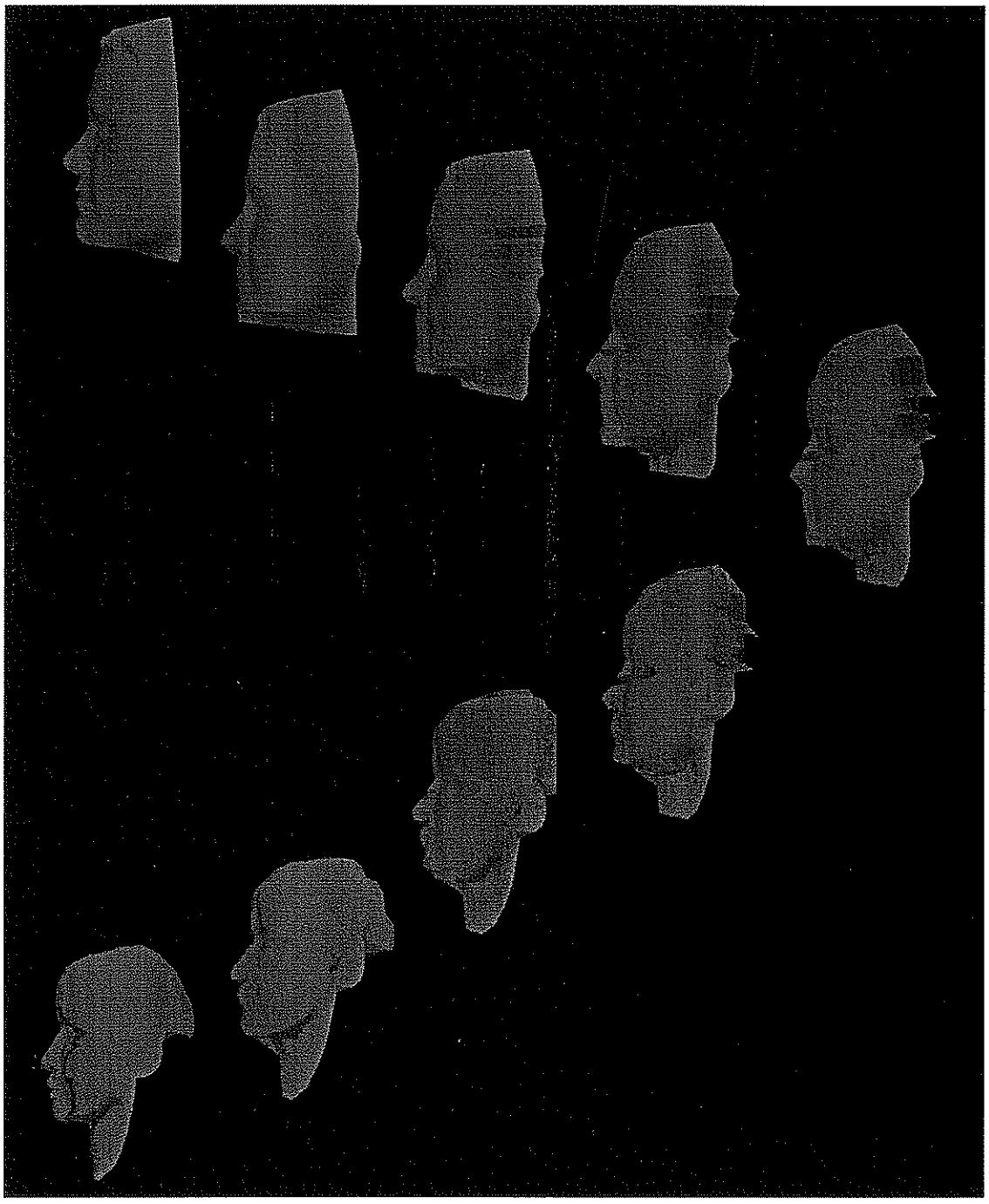


Figure 5.15: An animation example

5.7Using the method for the calculation of the contours as a method for shape simplification

The above method has been implemented as an aid of the computer metamorphosis method for determining the number and the position of slices. Soon it has become apparent that the same methodology could be applied in reducing the complexity (number of vertices) required to represent a polygonal object, say for rapid prototyping. In these situations we can afford to lose some precision of the representation of the object and reduce its modelling size. One example of this kind of application is compression and transmission of polygonal objects with variable precision. In this case the information that needs to be transmitted is simply a set of contours. The approximated model can be reconstructed at the receiver, using one of the existing surface reconstruction algorithms or it can be visualised be applying volume rendering techniques.

The modelling size of the approximation can be reduced further by applying a simplification technique on 2D for each selected contour. Figure 5.16 presents the pseudocode we used for simplifying each 2D contour. This method effectively keeps a vertex of the contour if the angle formed on it is sufficiently different than π. Figure 5.17 presents some approximations of the object 'face'. The first column contains the reconstructed object using a set of selected contours. The second and third columns further simplify the original object by applying the algorithm of Figure 5.16 on each contour. The Objects of the second column were obtained by using a threshold angle of 12^0 while the objects of the third column were obtained by using a threshold angle of 18^0.

Algorithm 5.2 Simplyfing a planar contour
Input: A contour , a user defined angular threshold.
Output: A set of points approximating the original contour List_of _Vertices

```
Last_Selected ← Vertex(1)
ADD(List_of _Vertices, Last_Selected)

Current ← Vertex(2)

for j ← 2 to Number_of_vertices do
begin
        Next ← Vertex(j+1);

        if |pi-Angle(Last_selected, Current, next)|<Threshold then
        begin
                ADD(List_of _Vertices, Current);
                Last_Selected ← Current;
        end
        Current ← Next
end
ADD(List_of _Vertices, Current)
```

Figure 5.16: Algorithm for the selection of vertices

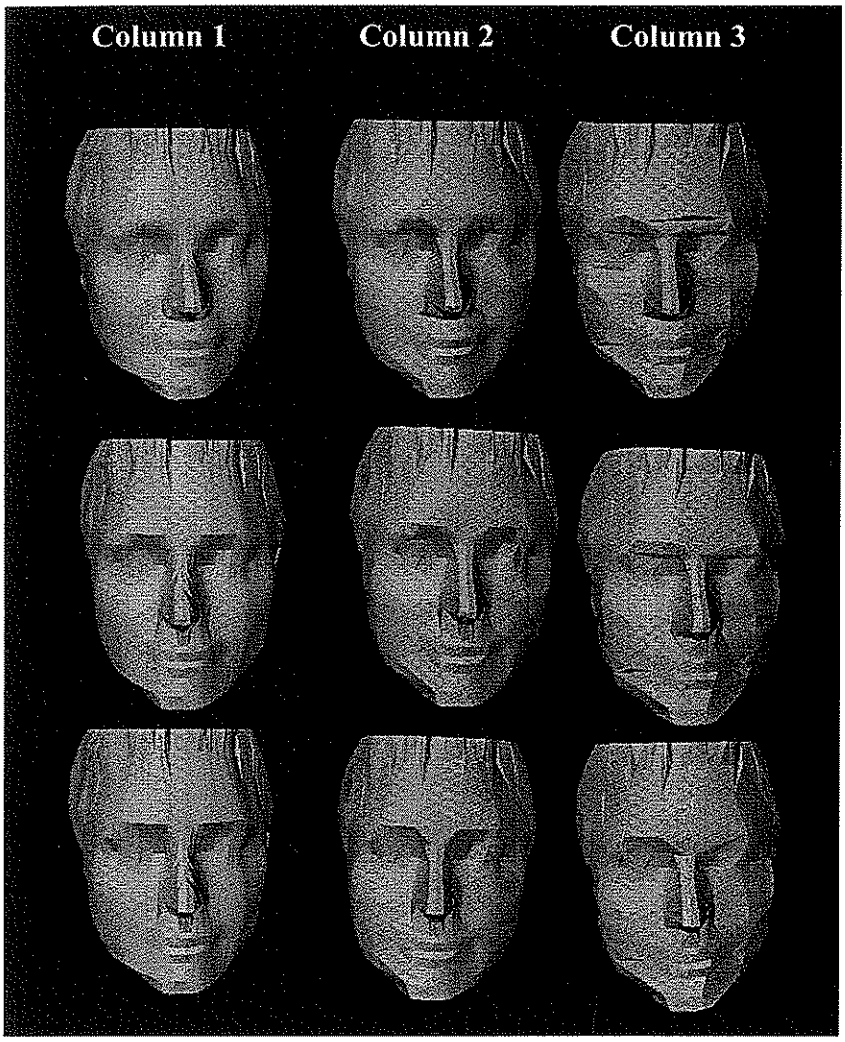


Figure 5.17: The object 'face' represented in a different degree of precision

The original object 'face' (Figure 5.3) had a modelling size of 173080 bytes. Table 5.2 contains the modelling size of the approximations presented in Figure 5.15. As we can notice the objects in column 2 of Figure 5.13 achieve a good approximation of the original object with only a small fraction of the its modelling size.

Size in bytes	**23 contours**	**20 contours**	**18 contours**
Angle 0°	83509 bytes	72018 bytes	64352 bytes
Angle 12°	26273 bytes	23436 bytes	21122 bytes
Angle 18°	16928 bytes	15302 bytes	14064 bytes

Table 5.2: The size in bytes of the files storing the objects of Figure 5.13.

5.8Application of the 3D metamorphosis method to general polygonal objects

The method that was described assumes that an intersection of a chosen plane with the object results in single contours. In general it is possible that the intersection results in two or more contours. Let us consider the situation shown in Figure 5.18, here the intersection of the first object with the P plane results in two contours.

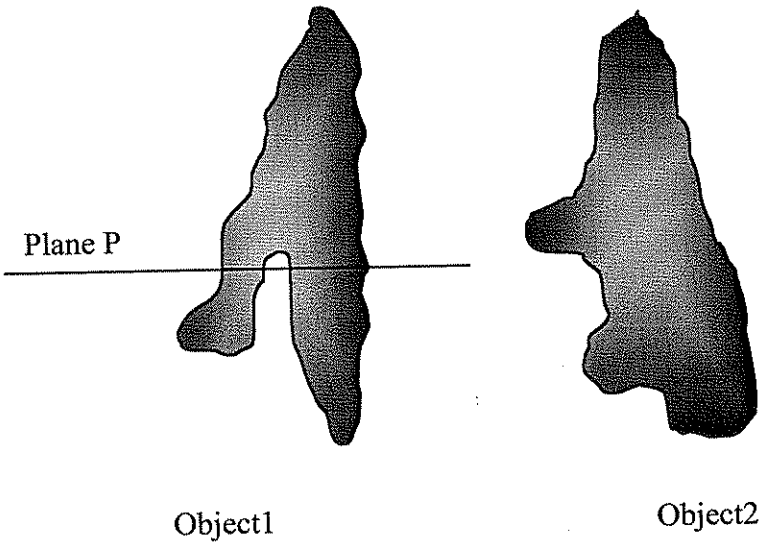


Figure 5.18: A view of two 3D objects

One of the possible ways for the metamorphosis to be achieved is shown in the Figure 5.19. Part A of the first object can be transformed to part X of the second object with part B collapsing, part D transforming to part Z and part C transforming to part Y.

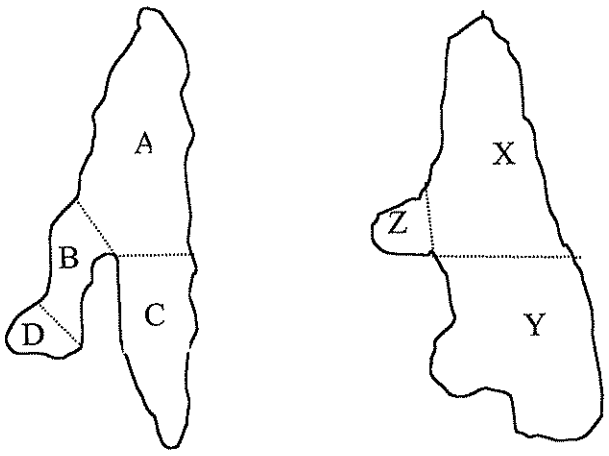


Figure 5.19: Morphing of shapes 'solution' 1

Another possible way for the metamorphosis is part A transforming to part X, part B transforming to part Z, part D collapsing and part C transforming to part Y (Figure 5.20).

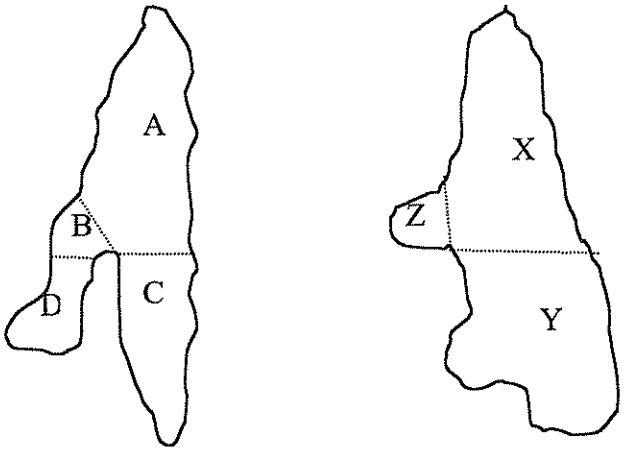


Figure 5.20: Morphing of shapes 'solution' 2

Another possible way for the transformation is part A transforming to part X and parts B, C joining together to form part Y (Figure 5.21).

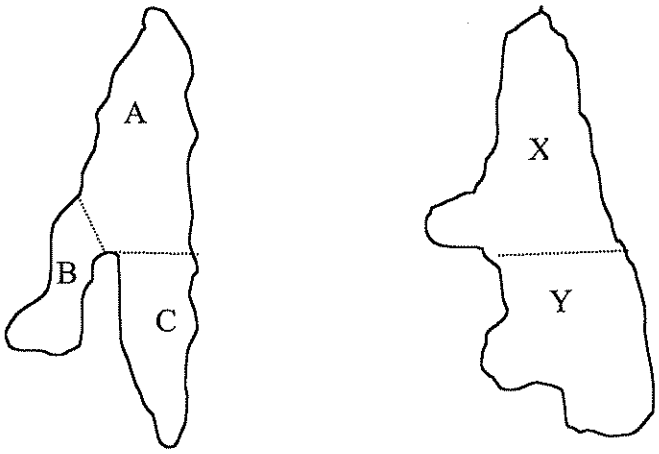


Figure 5.21: Morphing of shapes 'solution' 3

It is obvious that these are not the only ways to achieve the metamorphosis of the first object to the second but nevertheless these examples serve the purpose of demonstrating the fact that depending on the specific situation there can be more than one acceptable ways of the metamorphosis. Which one is chosen is based on the

specific situation. Therefore it appears that granting the user the possibility to guide the animation will result into more positive results. This can be accomplished by allowing the user to divide the objects into parts, define correspondences for these parts and provide an axis for the slicing of each part. In Figure 5.22 we see the necessary definitions for a metamorphosis according to the first example.

