FEATURE-BASED 3D MORPHING

Η ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

υποβάλλεται στην
ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής Εξεταστική Επιτροπή

από τον

Αθανασιάδη Θεόδωρο

ως μέρος των Υποχρεώσεων για τη λήψη του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ
ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ
ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιανουάριος 2009

Dedication

To my parents, Sotiri and Fotini,
and my sister, Amalia.

Acknowledgements

I would like to thank my supervisor Professor Ioannis Fudos for his valuable help, his advice and the patience he has shown during the elaboration of this work.

I would also like to thank Professor Christophoros Nikou for his help, suggestions and advice.

Furthermore, I would like to thank my friends Iraklis Goudas and Vicky Stamati for their help and valuable advice.

Table of Contents

List of Figures

List of Tables

List of Algorithms

GLOSSARY

$A \cdot B$,$A \times B$	dot product of vectors, cross product of vectors
$\|\|X\|\|$	magnitude of vector X
sin^{-1},cos^{-1},tan^{-1}	inverse sine, inverse cosine and inverse tangent functions
X^T	vector transpose
$\frac{\partial P}{\partial u}$	first-order partial derivative of P with respect to u component

Abstract

Theodoros Athanasiadis.
MSc, Computer Science Department, University of Ioannina, Greece. January 2009.
Supervisor : Ioannis Fudos.

3D morphing is the technique of smooth transition between two (or more)solid objects. Animation using deforming objects is frequently used in computer graphics for entertainment. Furthermore, smooth animation of nonrigid objects (e.g. articulated objects) can be accomplished by 3D morphing on a set of object snapshots.

We present an approach to 3D morphing of arbitrary genus-0 polyhedral objects. The technique is based on a sphere mapping process that maintains the correspondence among the initial polygons and the mapped ones and preserves topology and connectivity. The method works well for all genus-0 objects without any user intervention. Finally, we present a fully automated feature-based technique that matches surface areas (features) with similar morphological characteristics between the two morphed objects and performs morphing according to this feature correspondence list. Alignment is achieved without user intervention and is based on pattern matching between the feature connectivity graphs of the two morphed objects. Features points are then detected and matched by using concavity and convexity extrema on each solid.

Εκτεταμενη Περιληψη στα Ελληνικα

θεόδωρος Αθανασιάδης του Σωτηρίου και της Φωτεινής.
MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιανουάριος 2009.
3Δ Μετασχηματισμός Μορφών Αντικειμένων βασισμένος σε ταίριασμα χαρακτηριστικών.
Επιβλέποντας: Ιωάννης Φούντος.

Ο 3Δ Μετασχηματισμός Μορφών (Morphing) είναι μία τεχνική δημιουργίας μιας ομαλής μετάβασης μεταξύ δύο ή περισσοτέρων αντικειμένων. Το animation με μετασχηματιζόμενα αντικείμενα χρησιμοποιείται συχνά στα γραφικά. Επιπλέον, ο 3Δ Μετασχηματισμός Μορφών μπορεί να χρησιμοποιηθεί και για την παραγωγή animation μετασχηματίζοντας διάφορα στιγμιότυπα του ίδιου αντικειμένου.

Σε αυτή την εργασία θα παρουσιάσουμε μία μέθοδο για τον 3Δ Μετασχηματισμό Μορφών ανάμεσα σε δύο αντικείμενα genus-0 με διαφορετική τοπολογία. Η τεχνική βασίζεται αρχικά σε μία διαδικασία προβολής πάνω στη μοναδιαία σφαίρα η οποία διατηρεί την τοπολογία και την συνδεσιμότητα των αρχικών πολυγώνων των μοντέλων, ενώ παράλληλα στοχεύει στην ελαχιστοποίηση της παραμόρφωσης των προβολών. Στη συνέχεια οι προβολές συνδυάζονται για την δημιουργία μίας κοινής τοπολογίας και τέλος παρεμβάλλονται με χρήση γραμμικής παρεμβολής με στόχο την δημιουργία μίας σειράς από ενδιάμεσα μετασχηματιζόμενα μοντέλα.

Η μέθοδος δουλεύει ικανοποιητικά για όλα τα genus-0 αντικείμενα χωρίς την παρέμβαση του χρήστη. Επίσης, θα παρουσιάσουμε μία πλήρως αυτόματοποιημένη μεθοδολογία βασισμένη στα χαρακτηριστικά των μοντέλων που αντιστοιχίζει περιοχές με παρόμοια μορφολογικά χαρακτηριστικά ανάμεσα στα δύο αντικείμενα.