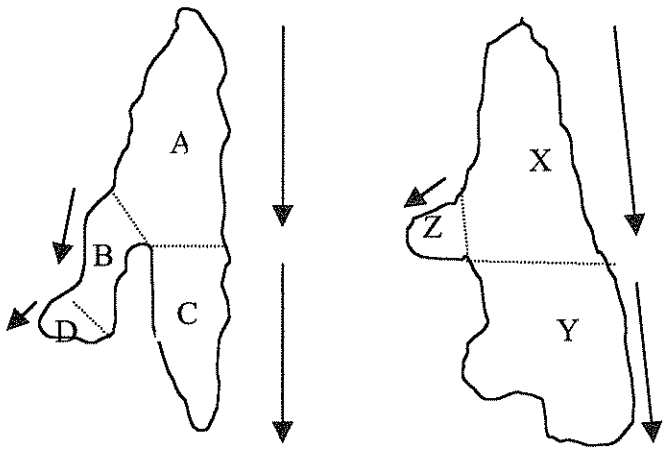


Figure 5.22: Breaking the objects into parts

5.9 Conclusions

The method presented in this chapter tackles the correspondence determination problem and it applies to three-dimensional polygonal objects which have the property that there exist an axis so that the intersection of the objects with any plane perpendicular to this axis does not result to more than one contour. The set of objects with this property is a superset of the objects that are star-shaped around an axis.

The examples that have been used so far show that the algorithm tends to associate regions of the two objects that look alike. The preservation of the contour ordering and the usage of Sederberg's and Greenwood's method for determining the vertex correspondence minimises the situations where any kind of self intersection occurs. The described method allows some user control over the transformation through mechanisms such as selecting the axis for the slicing of the objects, defining the number of slices for each object and defining the anchor points on the contours. Further control of the transformation can be granted by allowing the user to divide the

objects into parts and define corresponding parts. This can allow the application of a similar methodology to more general polygonal objects, where the intersection of the object and a plane results in multiple contours. The application of this method can lead to animations that combine a variety of transformations, like scaling, rotation, deformation and extension or compression of the object.

Chapter 6

Performance and Complexity of the 3D Metamorphosis Algorithm

6.1 Introduction

In this chapter we evaluate the three-dimensional metamorphosis method described in Chapter 5 according to a number of different criteria. These criteria include the estimation of the time complexity for the different steps of the method and the estimation of the number of vertices and polygons of the intermediate objects. These theoretical estimations are in most cases accompanied by experimental measurements obtained by the application of this method to the metamorphosis of specific objects. At the end of the chapter we present a simplified method attempting to solve the vertex correspondence determination problem in real time. All the discussion in this chapter assumes that the intersection of the objects with parallel planes results in single contours. This does not limit the generality of the conclusions drawn since, as shown in Chapter 5, the objects can be partitioned by user interaction to a collection of objects where our metamorphosis method can be applied.

6.2 Steps of the 3D metamorphosis method

The method described in the previous chapter operates in a number of consecutive steps.

Step 1 *Defining the contours*: This is the step of the algorithm that finds out the contours, necessary to describe each object involved in the metamorphosis. This can either be part of the metamorphosis or a pre-processing step performed once to form a library of objects.

Step 2 *Correspondence determination*: This is the step used to calculate the correspondence of contours and their vertices. This is the core of the metamorphosis method.

Step 3 *Interpolation of corresponding vertices*: This step involves the interpolation of the position of corresponding vertices.

Step 4 *Reconstruction of the intermediate objects*: This step reconstructs the intermediate objects from the intermediate contours created by the interpolation step.

6.3 Defining the contours

The objects that were used were modelled with the polygon-based representation. This representation consists of:

A list of the object vertices – each vertex is defined by a three-dimensional co-ordinate in a local co-ordinate system associated with the object.

The polygons – Each polygonal facet is described by a set of vertices that comprise this facet. The vertices are presented as indices in the vertex list.

We consider Euler-valid polygonal shapes without holes. In this case the topology of the objects obey the generalised formula:

$$V-E+F=2$$

where V, E, and F, are respectively the number of vertices, edges and facets of the object.

It is assumed that the user has provided a direction for the slicing in a form of a vector. The calculation of the contours created by the intersection of the object with the parallel planes proceeds as follows. Initially the objects are read from the database describing them. A list is created storing the co-ordinates of the object vertices as they are read from the database. A list of edges is created storing for each edge the two indices of the polygons that share it. In order to calculate a particular contour we test the intersecting plane with the edges of the object.

Let P_1P_2 be an edge of an object (Figure 6.1). The points $P_1=(x_1, y_1, z_1)$ and $P_2=(x_2, y_2, z_2)$ define the parametric equations:

$$x-x_1=t(x_2-x_1)$$
$$y-y_1=t(y_2-y_1)$$
$$z-z_1=t(z_2-z_1)$$

This set of parametric equations describes the straight line that passes through points P_1 and P_2.

The direction that the user has defined will be the normal to the slicing plane with direction cosines _, _, _. Then the equation of the plane in the normal form is:

$$\_x+\_y+\_z=p \qquad (6.1)$$

Having the equation of a plane in normal form we can obtain all planes parallel to the given plane by varying p.

The distance _ of a point P_1 (x_1, y_1, z_1) to the plane of equation (6.1) is:

$$\_=\_x_1+\_y_1+\_z_1-p \qquad (6.2)$$

As the vertices of the object are read from the database their distance from the plane $\_x+\_y+\_z=0$ that passes through the origin is calculated using the equation (6.2). We store the maximum and the minimum distances ($\__{max}$, $\__{min}$). The value of p of the planes that are used for slicing is within the range $\__{min}<p<\__{max}$. If n equidistant intersecting planes are used then:

$$p_i=\__{min} + i(\__{max}- \__{min})/n \qquad \text{for } i=1\ldots n.$$

In order to calculate the point where an edge P_1P_2 is intersected by a given plane we solve the linear system:

$$x-x_1=t(x_2-x_1)$$
$$y-y_1=t(y_2-y_1)$$
$$z-z_1=t(z_2-z_1)$$
$$\_x+\_y+\_z=p$$

If there exists a single solution, a point $P_{1,2}(x_3, y_3, z_3)$ and a value for t will be calculated. The point $P_{1,2}$ belongs to the straight line that passes through the points P_1, P_2. It also belongs to the line segment P_1P_2 if and only if $0\leq t\leq 1$.

The procedure used to calculate the intersection of an object with a plane proceeds as follows. Initially we search the edges of the object and find an edge P_1P_2 (Figure 6.1) that is intersected by the plane. The point where the plane intersects P_1P_2 is the first vertex of the contour. We search the edge-polygon list and find a polygon B that contains P_1P_2. This polygon will also contain another edge intersected by the plane (in Figure 6.1 edge P_3P_4). The point where the plane intersects P_3P_4 is the second vertex of the contour. The edge P_3P_4 also belongs to another polygon C, we follow P_3P_4 and visit C. In general suppose that X is the most recently visited polygon and *e* is the edge of X that led us to X; let *g* be the other edge of X intersected by the plane, the point of *g* where the plane intersects *g* is added to the list of vertices that form the contour. Edge *g* belongs to another polygon Y, we visit Y and start the same procedure anew for polygon Y. This procedure is continued till we return to the original polygon B. At this point the contour has been formed by the points of intersection determined as above. It is obvious that the contour vertices can be

calculated in a clockwise or anti-clockwise order along the contour. We used the convention that all contours vertices should follow a clockwise order. Therefore if the vertices are calculated in an anti-clockwise order the order of the contour vertices is reversed. This discussion assumes that the polygons of the shape are convex (which is a reasonable assumption in most practical cases).

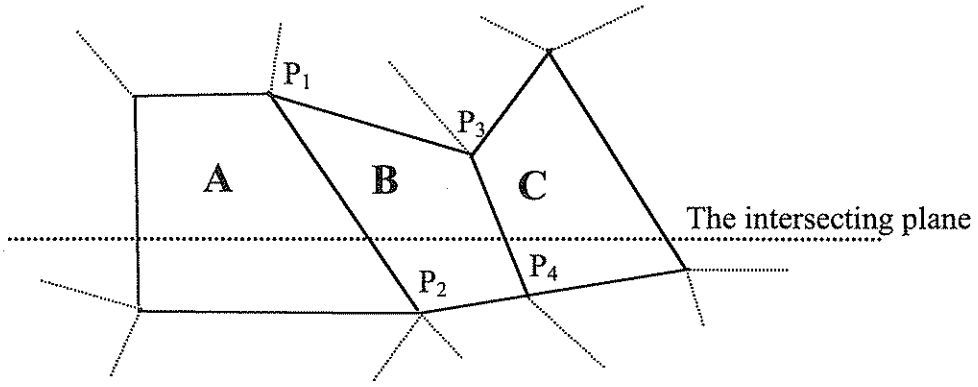


Figure 6.1: Calculating the contour created by the intersection of the object with a plane

Figure 6.2 presents the algorithm used for the calculation of a contour. The procedure FIND_EDGE(i) finds an edge of the object intersected by the i'th slicing plane, we will denote this plane as R_i.

Algorithm 6.1 Finding the contours

Input: An object described by polygonal representation

Output: A set of contours

```
for all planes i do
begin
        FirstEdge ← FIND_EDGE(i);
        CALCULATE(pol1, pol2, FirstEdge);
        FirstPolygon ← pol1;
        Findcontour(FirstEdge, FirstPolygon);
end;
```

```
procedure FIND_CONTOUR(edge OldEdge, polygon OldPolygon)
begin
        CALCULATE(pol1, pol2, OldEdge);
        if pol1=OldPolygon then
                NewPolygon ← pol2;
        else
                NewPolygon ← pol1;

        if NewPolygon!=FirstPolygon then
                for all edges e of NewPolygon do
                begin
                        If (e!=OldEdge) AND (e is intersected by the plane) then
                        begin
                                point ← INTERSECTION(e, plane);
                                ADD_CONTOUR(point);
                                FIND_CONTOUR(e, NewPolygon);
                        end;
                end;
        else
        begin
                point ← INTERSECTION(e, plane);
                ADD_CONTOUR(point);
        end;
end;
```

Figure 6.2 Algorithm for the calculation of a contour

A naïve implementation of FIND_EDGE would have to search randomly the edges of the object. In this case FIND_EDGE would have a complexity O(|E|) where |E| is the number of edges of the object. A more efficient method was identified for finding the first edge intersected by a given plane. This method resembles to a climbing algorithm and operates as follows. Assume that of we want to find an edge intersected by the first plane R_1 described by the equation $\_x+\_y+\_z=p_1$. As it was mentioned before, while the vertices of the object are read from the database their distance from the

plane _x+_y+_z=0 that passes through the origin is calculated using the equation (6.2). Let v be the vertex with the minimum distance $_{min}$. The distance of v from R_1 is $_{min}$–p. We calculate the distance from R_1 for all the vertices adjacent to v. If there exists a vertex x that its distance to the plane has a different sign than the distance of the vertex v then the edge (v, x) is intersected by the plane. If there is no such vertex then we visit the vertex with less distance from R_1. In general suppose y is the most recently visited vertex. For all the vertices adjacent to y we calculate their distance to the plane R_1. If there exist a vertex w that its distance to the plane R_1 has a different sign than the distance of the vertex y then edge (y, w) is intersected by the plane. If there is no such vertex then we visit the vertex z having the less distance from R_1 and start the same procedure anew starting from vertex z. This procedure will give us an edge (r, s) intersected by the first plane R_1. In order to calculate an edge intersected by the second plane we repeat the procedure described above starting with r. In general suppose that we have calculated an edge (b, d) intersected by the plane R_i we repeat the climbing algorithm starting from vertex b for the plane R_{i+1}.

The above implementation of the procedure FINDEDGE is presented in Figure 6.3.

Algorithm 6.2 Calculating an edge of the object intersected by the plane
Input: The polygonal representation of an object
Output: An edge of the object that is intersected by the plane

---

```
procedure FINDEDGE(vertex v, plane i)
begin
        int         min_distance ← infinite;
        int         current_distance;
        vertex      current_vertex;
        boolean     found ← FALSE;
        int         distance ← FIND_DISTANCE(v, i);

        While (there are vertices x adjacent to v)AND(NOT found)
        begin
                current_distance ← FIND_DISTANCE (w, i);
```

```
            if (current_distance* distance)<0
                    found ℜ TRUE;
            else
                    if |current_distance|<|min_distance| then
                    begin
                            min_distance ℜ current_distance;
                            min_vertex ℜ w;
                    end;
        end;
        if found then
                return (v, w);
        else
                FINDEDGE(min_vertex, i);
end;
```

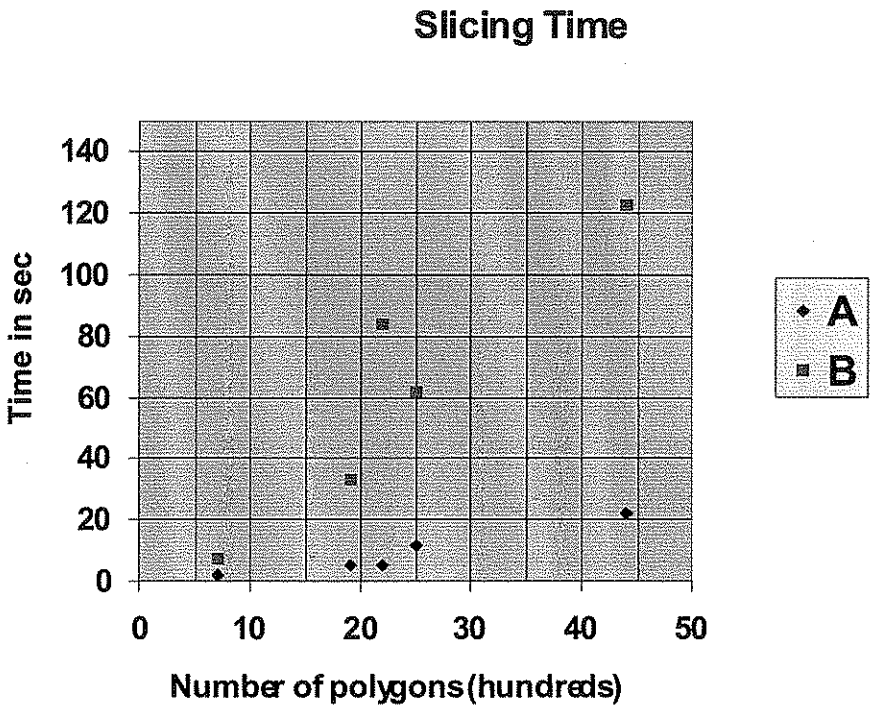
Figure 6.3: The procedure FINDEDGE

Table 6.1 presents the measurements for the time needed to slice an object. The second column contains the number of polygons of the object, the third column the time needed to find the intersection of the object with twenty planes while for the fourth column again twenty planes are used but only the most characteristic contours are used.

OBJECT	POLYGONS	SLICING TIME. THE OBJECT IS INTERSECTED BY 20 PLANES (SECONDS)	SLICING TIME. FINDING THE MOST CHARACTERISTIC CONTOURS (SECONDS)
Plane	**686**	2	7
Face	**1882**	5	33
Glove	**2191**	5	84
Teapot	**2518**	12	62
Head	**4357**	22	123

Table 6.1: Slicing time

Figure 6.4 presents a plot of the time versus the number of polygons of the object. When the planes are equidistant the time is almost proportional to the number of polygons. When we calculate the most characteristic contours the total time is dominated by the time needed to calculate the energy between every two successive contours.



A Intersection of the object with 20 equidistant planes

B Intersection of the object with 20 equidistant planes and finding the most characteristic of the resulting cross-sections.

Figure 6.4: Slicing time versus number of polygons

6.4 Correspondence determination

We will attempt to calculate the complexity of the second step of the algorithm, the correspondence determination step. Usually this kind of measurements is related to

the number of polygons or the number of vertices of the original objects. Since the proposed algorithm operates on a new representation of the objects involved it is difficult to relate the complexity of the algorithm with the original objects. Therefore we will consider the performance characteristics of this algorithm with respect to the contour representation.

Table 6.2 presents the time measurements we obtained with the application of the 3D metamorphosis algorithm. The second column contains the number of polygons of the first object, the third and fourth the number of contours and the total number of vertices in the contour representation. The next four columns present the same information for the second object. The last two columns present the time needed for correspondence determination and the time needed for the triangulation. In all these examples we used linear interpolation and the interpolation time was negligible compared to the times needed for the other steps.

FIRST OBJECT	Polygons	Contours	Vertices	SECOND OBJECT	Polygons	Contours	Vertices	Cor. Time	Triang. Time
Pot	2518	7	579	**Face**	1882	17	1022	122	29
Glove[1]	2191	16	485	**Head**	4357	23	2896	301	110
Glove[1]	2191	16	485	**Head**	4357	23	607	52	10
Glove[1]	2191	16	485	**Banana**	256	17	282	23	7
Glove[1]	2191	16	485	**Face**	1882	17	1022	100	23
Glove[1]	2191	16	485	**Glove**[1]	2191	16	485	41	6
Glove[1]	2191	16	485	**Teapot**	2518	7	579	56	20
Glove	2191	16	485	**Biplane**	686	10	206	18	6
Face	1882	29	984	**Biplane**	686	10	206	36	16
Face	1882	29	2669	**Head**	4357	23	2896	1815	184
Face[1]	1882	29	984	**Head**[1]	4357	23	607	103	13
Biplane	686	10	206	**Biplane**	686	10	206	9	3
Biplane	686	10	206	**Teapot**	2518	7	579	28	14
Biplane	686	10	206	**Banana**	266	17	282	10	4
Biplane	686	10	206	**Face**	1882	17	1022	46	22

Biplane	686	10	206	**Head**	4357	23	2896	130	106

[1]Simplified representation

Table 6.2: Time measurements

The timings show that this method is far from giving results in real time but for most case the result can be calculated at interactive rates. This is not unusual for the methods used for correspondence determination. For indicative purposes some of the times needed for some of the other methods, as provided by their authors are given below.

The method presented in [HUG92] is quite time consuming, for a volume of size n the computation of every inbetween image takes at least $O(n^3 \log n)$ time.

Lerios et. al in [LER95] provide an example were the objects were represented by a 300^3 voxel grid and the correspondence which was specified by 26 element pairs was established in 8h by an novice and 3h by an expert then 8h were necessary on a SGI Indigo V2 to obtain a sequence of 50 in-between shapes.

Cohen et. al. in [COH98] reported that it took 40 min on a SGI 4400 to create an intermediate 200^3 volume with 20 anchor points.

The method presented in [KAN97] is one of the fastest methods. The authors report that when the first object consists of 5798 polygons and the second object of 3765 polygons it took 46 seconds on a SGI 4400.

For the mainly interactive methods the time needed is dominated by the time necessary for the animator to specify a number of corresponding features on the two objects.

[DEC96] The authors report that it took 1h 15min of interaction to create a correspondence between two relatively simple shapes.

[STA98] Again the user interaction is extremely heavy and according to the authors, it took them 6h to create a morph between a human (17374 polygons) and a triceratops (5438 polygons).

Let us consider an object approximated by m parallel contours morphed to an object approximated by n contours. The correspondence determination involves the filling of an m←n matrix with each entry of the matrix involving the application of Sederberg's and Greenwood's algorithm to a pair of polygonal contours. Let us consider that the first contour set contains $|V_1|$ vertices and the second contour set $|V_2|$ vertices. $|V_{1j}|$ denotes the number of vertices in the j'th contour of the first object while $|V_{2j}|$ denotes the number of vertices in the j'th contour of the second object. We can write:

$$|V_{11}|+|V_{12}|+\ldots+|V_{1m}|=|V_1|$$
$$|V_{21}|+|V_{22}|+\ldots+|V_{2n}|=|V_2|$$

The number of operations needed to transform the i'th contour of the first object to the j'th contour of the second object is in the order of $|V_{1i}|\_|V_{2j}|$.

The order of the total number of operations is:

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(|V_{1i}|?|V_{2j}|)=\sum_{i=1}^{m}(|V_{1i}|?|V_2|)=|V_1|?|V_2|$$

Therefore the complexity of the algorithm is $O(|V_1|\_|V_2|)$.

In Figure 6.5 we keep the vertices of the first object constant and presents the plot of the time needed for the correspondence determination versus the number of vertices of the second object. There are three different paths, one for the first object being 'glove', one for 'biplane' and one for 'face'. As we can see the time is almost proportional to the number of vertices of the second object which agrees with the theoretical estimation of the complexity.

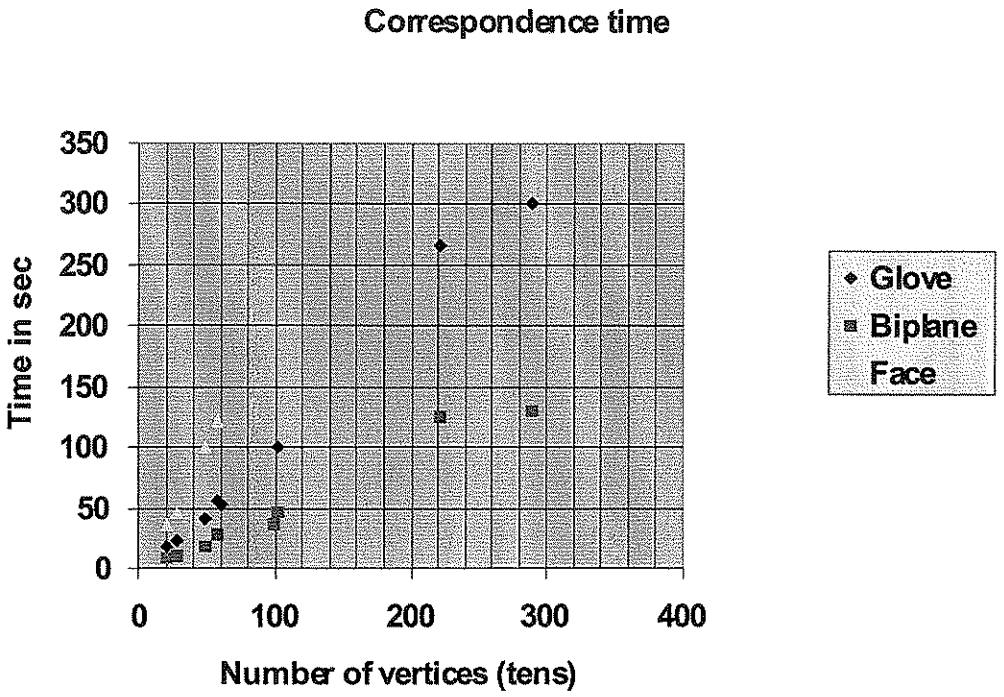


Figure 6.5: Correspondence time versus number of vertices in the contour set.

The number of vertices in the contour representation is related to the number of contours used. For example the following chart (Figure 6.4) shows that for the object 'Plane' the number of vertices is almost proportional to the number of contours used.

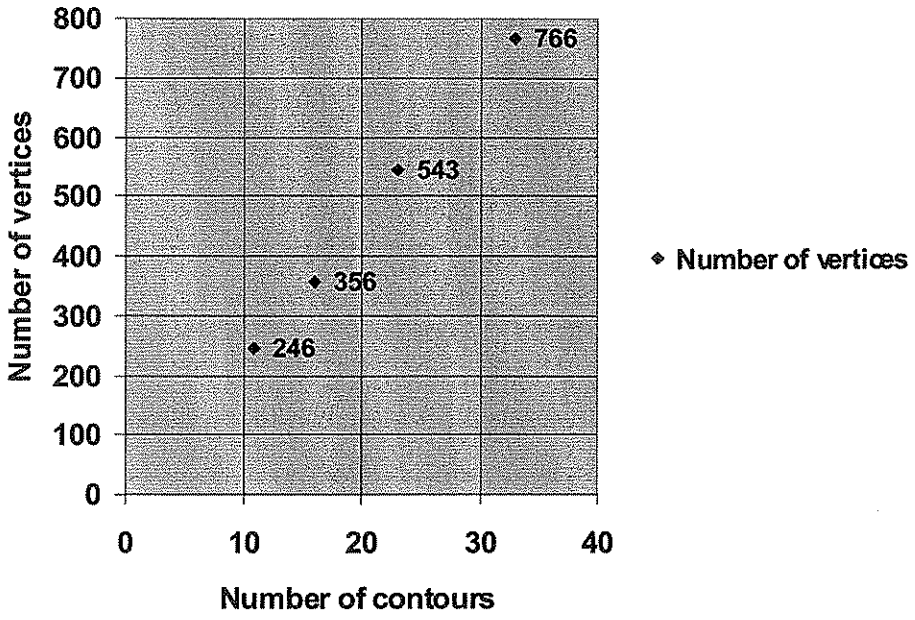


Figure 6.6: Number of vertices vs contours for the object 'teapot'.

In Table 6.3 we present the time measurements for the metamorphosis of the two three-dimensional polygonal objects 'banana' and 'biplane' presented in Chapter 5. Each row corresponds to a particular metamorphosis where a different number of contours is used for the object representation. The first column of the table contains the number of contours used to approximate the objects. The second column gives the time needed to calculate the correspondence of contours and their respective vertices. The algorithm was executed on a Octane SGI workstation. Other examples with different pair of objects have shown analogous results.

NUMBER OF CONTOURS FOR EACH OBJECT	CORRESPONDENCE TIME. (SECONDS)	TRIANGULATION TIME PER FRAME. (SECONDS)	TOTAL TIME FOR A 24 FRAME ANIMATION. (SECONDS)
11, 11	7	0.4	17
11, 16	11.3	0.5	23
11, 23	16	0.8	127
11, 33	23	1.2	59
16, 11	11	0.5	23
16, 16	16	0.6	24
16, 23	13.5	0.8	38

16, 33	34	1.1	61
23, 11	9	0.8	28
23, 16	13	0.8	32
23, 23	35	0.8	54
23, 33	50	1.1	77
33, 11	24	1.1	61
33, 16	34	1,1	72
33, 23	51	1,2	70
33, 33	71	1.1	99

Table 6.3: Time for correspondence determination

The following chart (Figure 6.5) relates the time needed for correspondence determination with the number of contours used to approximate the original objects 'banana', 'biplane'.

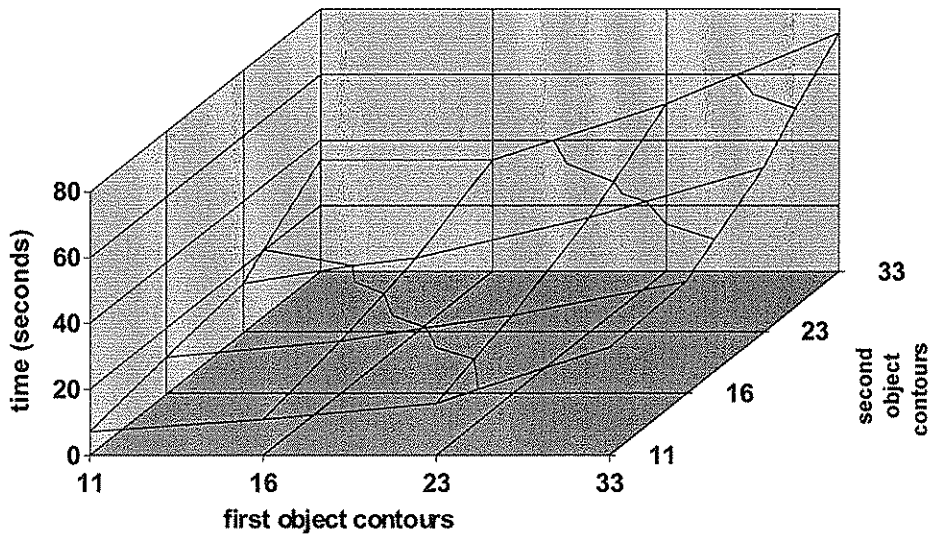


Figure 6.7: Correspondence determination time, varying the number of contours describing the two objects.

Let us consider the situation that the user breaks-up the objects into parts and define correspondences for these parts. Suppose that the two objects are divided in p parts. The number of the operations needed to establish correspondences is in the order of:

$$\sum_{i=1}^{p} \left(\left| V_1^i \right\| V_2^i \right|)$$

Where $|V_1^i|$, $|V_2^i|$ is equal to the number of vertices in the i'th part of the first and the second object respectively.