Αρχικά, γίνεται κατάτμηση των μοντέλων σε περιοχές σημαντικών χαρακτηριστικών από τις οποίες κατασκευάζονται γράφοι συνδεσιμότητας των χαρακτηριστικών. Στη συνέχεια, γίνεται ευθυγράμμιση χωρίς την παρέμβαση του χρήστη και βασίζεται στην αντιστοίχηση προτύπων μεταξύ των γράφων συνδεσιμότητας των χαρακτηριστικών των δύο αντικειμένων. Από τις περιοχές που αντιστοιχήθηκαν εντοπίζονται σύνολα χαρακτηριστικών σημείων και αντιστοιχίζονται χρησιμοποιώντας τα τοπικά ακρότατα μιας συνάρτησης που βασίζεται στην κυρτότητα κάθε αντικειμένου. Στόχος της ευθυγράμμισης και της αντιστοίχησης των χαρακτηριστικών είναι η επιπλέον βελτίωση του τελικού οπτικού αποτελέσματος.

Τέλος, περιγράφουμε και συγκρίνουμε τα αποτελέσματα της υλοποίησης της μεθοδολογίας σε ένα σύνολο από μοντέλα με διαφορετική μορφολογία.

Chapter 1

Introduction

1.1 Introduction

Image morphing is a popular technique for creating a smooth transition between two images. It is a special effect used primarily in motion pictures and animation that transforms (morphs) one image into another through a seamless morph sequence. The simplest method for changing an image into another is to cross-fade between the two images. Other approaches use physics based analogs such as the 2D particle system where pixels of one image are mapped onto pixels of the other [24]. There is a class of methods for image morphing that involve image warping around regular (e.g. a sphere of a cylinder) or non regular (e.g. a free-form surface) shapes [22, 24]. More advanced methods include morphing based upon fields of influence surrounding two-dimensional control primitives often called features [4].

Shape morphing is a technique that aims to generate a smooth sequence that transforms a source shape into a target shape. Although we have some quite efficient and effective methods for 2D morphing, the 3D case remains an open problem both in terms of feasibility and efficiency.

Existing methods for 3D morphing can be categorized into two broad classes: *volume based* or *voxel based* [19] and *mesh based* or *structural* [14] approaches. The volume-based approach represents a 3D object as a set of voxels usually leading in computationally intensive computations. The mesh-based approach exhibits better results in terms of boundary smoothness and rendering since the intermediate morphs are represented as volumes and techniques such as marching cube [21] are employed to acquire the final polygonal representation used for rendering. Furthermore, most applications in graphics use mesh-based representations, thus making mesh-based modeling more broadly applicable. However, volume-based methods surpass the mesh based ones in that they can handle the morphing of very different topologies more easily, since volume to volume morphing is a lot similar to image morphing by means of treating voxels instead of pixels.

Although mesh morphing is more efficient as compared to volume-based morphing, it requires a considerable preprocessing of the two considered objects. Mesh morphing involves two steps. The first step establishes a mapping between the source and the target object (correspondence problem), which requires that both models are meshed isomorphically with a one-to-one correspondence. The second step involves finding suitable paths for each vertex connecting the initial position to the final position in the merged mesh (interpolation problem). For performing structural morphing, we can use *boundary representation (Brep)* or *surface representation* in which we represent each object by its surface description, or *volumetric or solid meshes*, for instance tetrahedral representations. In volumetric mesh morphing, it is much easier to maintain robustness and avoid the folding phenomenon. However, volumetric mesh morphing is computationally expensive as compared to surface mesh morphing since the number of elements in the former case is much larger in comparison to the latter case.

The method proposed in this work is based on surface mesh morphing. It introduces a sound and complete approach to morphing between any two genus-0 objects. Recall that genus-0 objects are by definition homeomorphic to the sphere. Our approach builds on a spherical mapping approach presented in [7] for the purpose of parameterization of closed surfaces. Our mapping works in two phases. In the first phase, we perform an initial mapping. In the second phase, we optimize the mapping to achieve a better placement under specific geometric criteria and under a set of topological constraints. For the first phase, we present a much faster alternative to [7] based on Laplacian smoothing and adapt the second phase accordingly to capture morphing related requirements. We also present an improvement of this approach that takes into consideration 3D features and derives a feature correspondence set to improve the final visual effect. This is a very important characteristic for similar objects, as in the case of morphing between two articulated human representations. Object alignment, feature detection and feature point matching is performed automatically without user intervention.