If we consider that each part has the same number of vertices the number of operations is in the order of:

$$\sum_{i=1}^{p} \frac{|V_1|}{p} \cdot \frac{|V_2|}{p} = p \cdot \frac{|V_1||V_2|}{p^2} = \frac{|V_1||V_2|}{p}$$

Hence by breaking the object into parts and defining correspondences for these parts we reduce the number of operations by the number of parts.

6.5 Complexity of the intermediate objects of the metamorphosis

Let us consider the metamorphosis of an object described by n contours to an object described by m contours. Without loss of generality we can assume that $m \leq n$. Let k be the number of contours for an intermediate object. Then:

$m \leq k < m+n$

Let $|V_1|$, $|V_2|$ be the total number of vertices of the first and the second set of contours respectively. These vertices will be distributed across the set of contours describing the original objects. We denote v_{1j}, v_{2j} the number of vertices in the j'th contour of each object.

Clearly $v_{1j} < |V_1|$, $v_{2j} < |V_2|$ and

$v_{11}+v_{12}+\ldots+v_{1m}=|V_1|$

$v_{21}+v_{22}+\ldots+v_{2n}=|V_2|$

We will calculate the lower and upper limit of the number of vertices |V| of an intermediate object (all intermediate objects have the same number of vertices).

Lower limit: It is clear that an intermediate object will have a vertex corresponding to each vertex of the initial objects therefore it is $\max(|V_1|, |V_2|)<|V|$

Upper limit: The k'th contour of the intermediate object will be an intermediate of two contours i, j with v_{1i}, v_{2j} vertices respectively. Let v_k be the number of vertices of this intermediate contour. Then $v_k<v_{1i}+v_{2j}$.

As it was mentioned it is k<m+n. Therefore in the worst case the intermediate objects will be described by m+n contours.

Let v_i denote the total number of vertices of the i'th contour of an intermediate object.

$|V|=v_1+v_2+\ldots+v_{m+n}$ (6.3)

$$\begin{aligned} v_1 &\le v_{11}+v_{21} \\ &\cdot \\ &\cdot \\ &\cdot \\ v_{m+n} &\le v_{1m}+v_{2n} \end{aligned} \qquad (6.4)$$

If we consider that all contours of the first object contain the same number of vertices V_1/m and similarly all contours of the second object contain the same number of vertices V_2/n summing all the expressions (6.4) we get:

$$|V| \le (m+n)\left(\frac{|V_1|}{m}+\frac{|V_2|}{n}\right) \quad V \le |V_1|\left(\frac{m+n}{m}\right)+|V_2|\left(\frac{m+n}{n}\right)$$

The number of triangles needed to triangulate the surface between two contours of v_1, v_2 vertices is v_1+v_2.

If the object is represented by k contours then the total number of triangles of the final reconstructed object is:

$$\sum_{i=1}^{k-1}(v_i + v_{i+1}) = v_1 + v_k + 2\sum_{i=2}^{n-1} v_i = v_1 + v_k + 2|V|$$

6.6 Interpolation of corresponding vertices.

Once the correspondence of the object vertices has been established the transformation is computed by interpolating between each pair of corresponding vertex locations. Either linear or spline interpolation can be employed for calculating the vertex trajectories. In both cases the complexity of the algorithm is O(|V|) where |V| is a number of vertices of the intermediate objects.

Since $|V| \leq |V_1| \cdot \frac{m+n}{m} + |V_2| \cdot \frac{m+n}{m}$ the complexity of the interpolation step is $O(|V_1|+|V_2|)$

6.7 Reconstruction of the intermediate objects.

The interpolation step results in one set of contours for each frame of the metamorphosis. If we apply a surface reconstruction algorithm for each set of planar contours we can construct the intermediate objects of the metamorphosis. The method we used for the triangulation of the surface between each pair of successive contours was introduced in [FUC82]. The following table (Table 6.4) presents the time needed to triangulate an intermediate object of the metamorphosis 'banana' to 'biplane'.

NUMBER OF CONTOURS FOR EACH OBJECT	TRIANGULATION TIME PER FRAME (SECONDS)
11, 16	1.6
16, 11	1.8
16, 16	1.8

11, 23	2.4
16, 23	2.5
23, 23	2.3
23, 11	2.4
11, 33	3.4
16, 33	3.4
23, 33	3.4
33, 33	3.5
33, 11	3.3
33, 16	3.3
33, 23	3.2
33, 33	3.4

Table 6.4 : Triangulation time per frame

6.7.1 Alternative methods for visualising the intermediate contour sets

As we can notice in the following charts (Figures 6.8-6.11) the triangulation time becomes increasingly significant as more frames are used for the metamorphosis. These measurements were obtained by applying the metamorphosis method to the objects 'banana', 'teapot' which were represented with different number of contours from chart to chart.

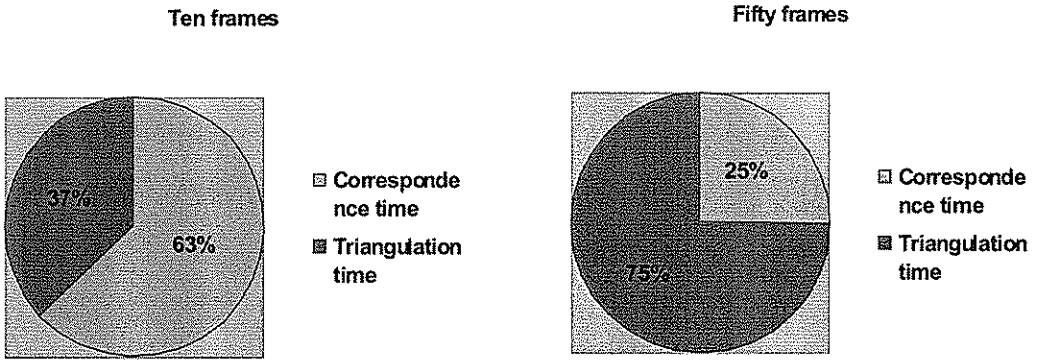


Figure 6.8: Percentage of the total execution time spent for correspondence determination and triangulation when both objects are represented by 11 contours.

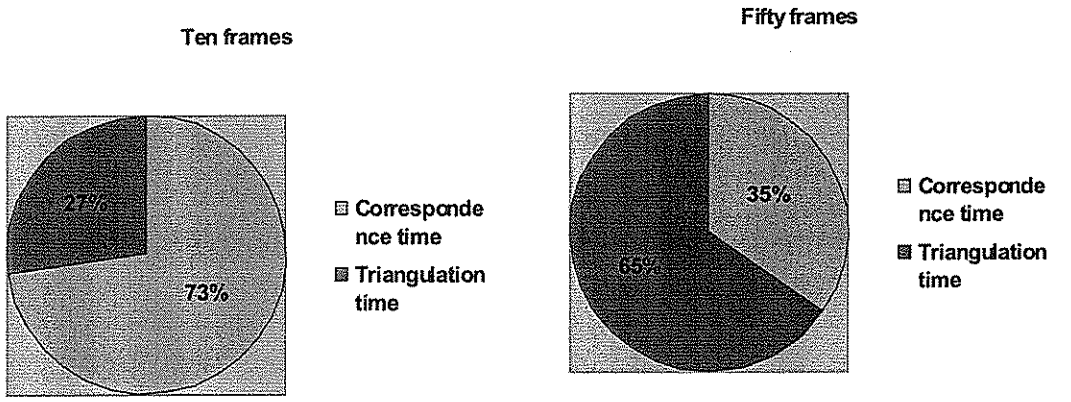


Figure 6.9: Percentage of the total execution time spent for correspondence determination and triangulation when both objects are represented by 11 contours.

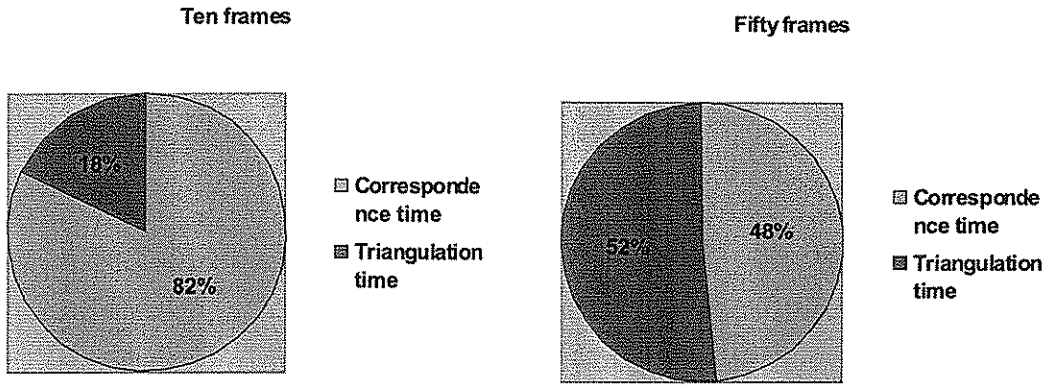


Figure 6.10: Percentage of the total execution time spent for correspondence determination and triangulation when both objects are represented by 23 contours.

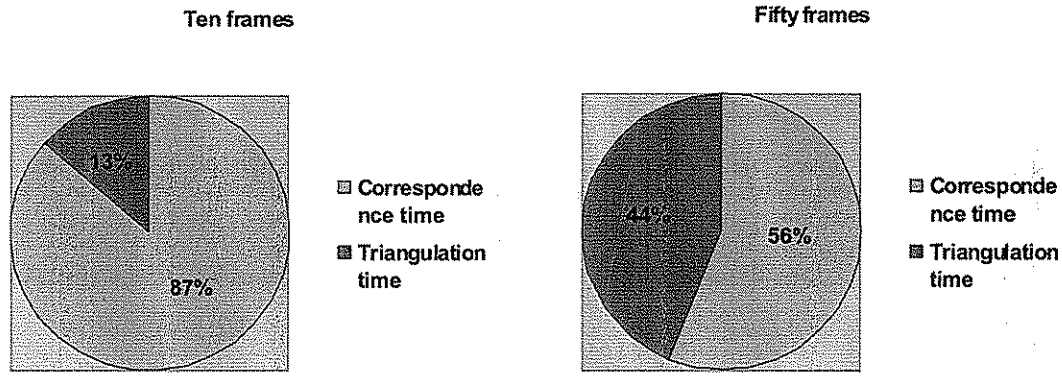


Figure 6.11: Percentage of the total execution time spent for correspondence determination and triangulation when both objects are represented by 33 contours.

Every intermediate contour of the animation keeps the same number of vertices. It is reasonable to assume that consecutive frames of the animation will contain the same triangles. Using that observation we can calculate the optimal triangulation for one frame of the animation and use all the connectivity information for k successive frames. That way we can reduce the triangulation time for a factor close to k.

6.7.2 Volume rendering techniques

The term volume rendering describes techniques that enable the visualisation of sampled scalar functions of three spatial dimensions. The major application in this

field is medical imaging, where volume date is available from X-ray Computer tomography scanners. CT scanners produce three-dimensional stacks of parallel plane images. The availability of stacks of parallel plane images motivated the development of techniques for viewing such volume data sets as three-dimensional field rather than as individual planes. This gives the immediate advantage that the information can be viewed from any viewpoint.

There have been techniques developed to display such volume data sets on a computer graphics monitor as some projection of the data rather than a cross-section of it. The marching cubes algorithm was independently reported by Wyvill and McPheeters in 1986 [WYV86] and by Lorenson and Cline in 1987 [LOR87]. Both algorithms essentially operate in two phases. In the first pass the cubes forming the boundary set are calculated. In the second phase of the algorithm those cubes in the boundary set are examined and a set of connected polygons are produced for rendering.

It easy to notice the similarity of the problem visualising these stacks of parallel plane images to the problem visualising the intermediate contour sets created by our metamorphosis method. Therefore volume rendering techniques can be used to visualise the intermediate contour sets.

6.8A simple heuristic method for the metamorphosis of closed planar contours

The time measurements presented in section 6.2.1 make it clear that the described method is far from giving results in real time. In this section we present a simplified heuristic method that comes quite close of actually producing real-time performance.

Suppose that we want to blend two closed planar contours. (Figure 6.12a). Let us assume that the user has defined a pair of corresponding vertices on the two contours so that vertex A of the first contour corresponds to vertex E of the second contour. We unfold the two closed contours to two straight linear pieces AA', EE' (Figure 6.12b).

The simple heuristic we propose is that a point X of the first contour corresponds to a point X' of the second if $\frac{AX}{AA'} = \frac{EX'}{EE'}$ that is we consider that corresponding points are proportionally spaced around the two contours.

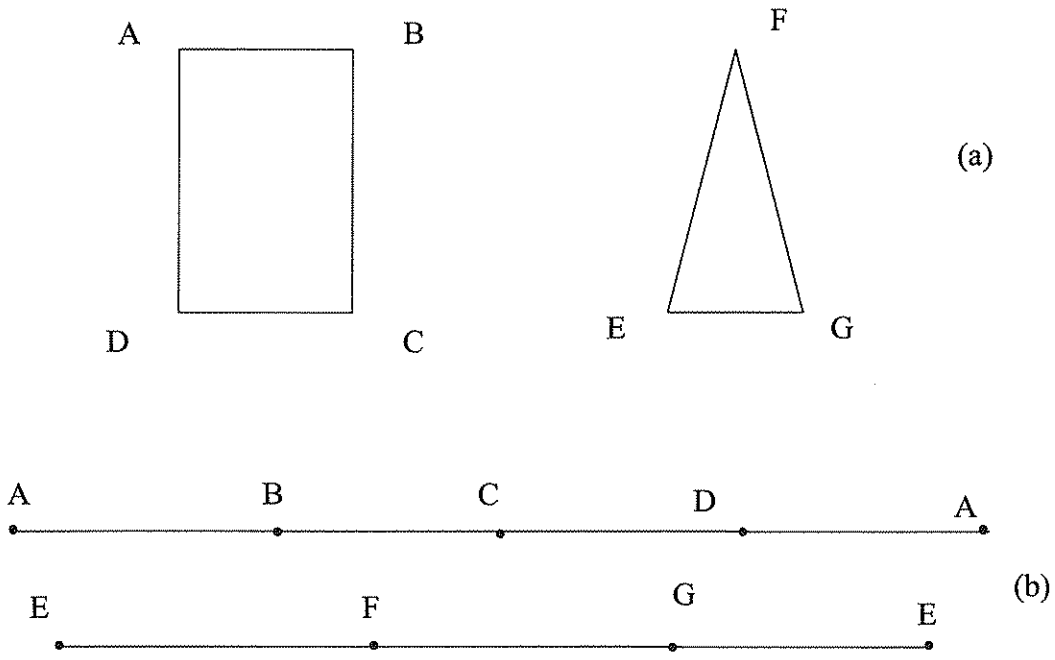


Figure 6.12: Stretching the contours of the two shapes

The proposed method is extremely fast (it can calculate the correspondence in real-time) and favours well in situations that the animation has a scaling or rotation component. In Figure 6.13, we present a computer generated animation, between two closed polygonal contours, obtained by the application of this method.

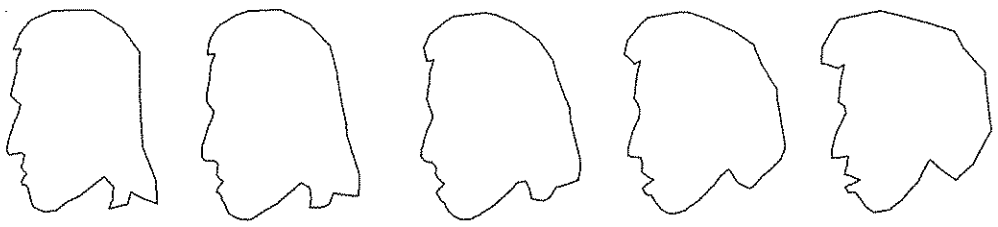


Figure 6.13: Metamorphosis between two closed polygonal contours

In this example, the user has defined a pair of corresponding vertices and the rest of the correspondences were calculated automatically. A finer level of control over the animation is possible if the user defines more than one pair of corresponding vertices.

6.9 Extension to 3D object metamorphosis.

Suppose that we have a pair of three-dimensional objects, made by a set of polygons, that we want to morph (Figure 6.14).

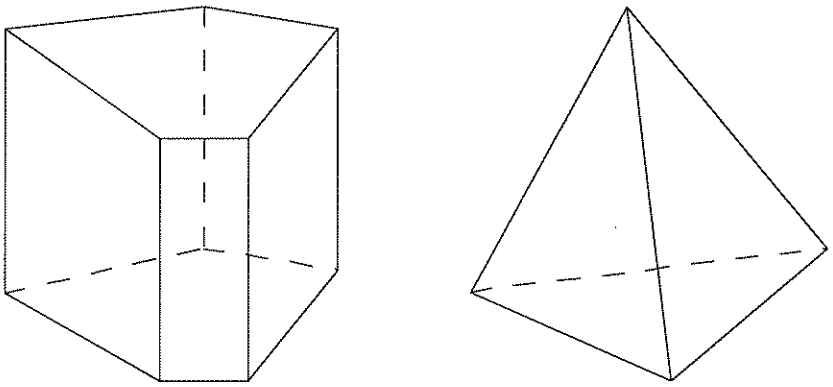


Figure 6.14: Two polygonal objects

Again we consider the intersection of an object with offsets of a given plane. But this time we consider that both objects are intersected with the same number of planes. Depending on the number of contours, we can reconstruct the original model to a reasonable precision using one of the existing algorithms for surface reconstruction from planar contours.

For the time being, we will consider situations where the intersection of the object with each of the different planes is a single contour.

We simplify the problem of morphing the original objects to the problem of morphing these two sets of planar contours. Since the contours do not intersect, we can order them according to their geometric position in space. We will use the position of a planar contour in this ordering as an index for referencing it.

Suppose that we are using M contours to describe each object. We can order the contours of each object from 1 to M (Figure 6.15).

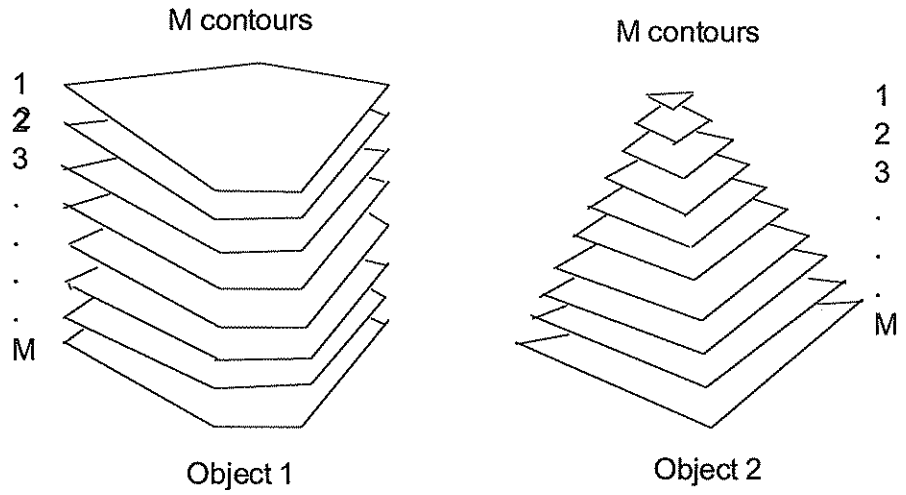


Figure 6.15: Approximation of the original objects with M parallel planar contours

We assume that the i'th contour of the first object corresponds to the i'th contour of the second object. By applying the methodology described in Section 6.10 for each pair of corresponding contours, we get new intermediate sets of contours. Once we have these, we can apply one of the existing algorithms for surface reconstruction from planar contours that will result in the intermediate objects of the metamorphosis.

6.9.1 Number of polygons of the intermediate objects

Let us consider the metamorphosis of a pair of objects described by n contours. An intermediate object of the metamorphosis will also be described by n contours.

Let us denote as $|V_1^i|$ number of vertices in the i'th contour of the first object and $|V_2^i|$ the number of vertices in the i'th contour of the second object. The i'th contour of each intermediate object will have $|V_1^i|+|V_2^i|$ vertices. The total number of vertices of an intermediate contour set is:

$$\sum_{i=1}^{n} (|V_1^i|+|V_2^i|)$$

The number of triangles needed to triangulate between two contours of x, y vertices is x+y.

The total number of triangles of an intermediate object of the animation is:

$$\sum_{i=1}^{n-1}(|V_1^i|+|V_2^i|+|V_1^{i+1}|+|V_2^{i+1}|) = 2?|V_1|+2?|V_2|-\left(V_1^1|+|V_1^n|+|V_2^1|+|V_2^n|\right)$$

6.9.2 Examples

In Figure 6.16, we present an animation sequence obtained by the application of this method. Here we transform the object 'pear' to the object 'glove'. Both objects are represented by 10 contours. These contours are obtained by the intersection of the original object with a set of parallel planes. The orientation and the number of planes are user defined and this is a way that a user can influence the resulting animation sequence

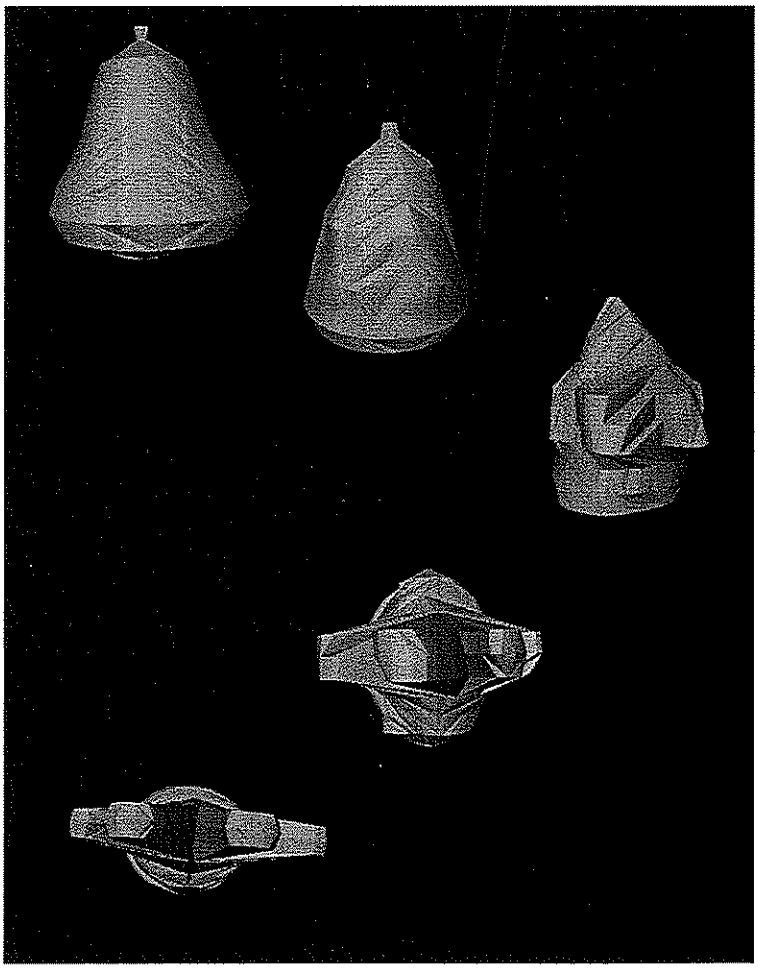


Figure 6.16: Animation from 'pear' to 'glove'

By applying the method described in section 6.4 to these two sets of contours, we can find a correspondence for their vertices. Once the correspondence of the vertices has been established the intermediate contours sets are computed by interpolating between each pair of corresponding vertex locations. Once we have calculated the intermediate contour sets, we apply an algorithm for surface reconstruction from planar contours [FUC82] to get the intermediate objects. This algorithm calculates a triangulated surface between every two successive contours. Since all the faces in the resulting intermediate objects are triangles, they remain planar during the animation.

Furthermore depending on the number of contours that are used for the approximation of the original objects and the algorithm that is used in the surface reconstruction step, we can get results close to a real time rate. For example the time needed for the correspondence determination from the object 'face' to the object 'head' when both objects are represented by 23 contours is 0.5 seconds.

The described method allows some user control over the transformation through mechanisms such as selecting the axis for the slicing of the objects and defining the number of slices for the objects.

6.10 Conclusions

In this chapter we evaluated the three-dimensional metamorphosis method described in Chapter 5 according to a number of different criteria. These criteria included the estimation of the time complexity for the different steps of the method. From the experimental results it was demonstrated that the algorithm is far from giving results in real time. There was also a theoretical estimation presented of the number of vertices and polygons of the intermediate objects. Experimental results showed that the complexity of the intermediate objects tends to be close to the complexity of the most complex of the initial objects. A simplified algorithm was also presented that manages to calculate correspondences in real-time.

Chapter 7

Parallelisation and benchmarking

7.1 Introduction

In this chapter we first discuss issues related to parallel processing and particularly with the application of parallel processing to computer graphics. We describe the methods employed for the parallelisation of Sederberg's and Greenwood's algorithm and the three-dimensional metamorphosis method and present the benchmarking results obtained. All parallelisation work was implemented on the ML-PVA parallel architecture of Queen Mary and Westfield College.

7.2 The ML-PVA Accelerator

The parallelisation was implemented on the ML-PVA accelerator originally built for the MONALISA project [SAH94] demonstrator. This accelerator is hosted by a Unix graphics workstation allowing it to be used as a shared resource connected to the network. The current architecture of accelerator, ML-PVA, in QMW comprises of the following parts. (Figure 7.1)

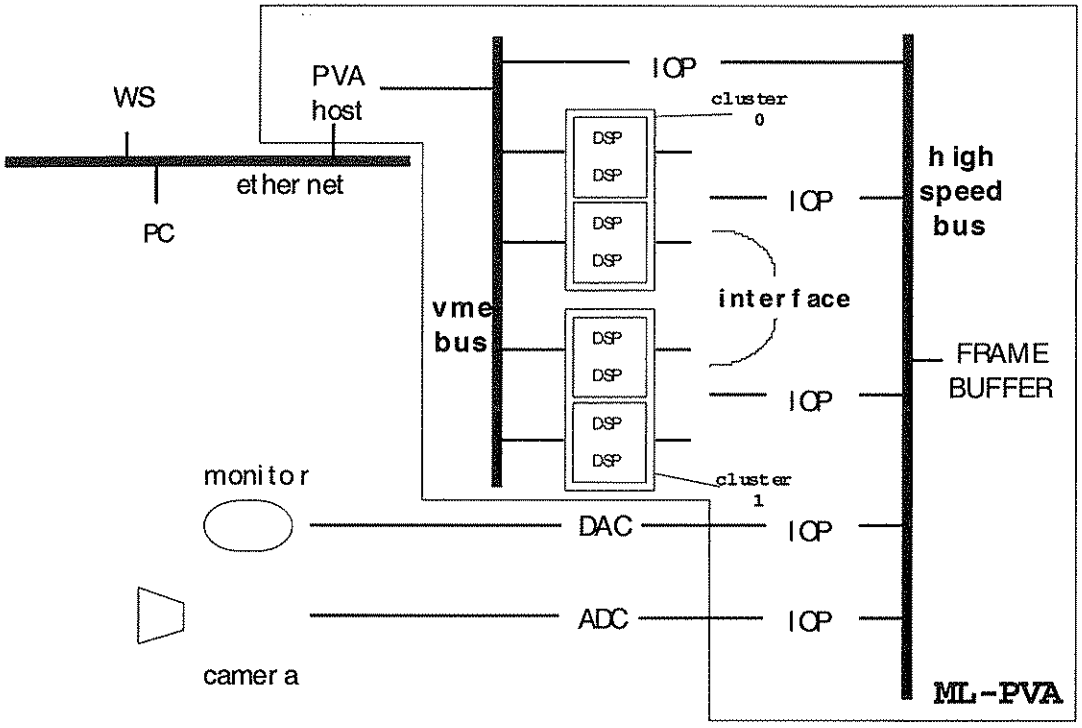


Figure 7.1: The ML-PVA architecture

- A frame buffer of 32Mbytes. The frame buffer subsystem is made up of an address generator, a set of RAM memory boards, and a number of FIFO buffered I/O processor boards which share a high speed synchronous back-plane bus. The operations within the frame subsystem and the communications to host computers are controlled by a Motorola 68040-based single board processor, running the real time operating system OS9.

-5 IOPs (Input Output Processors). These are intelligent input output processors used for accessing the frame buffer. A standard interface is provided by the use of this hardware. In generall all IOPs are the same except a few add on devices for the ones attached to the converters.

A 68030 main processor (PVA Host) configured with

- A local hard disk of 40Mbytes
- An external hard disk of 320Mbytes (backup disk)
- A floppy disk driver

A card computer running a real-time operating system, controlling the ML-PVA, and responsible for the management of pool processors (DSPs), IOPs, and the frame buffer.

- 8 DSPs (DSP96002) Processing units on which given tasks run (operating at 40Mhz with a peak performance of 60Mflops) each with 4MB local SRAM. Two of them located on the same board (Figure 7.2), sharing two global on-board memory banks; a 4MB bank of SRAM and a 16MB bank of DRAM both of which are mapped in the VME address space. Each board also has 2 DbeX (peripheral expansion bus) interfaces which are used for data transfer to/from the frame buffer. Two boards constitute a cluster which has single IOP to access the frame buffer.

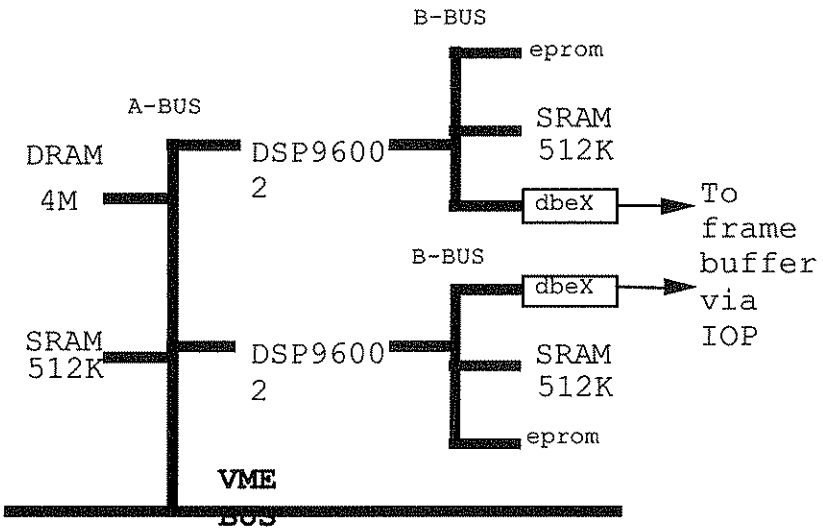


Figure 7.2: Processor board memory/bus configuration.

7.2.1 The Programming model of ML-PVA : SAPS

The programming model of ML-PVA is based on Self Adaptive Parallel Servers (SAPS) model [SAH91], which provides Single Program Multiple Data (SPMD) parallelism. In this model, a parallel application is composed of a number of copies of the same sequential program each running on a separate processor node. This form of parallelism is widely used on message passing multiprocessor systems because it has a relatively simple and well defined structure, and it facilitates a uniform and easy-to-understand usage of communication primitives. SPMD style also provides a

framework for developing parallel application software using sequential programming techniques, and therefore enables application software already developed for sequential machines to be used for parallel architectures, without undergoing major changes.

Under the SAPS model, the mechanisms for SPMD parallelism are encapsulated within servers. A server when requested executes multiple copies of an associated sequential procedure in parallel in SPMD mode. The data is provided by the client within the service request. The interface between the application programs, as clients, and the parallel servers is conveniently hidden in the procedure call mechanism which is a well-understood facility to develop modular programs.

The interaction between a SAPS and an application as its client, for the actual provision of the service, is based on standard remote procedure call which is structured as a service-request and a service-reply is also supported by a data objects scheme which enables data decomposition. Servers fetch application data via operation invocations on data objects.

The server-application (client) duality provides the means for the separation of concerns and therefore the separation of the building of servers and their utilisation by the applications. Applications are conventional sequential programs developed independent of the concerns for parallelism. The main building block of a server is also a sequential procedure; a parallel SPMD structure is obtained when this procedure is interfaced to a standard template. This scheme allows building servers using existing sequential software without major modifications.

A SAPS has a multi-process structure. This structure contains the mechanisms for the reception of service requests, their processing, and the transmission of the results back. A dispatcher process and a pool of workers form a process farm where the dispatcher farms out work to workers and each worker when becomes idle, requests for more work from the dispatcher process. It is the multiplicity of the workers that provide parallelism within the server. Workers run on the pool processors in a one node per worker fashion. The number of nodes used by the workers of a particular server is not statically decided, it depends on run time availability of nodes. A server

can start operating with a single worker and dynamically increase its worker population at run time as more nodes become available.

Within the SAPS structure it is the scheduler process which is responsible for the resource management activities. It manages the configuration (it creates the dispatcher and worker processes), and it reconfigures the SAPS structure by adding or deleting workers. The scheduler coordinates its activities with other SAPS schedulers through the pool manager.

The structure of a server is completely transparent to its clients. Each server has two message communication access points: one for handling service requests (service access point), and the other for monitoring the processor resources and co-ordinating its resource usage with other computing agents (resource management access point).

The SAPS structure as a whole is supported by a standard software template. The complete functionality of a SAPS as a generic server object is programmed within this template. To create a SAPS blueprint for a particular service all that is required is the interfacing of a conventional sequential procedure (the task proper) to a copy of the template. This is achieved via a local procedure call interface.

7.2.2 Developing an application for ML-PVA

There are two types of application programming;

1- PVA-Host driven (non-server) programs
2- System-Host driven (client-server) programs

The first type of application programs run over a number of DSPs under the ultimate control of PVA host. There is no direct communication between PVA host and System Host (Workstation) while the programs running over DSPs. Only the initialising actions are taken by the application running on the System Host. Generic application of this kind reads data from frame buffer, performs a specific algorithm

and writes results to frame buffer. This kind of application does not need any code to be run on the System Host.

The second type, client-server, performs an algorithm over a number of DSPs under the control of System-Host. The system-host, sends tasks and collects the result. It is system-host's responsibility to provide data to the server tasks. The responsibility of PVA-Host, for this type of programming, is to fetch the availability messages from DSPs and to dispatch a task.

Developing and compiling a non-server program is explained first. For client-server based programming only the differences are mentioned in the following paragraphs.

7.2.3 Implementing a Client Server program on ML-PVA

A client server program runs over all three platforms. Although a programmer would not need to develop any program on PVA-Host, PVA-Host takes the responsibility of communicating with System-Host and DSPs, and dispatching the tasks.

The actual implementation of a client program based upon an RPC command to PVA-Host. Before calling this RPC command, which is sending tasks to pool manager to be dispatched, client is responsible for sending the data and all relevant information to the frame buffer.

The basic protocol for a client-server based programming over this system is given in Fig 7.3.

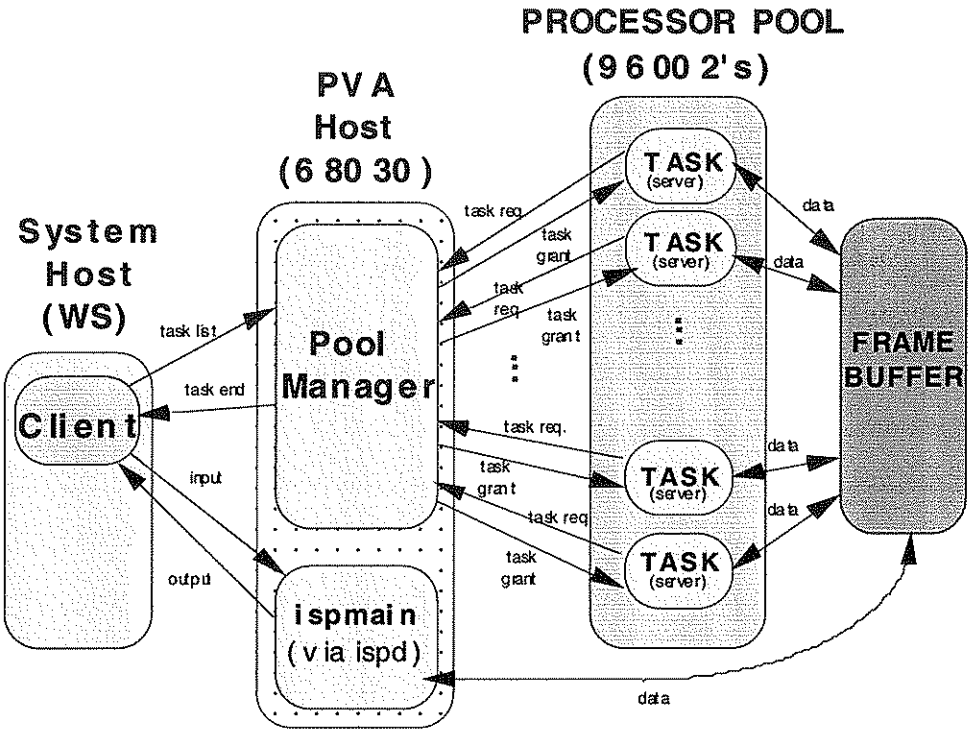


Figure 7.3: Platforms, entities and communication schemes for client-server programming.

The sample program running on the system host has six distinguishable parts;

- Pre-processing (a sequential part of the algorithm)
- Initialising the parallel processing environment
- Preparing and sending data
- Preparing and sending tasks (implicitly waiting for the results)
- Collecting the results.
- Post-processing of the results (sequential part of the algorithm)

The sample program running on DSP has three distinguishable parts. The third part is a forever loop comprising five sections.

- Pre-processing (a sequential part of the algorithm)
- Reading initialisation data

- Waiting for a task to be dispatched forever
    - Getting task and task information
    - Reading data
    - Computing and producing local results
    - Communicating with pool manager over the results
    - Writing results

Given parts should be considered as a superset of any program structure, as all of them are not expected to be in each program.

7.3 Parallelisation of Sederberg's and Greenwood's Algorithm

Sederberg's and Greenwood's algorithm addresses the two-dimensional shape-blending problem. We discussed this algorithm in detail in Section 2.2. Sederberg's and Greenwood's algorithm assumes that an initial correspondence has been defined for a pair of vertices. Here we consider the more general problem where no pair of corresponding vertices has been defined. Suppose that the two polygonal contours that are to be morphed have m, n vertices respectively. A given vertex of the first contour will correspond with at least one of the n vertices of the second. Therefore the algorithm must be executed for n initial correspondences and the chosen correspondence is the one that needs the minimum amount of energy. The execution of the algorithm for one of these n initial correspondences has been identified as the simplest task that we could decompose the total work to.

A server must have access to the co-ordinates of the vertices so that it can calculate the energy of the contour blending for a given initial correspondence. When this calculation finishes it must also be able to return its result to the host. Therefore, there are routines implemented allowing the host to put the data in the frame-buffer and read the results from the frame buffer.

Since it is desirable to minimise the communication between the processor pool and the frame buffer it was decided that tasks should be organised as packets of the simple

identified task. In the situation that k of these packets are used the processor assigned the i'th task would calculate the energy for the blending of the two contours when vertex 0 of the first shape corresponds with vertices (n/k)i,...,(n/k)(i+1)-1 of the second. The server finds which of these initials correspondences yields the minimal amount of work and what is the complete vertex correspondence for that initial correspondence. Finally the server attempts storing its result to the frame buffer and does so if the energy needed for its solution is less than the energy calculated from all other tasks that have been served. The approach we followed, for comparing the partial results, was the implementation of a WK kernel command WKCombine that combines the results of the different servers storing the results in the frame buffer (overwriting any previously stored result) only if the energy needed is less than the current found minimum. When the execution of all tasks is finished the Silicon Graphics host reads the stored result from the frame buffer, this result is the global solution to the correspondence problem.

The Remote Procedure Call that is sending a task to the pool manager contains the following fields: the name of the first object in the frame buffer keeping the data of the vertices of the first contour, the name of the object in the frame buffer keeping the data of the vertices of the second contour, two numbers equal with the number of vertices of each contour and a number of shifts that should be performed in the first vertex of the second polygon to agree with the initial correspondence that must be calculated.

7.3.1 Benchmark Results

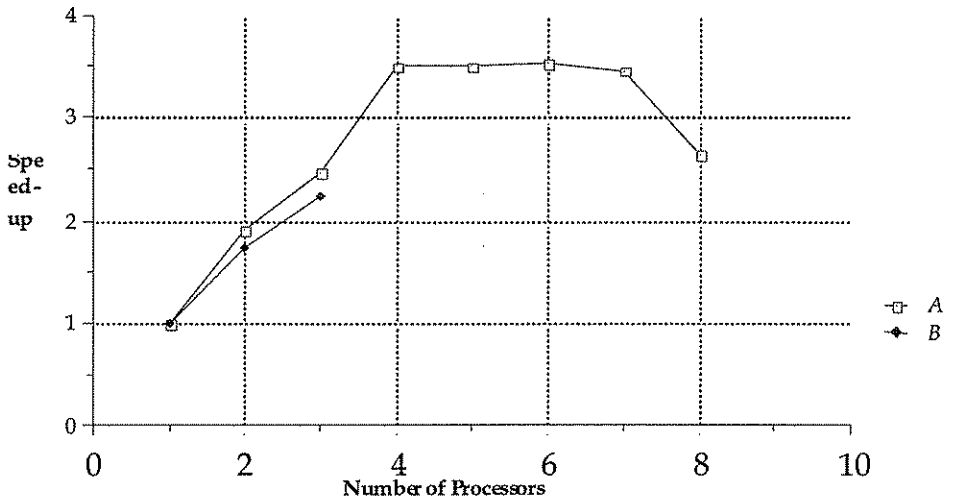
Two approaches have been followed for determining the size of the packet. The first approach divides the number of elementary tasks to the maximum number of available processors (in our case the maximum number of available processors was eight) while the second approach uses packets of fixed size. In this section we compare the speedup obtained using these two approaches. Figures 7.4 to 7.8 present the speedup obtained. For the situation that we use packets of fixed size we consider

cases where every active DSP gets at least one packet of work. In all these examples the packet size was ten elementary tasks.

When there are more than four active DSPs there at least two DSPs used being on the same board. As we can notice the speedup obtained when there are no DSPs used on the same board is very close to the maximum expected value while there is a drop in the speedup obtained when there are active DSPs sharing the same board. We can also notice that the maximum speedup obtained increases with the size of the problem.

The following charts show that better results are obtained by using a number of tasks equal to the number of available DSPs.

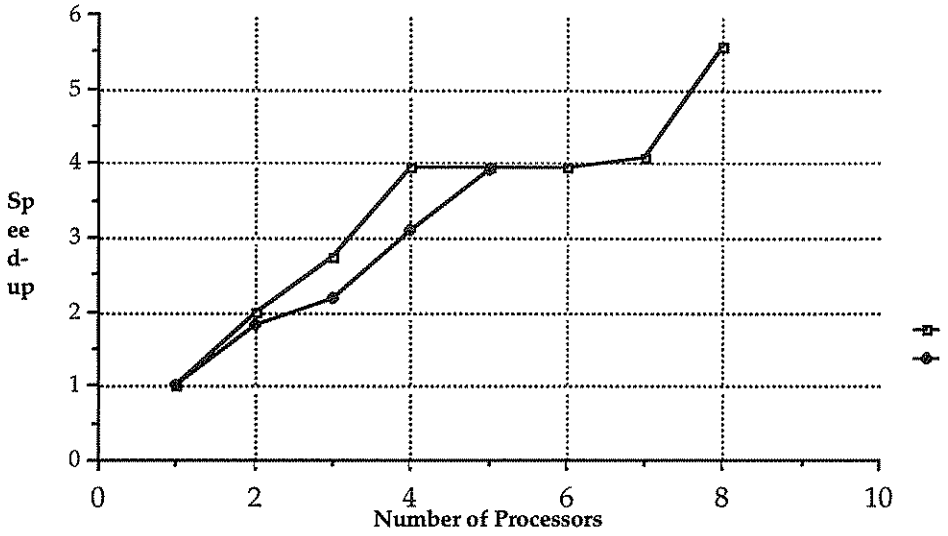
Figure 7.4: The contours contain 23 vertices each



A Size of packet equal to (total number of elementary tasks)/(total number of available DSPs)

B Fixed size packet of ten elementary tasks

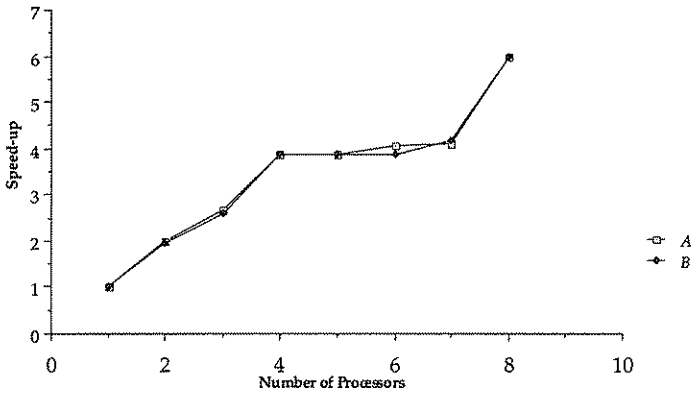
Figure 7.5: The contours contain 44 vertices each



A Size of packet equal to (total number of elementary tasks)/(total number of available DSPs)

B Fixed size packet of ten elementary tasks

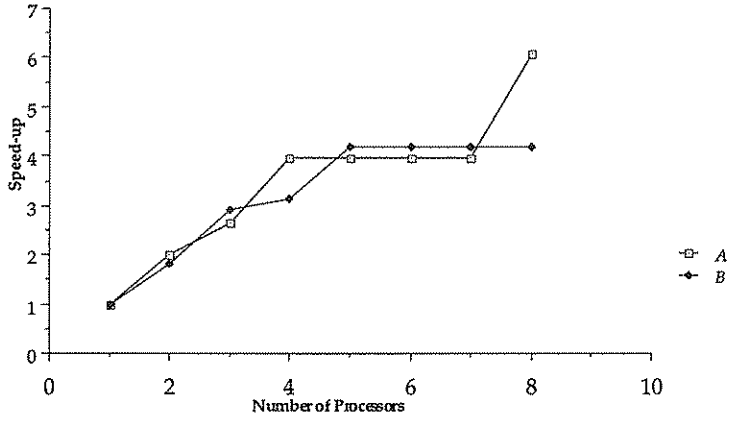
Figure 7.6: The contours contain 78 vertices each



A Size of packet equal to (total number of elementary tasks)/(total number of available DSPs)

B Fixed size packet of ten elementary tasks

Figure 7.7: The contours contain 88 vertices each



A

Size of packet equal to (total number of elementary tasks)/(total number of available DSPs)

B Fixed size packet of ten elementary tasks

7.4 Parallelisation of the 3D metamorphosis method.

The most time consuming part of the 3D metamorphosis algorithm (Section 3.6, Figure 3.8) is the part of filling of the matrix *Edge*. If the first object consists of m contours and the second object consists of n contours the method requires the filling of a $m \leftrightarrow n$ matrix. The [i, j] entry of the matrix must contain the energy needed to transform the i'th contour of the first object to the j'th contour of the second object.

The step that fills the matrix *Edge* was the part that was identified for parallelisation. The total amount of tasks was taken to be equal to the number of the available DSPs. In order to break up the total work into tasks we considered unfolding the matrix according to its rows (Figure 7.9).

In this linear representation of the matrix the index of an entry [i, j] is (i-1)m+j. If N is the number of available DSPs then the k'th task will calculate the value of the entries [i, j] that satisfy the condition:

$$\frac{mn(k-1)}{N} < (i-1)m + j \leq \frac{mnk}{N}$$

$$\begin{pmatrix} 1,1 & 1,2 & \cdots & 1,m \\ 2,1 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ n,1 & \cdots & \cdots & n,m \end{pmatrix}$$

The *Edge* matrix.

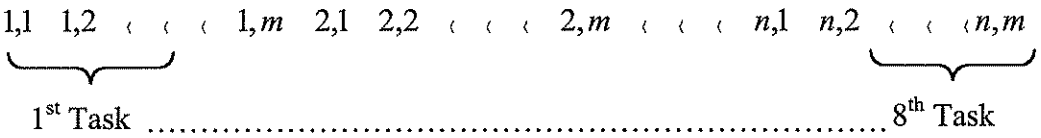


Figure 7.8: Unfolding of the matrix according to its rows.

A server must have access to the co-ordinates of the vertices to be able to calculate the energy of the contour blending for a given pair of contours. When this calculation finishes it must be also able to return its result to the host. Therefore there were routines implemented allowing the host to put the data in the frame-buffer and read the results from the frame buffer.

The RPC sending a task to the pool manager contains the following fields: the name of the first 'file' in the frame buffer keeping the data of the vertices of the first object, the name of the 'file' in the frame buffer keeping the data of the vertices of the second object, the name of the 'file' that will be used to store the results and the task number.

When all the tasks have been served the host reads the results from the frame buffer and fills the matrix *Edge* then the execution of the algorithm continues serially as in Figure 3.8.

7.4.1 Benchmarking results

Figures 7.9-7.11 present the speedup obtained with the parallelisation. When there are more than four active DSPs there at least two DSPs used being on the same board. As we can notice the speedup obtained when there are no DSPs used on the same board is very close to the maximum expected value while there is a drop in the speedup obtained when there are active DSPs sharing the same board. We can also notice that the maximum speedup obtained increases with the size of the problem.

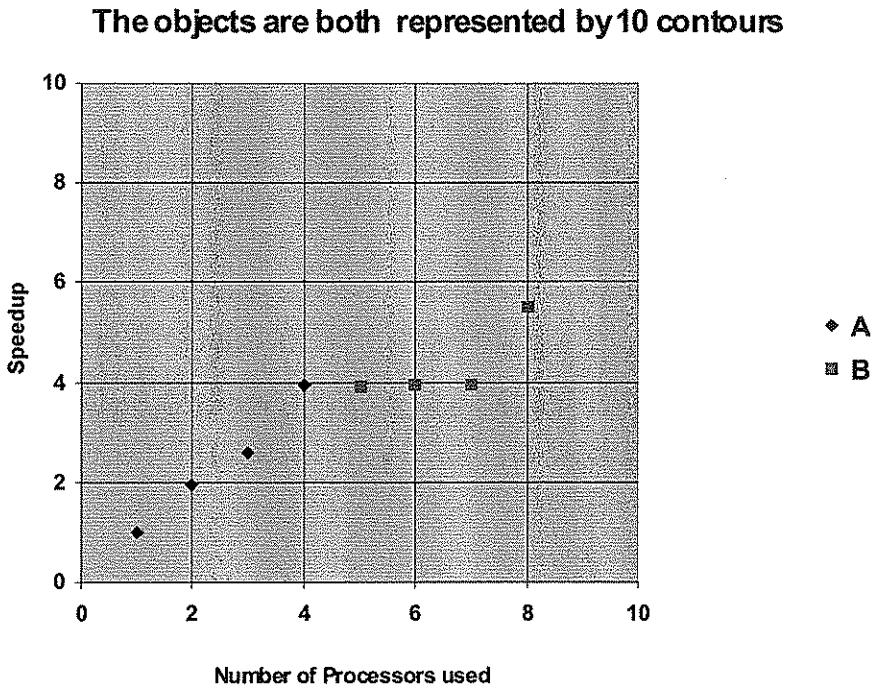


Figure 7.9: Speedup vs number of processors when both objects are represented by 10 contours.

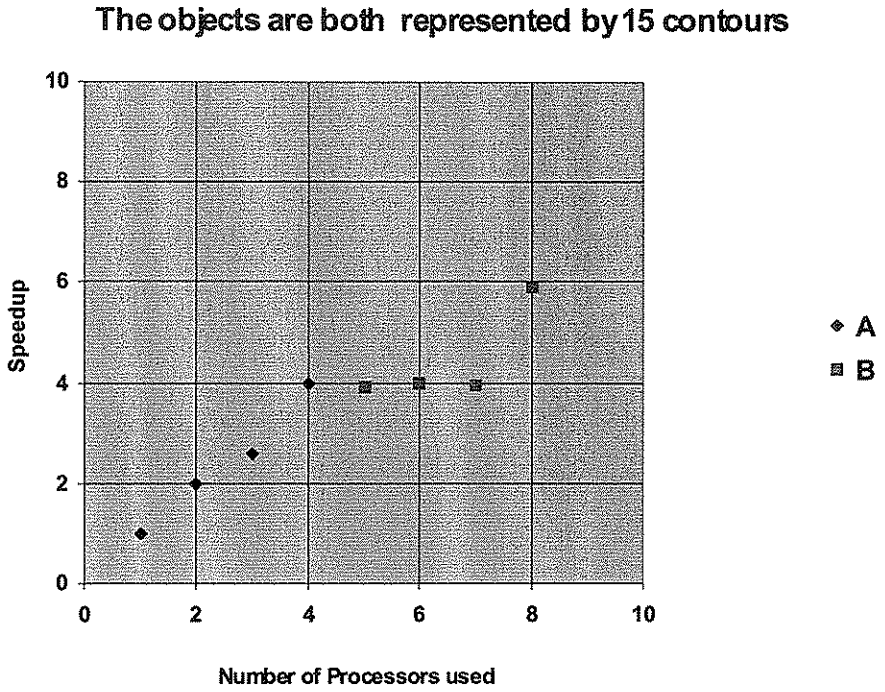


Figure 7.10: Speedup vs number of processors when both objects are represented by 15 contours.

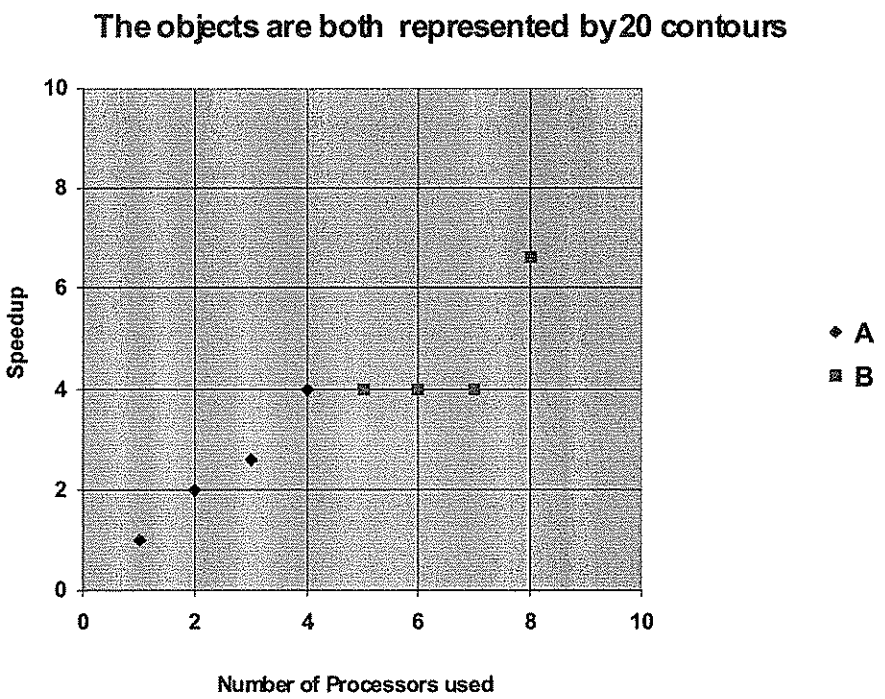


Figure 7.11: Speedup vs number of processors when both objects are represented by 20 contours.

7.5 Conclusions

In this chapter we discussed issues related to parallel processing and particularly with the application of parallel processing to computer graphics. We also described the methods employed for the parallelisation of Sederberg's and Greenwood's algorithm and the three-dimensional metamorphosis method and presented the benchmarking results obtained. The parallelisation work was implemented on the ML-PVA parallel architecture using the SAPS programming model. This served as a performance test of this parallel architecture and also as an evaluation test of the framework it operates.

Chapter 8

Conclusions and Future Work

8.1 Overview of the dissertation

The dissertation presented a method for the metamorphosis of shapes consisting of parts that can be geometrically ordered in space. This method uses the position of these parts in the ordering as a way of reducing the complexity of the correspondence problem. When the shapes are not clearly separated to a number of geometrically ordered parts we considered simplified representations of the objects.

The method was initially applied to situations of two-dimensional polygonal shapes and later to three-dimensional polygonal shapes. The performance characteristics of the algorithm were discussed and a parallelisation method and benchmarking results were presented.

8.2 Summary of Contributions

This dissertation has presented a methodology for the metamorphosis of three-dimensional polygonal objects.

The main argument of the dissertation is that there is a need for a method that can relieve the animator of the tedious task of defining a great number of corresponding components on the two objects. Furthermore, this automated process of defining correspondences should attempt to identify similarities between the two objects and couple these similarities during the interpolation step. There were methods developed for two-dimensional and three-dimensional polygonal objects.

At the introduction of Chapter 1 there were a number of objectives set for this work. In this section we will recap these objectives and discuss the degree they have been achieved.

Connected intermediate objects: The shapes of the intermediate objects are fully connected since the polygons are created by a full triangulation between every pair of successive contours. Since all the polygons of the intermediate objects are triangles they also remain planar during the animation.

Avoidance of distortion: The method proposed in this thesis uses the geometric information of the original objects, the result is that it avoids the undesirable situations where a small part of the first object transforms to a large part of the second it is incorporated into the minimisation of energy

No Self-intersection: Self intersection rarely appears. The usage of Sederberg's and Greenwood's algorithm on the contour level and the preservation of the contour ordering do not allow self intersection.

The examples presented in this thesis support the argument that the minimisation of energy succeeds in identifying the similar features of the two objects and maintains them during the animation. The user interaction is quite minimal but can be easily increased to guide the animation. It constitutes of defining constants of the material, choosing the axis of slicing and choosing anchor points on the contours.

The application of the method puts some restriction on the initial objects. The method applies to objects that have the property that there exists an axis such that the intersection of the object with any plane perpendicular to this axis is a single contour.

The method allows correspondences that lead to a variety of transformations. Transformations containing comBinations of the object scaling, translating, rotating, compressing and extending, can be potentially achieved.

A side product of this metamorphosis method was an algorithm for the automatic selection of a number of characteristic contours that describe the objects. This algorithm can be also used as a shape simplification technique.

Some time consuming tasks related with the 3D method were parallelised on the experimental ML-PVA parallel architecture. This served as a performance test of this parallel architecture and as an evaluation test of the framework it operates.

8.3 Discussion of the 3D metamorphosis method in relation with other methods

In this section we are going to compare the results of the 3D method with the methods described in Chapter 1.

The method proposed uses both geometrical and topological information and avoids distortion appearing in [BET89] and disconnected objects appearing in [HON88].

Our method is quite computationally intensive, but for most cases the correspondences can be computed at interactive rates. The time needed is comparable to the time needed by [COH98] and less that the time needed by [LER95], [DEC96] and [STA98]. The method proposed in [KAN97] solves the correspondence determination problem a lot faster.

Our method manages better control than the fully automated methods of [HUG92], [PAY92], [KEN91], [CHE89], [KAN97], [KAU91] and manages to create pleasing animations without the excessive user interaction needed by [LER95], [CHE95], [DEC96] and [STA98].

There is a restriction on the objects that the method can handle but this is not unusual for methods operating in the polygonal representation. [KEN91], [DEC94] demand that the objects are star-shaped, [LAZ97] operates on objects that are star-shaped around an axis, [KAN97] morphs objects that are homeomorphic to a disk. Some volumetric methods like [COH98] operate on objects of general topology with impressive results but on the other hand these methods do not seem capable of producing certain types of transformations, like transformations with a partial rotational component.

8.4 Further work

The 3D metamorphosis method proposed fares well when the intersection of the objects with the planes result in single contours. When the intersection of the objects with planes result in multiple contours (branching) user interaction can reduce the problem to a collection of simpler problems. Further research should be carried out in order to develop techniques that would smoothly join the different parts together during the animation. Determining automatically whether an object can be used by our method would be a very useful addition.

It would be worthwhile to extend this method so that the slicing planes are perpendicular to the line segments of a 3D polyline rather than perpendicular to a straight axis. This would allow our method to be applied to a wider category of shapes.

In all cases considered we dealt only with the changes on the object's shape. In general objects will be texture mapped and therefore we should consider how the texture of the objects would behave during the animation.

As it was mentioned, the method of the automated selection of contours necessary to describe the objects could be used as a method for shape simplification. Better results could be obtained by the usage of a more sophisticated method for the simplification of the contours on the 2D level.

All the methods developed in this thesis would be greatly assisted by the development of an intergrated user interface that would give the animator the chance to guide the animation through mechanisms as orienting, scaling and translating the shapes defining the axis for slicing, separating the shapes into parts etc.

The contour representation of a 3D polygonal object has the same problems as the related problem of how to reconstruct a surface from a set of parallel planar contours. When branching occurs ambiguities arise about how the successive contours should triangulate. But the simplification method has a certain advantage over the surface reconstruction problem: when we simplify the objects we also have the original models so even if there exist situations of multiple branching we do have the information as to how these multiple branches connect to each other. Therefore a possible extension of the simplified representation could be to incorporate this connectivity information in the contour representation which will allow this method to be applied in the simplification and subsequent reconstruction of general 3D polygonal objects.

In summary the 3D metamorphosis method that has been presented appears to give very good results for the objects for which it has been in order to be used in a commercial environment it has to be extended.

Bibliography

[AOK96] Y. Aoki, Kang Seok. Morphing of 2-D models by Fresnel transform. Proceedings of International Conference on Image Processing IEEE vol.3, 1996, pp. 727-30.

[BEI92] T. Beier, S. Neely. Feature-based image metamorphosis. Computer Graphics, Proceedings SIGGRAPH '89, pp. 35-42.

[BET89] E. Bethel, S. Uselton. Shape Distortion in Computer - Assisted keyframe Animation. In State of the Art in Computer Animation. Magnenat Thalmann N. and Thalmann, D., eds., Springer Verlag, New York, 1989, pp. 215-224.

[CAR97] E. Carmel, D. Cohen. Warp-guided object-space morphing. Visual Computer, vol.13, no.9-10, 1997, pp. 465-78.

[CAT78] E. Catmull. The problems of computer Assisted animation. Proceedings ACM SIGGRAPH vol 12, no 3, August 1978.

[CHE89] E. Chen, R. Parent. Shape Averaging and Its Applications to Industrial Design. IEEE Computer Graphics and Applications vol. 9, no. 1, January 1989, pp. 47-54.

[CHE95] D. T. Chen, A. State, D. Banks. Interactive shape metamorphosis. Proceedings 1995 Symposium on Interactive 3D Graphics. ACM. 1995, pp. 43-4.

[CHE96] M. Chen, M. W. Jones, P. Townsend. Volume distortion and morphing using disk fields. Computers & Graphics, vol.20, no.4, July-Aug. 1996, pp. 567-75.

[CHU94] Chun-Hsiung Chuan. Kuo C-CJ. Contour metamorphosis using the wavelet descriptor. Proceedings of Spie - the International Society for Optical Engineering, vol.2182, 1994, pp. 288-99.

[COH96] D. Cohen-Or Levin D. Solomovici A. Contour blending using warp-guided distance field interpolation. Proceedings. Visualization '96 (IEEE Cat. No.96CB36006). ACM. 1996, pp.165-72.

[COH98] D. Cohen-Or Levin D. Solomovici A. Three-dimensional distance field metamorphosis. ACM Transactions on Graphics, vol.17, no.2, April 1998, pp.116-41.

[DEC94] P. Decaudin, A. Gagalowicz. Fusion of 3D shapes. The 5th Eurographics Workshop on Animation and Simulation, Oslo Eurographics Association 1994, pp.1-14.

[DEC96] D. De Carlo, J. Gallier. Topological evolution of surfaces. Procceedings of Graphics Interface '96, pp. 194-203.

[FUC82] H. Fuchs, Z. M. Kedem, S. P. Uselton. Optimal surface reconstruction from planar contours. Computer Graphics proceedings SIGGRAPH vol. 16, no. 3, 1982, pp. 69-75.

[GAL96] E. Galin, S. Akkouche. Blob metamorphosis based on Minkowski sums. Blackwell Publishers for Eurographics Assoc. Computer Graphics Forum, vol.15, no.3, 1996, pp.143-53.

[GOL96] E. Goldstein, C. Gotman. Polygon morphing using a multiresolution representation. Proceedings of Graphics Interface '96, pp. 50-60.

[GUI94] L. Guibas, J. Hershberger. Morphing simple polygons. Proceedings of the Tenth Annual Symposium on Computational Geometry. ACM Press. 1994, pp.267-76.

[HAS97] A. E. Hassanien, M. Nakajima. Image morphing with snake model and thin plate spline interpolation. SPIE-Int. Soc. Opt. Eng. Proceedings of Spie - the International Society for Optical Engineering, vol.3229, 1997, pp.407-16.

[HAS98] A. E. Hassanien, M. Nakajima. Image morphing of facial images transformation based on Navier elastic body splines. Proceedings Computer Animation '98 (Cat. No.98EX169). IEEE Comput. Soc. 1998, pp.119-25.

[HON88] T. Hong, N. Magnemat-Thalmann, D. Thalmann. A General Algorithm for 3-D Shape Interpolation in a Facet-Based Representation. Proceedings of Graphics Interface '88 (June 1988) pp. 229-235.

[HUG92] J. F. Hughes. Sheduled Fourier Volume Morphing. Computer Graphics, proceedings SIGGRAPH '92, vol. 26, no. 2, 1992, pp. 43-46.

[KAN97] T. Kanai, H. Suzuki, F. Kimura F. 3D geometric metamorphosis based on harmonic map. Proceedings The Fifth Pacific Conference on Computer Graphics and Applications (Cat. No.97TB100206). IEEE Comput. Soc. 1997, pp.97-104.

[KAU91] A. Kaul, J. Rossignac. Solid-Interpolating Deformations: Construction and Animation of PIPs. Proceedings of Eurographics '91.

[KEN91] J. Kent, R. Parent, W. Carlson. Establishing Correspondences by Topological Merging: A new Approach to 3-D Shape Transformation. Proceedings of Graphics Interface '91 (Calgary, Alberta, June 1991) pp. 271-278.

[KEN92] J. Kent, R. Parent and W. Carlson. Shape Transformation for Polyhedral objects. Proceedings of SIGGRAPH '92 (Chicago, July 26-31, 1992) pp. 47-54.

[KOC84] D. Kochanek, R. Bartels. Interpolating Splines with Local tension, Continuity and Bias Control. Proceedings of SIGGRAPH '84 (Minneapolis Minnesota July 23-27, 1984) In Computer Graphics vol. 18, no. 3 (July 1984) pp. 33-41.

[LAZ97] F. Lazarus, A. Verroust. Metamorphosis of cylinder-like objects. Journal of Visualization & Computer Animation, vol.8, no.3, July-Sept. 1997, pp.131-46.

[LER95] A. Lerios, C. D. Garfinkle, M. Levoy. Feature-based volume metamorphosis. Computer Graphics Proceedings. SIGGRAPH 95. ACM. 1995, pp.449-56.

[LOR87] W. E. Lorensen and H. E. Cline, Marching Cubes, A High Resolution 3D Surface Construction Algorithm, Computer Graphics Proceedings. SIGGRAPH 87, pp. 163-69.

[MCH90] J. MchuGh. Algorithmic graph theory. Prentice Hall International editions 1990.

[MIN88] E. Minieca. Optimization Algorithms for networks and graphs Marcel Dekker Inc. 1988.

[PAR91] R. Parent. Shape transformation by boundary representation interpolation: a recursive approach to establishing face correspondences. Journal of Visualizaton and Computer Animation. vol 3 issue 4, 1992, pp. 219-239.

[PAY92] B. Payne, N. Toga. A distance Field Manipulation of Surface Models. IEEE Computer Graphics and Applications vol. 12, no. 1, Jan. 1992, pp. 65-71.

[REE81] W. Reeves. Inbetweening for Computer Animation Utilizing Moving Point Constraints. Proceedings of SIGGRAPH '81 (Dallas , Texas, August 3-7, 1981), In Computer graphics vol. 15, no. 3 (August 1981) pp. 263-269.

[SED92] T. W. Sederberg, E. Greenwood. A Physically Based Approach to 2-d Shape Blending. Computer Graphics (Proc. SIGGRAPH) 26(2): 1992 pp. 25-34.

[SED93] T. W. Sederberg, Gao Peisheng, Wang Guojin and Hong Mu. 2-D Shape Blending :An Intrinsic Solution to the Vertex Path Problem. Proceedings SIGGRAPH '93 pp. 15-18.

[SEU94] Seung-Yong Lee. Kyung-Yong Chwa. Hahn J. Sung Yong Shin. Image morphing using deformable surfaces. Proceedings of Computer Animation '94. IEEE Comput. Soc. Press. 1994, pp. 31-9.

[SEU95] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, G.Wolberg. Image metamorphosis using snakes and free-form deformations. Computer Graphics Proceedings. SIGGRAPH 95. ACM. 1995, pp. 439-48.

[SEU96] Seung-Yong Lee, G. Wolberg, Kyung-Yong Chwa. Sung Yong Shin. Image metamorphosis with scattered feature constraints. IEEE Transactions on Visualization & Computer Graphics, vol.2, no.4, Dec. 1996, pp. 337-54.

[SEU98] Seung-Yong Lee, G. Wolberg, Sung Yong Shin. Polymorph: morphing among multiple images. IEEE Computer Graphics & Applications, vol.18, no.1, Jan.-Feb. 1998, pp. 58-71.

[SHA95] M. Shapira, A. Rappoport. Shape blending using the star-skeleton representation. IEEE Computer Graphics & Applications, vol.15, no.2, March 1995, pp.44-50.

[STA98] A. State, M. C. Lin, D. Manocha, M. A. Livingston. Feature-based surface decomposition for correspondence and morphing between polyhedra. Proceedings Computer Animation '98 (Cat. No. 98EX169). IEEE Comput. Soc. 1998, pp.64-71.

[STE85] S. Steketee, N. Badler. Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and phrasing Control. Proceedings of SIGGRAPH '85 (San Fransisco, California, July 22-26, 1985), pp. 255-262.

[WAT92] A. Watt, M. Watt. Advanced Animation and Rendering Techniqes theory and practice. Addison-Wesley 1992.

[WOL88] G. Wolberg. Image warping among planar shapes. New Trends in Computer Graphics. Proceedings in Computer graphics '88 pp. 209-218.

[WOL89] G. Wolberg. Skeleton Based Image Warping. Visual Computer, Volume 5, Number 1/2 March 1989 pp. 95-108.

[WYV86] B. Wyvill, C. McPheeters and G. Wyvill. Data structure for Soft Object. Visual Computer Volume 2, pp. 227-34.

[YUE95] Yue Man Sun, Wenping Wang, F. Y. L. Chin. Interpolating polyhedral models using intrinsic shape parameters. Proceedings of the

Third Pacific Conference on Computer Graphics and Applications Pacific Graphics '95. Computer Graphics and Applications. World Scientific. 1995, pp.133-47.

[YUE96] Yuefeng Zhang. A fuzzy approach to digital image warping. IEEE Computer Graphics & Applications, vol.16, no.4, July 1996, pp.34-41.