1.2 Background Material

1.2.1 Mesh Parameterization

Given any two surfaces with similar topology it is possible to compute a one-to-one mapping between them. If one of these surfaces is represented by a triangular mesh, the problem of computing such a mapping is referred to as mesh parameterization. The surface that the mesh is mapped to is typically refered to as the parameter domain. The purpose of mesh parameterization is to obtain a piecewise linear map, associating each triangle of the original mesh with a triangle of a domain. An important goal of parameterization is to obtain *bijective (invertible)* maps, where each point on the domain corresponds to exactly one point of the mesh. The geometric shape of the domain triangles will typically be different than the shape of the original triangles, resulting in angle and area distortion. The distortion is an important factor of the parameterization and applications typically try to minimize the distortion for the whole mesh. Maps that minimize the angular distortion are called *conformal* and maps that minimize area distortion are called *authalic.* Mesh parameterizations have numerous applications in computer graphics such as in Morphing, Mesh Completion, Remeshing, Surface fitting and Texture Mapping.

1.2.2 Non Linear Optimization

In mathematics, nonlinear optimization is the process of solving a system of equalities and inequalities, collectively termed constraints, over a set of unknown real variables, along with an objective function to be maximized or minimized, where some of the constraints are nonlinear. More specifically in the context of this work we will deal with continuous nonlinear problems of the following form:

$$\min_x f(x) \tag{1.1}$$

$$\text{s.t. } g^L \leq g(x) \leq g^U \tag{1.2}$$

$$x^L \leq x \leq x^U \tag{1.3}$$

where the $x \in R^n$ are the optimization variables (possibly with lower and upper bounds, $x^L \in (R \cup -\infty)^n$ and $x^U \in (R \cup +\infty)^n$). The function $f : R^n \to R$ is called the *objective function.* A vector x satisfying all the constraints of 1.2 is called a *feasible solution* to the problem. The collection of all such points forms the *feasible region.* The Non Linear Problem (NLP) is to find a feasible point x^* such that $f(x) \geq f(x^*)$ for each feasible point x.

For solving optimization problems like the above there are two categories of algorithms, *global* and *local* optimization methods. Local optimization means that the method attempts to find a *local* minimum, and there is no guarantee that you will get the global minimum for the problem, while global optimization methods try to find the *global* minimum of the *objective function.* In some cases the *local* minimum found is in fact the *global* minimum (convex problems).

Local optimization algorithms generally depend on derivatives of the *objective function* and constraints (gradients and hessians) to aid in the search. For this reason it is useful the function to be real-valued and twice continuously differentiable. There are ways to tackle this strict requirement, but then there is no guarantee that the solver will find a solution. Local optimization also depends on the initial values of the variables, the better the initial values are the faster the solver will converge to a solution. Examples of *local* optimization methods are the *Sequential Quadratic Programming* (SQP) method which is a generalization of Newtons's method for unconstrained optimization, the *augmented Lagrangian* method and the *Interior Point* method.

Global optimization algorithms try to find the best set of parameters to minimize the *objective function.* In general, there can be many solutions that are locally optimal but not globally optimal. Consequently, global optimization problems are typically quite difficult to solve exactly and most methods incorporate probabilistic (random) elements in the algorithms (through random parameter values, etc). More modern methods employ strategies aiming to search the search space in a more intelligent way (*Genetic algorithms*). For *global* optimization there are several algorithms, some known types are *Simulated Annealing*, *Tabu Search*, *Genetic Algorithms* and *Branch and Bound Algorithms.*

1.2.3 Laplacian Smoothing

In many applications, like the finite element method mesh quality affects numerical stability as well solution accuracy. Therefore, the quality of the mesh triangles is an important factor. There are various existing mesh improvement methods that can roughly be classified into two categories, methods that use topological modifications performed by inserting or removing nodes as well as local reconnection of nodes, and smoothing methods based on relocating existing nodes.

Amongst the second category, the Laplacian smoothing method, has gained popularity due to its simplicity and efficiency. Laplacian smoothing is the most commonly used and straightforward method for mesh smoothing. It simply moves each node to the centroid of the polygon formed by its adjacent nodes. It is a local smoothing algorithm because, in each step, the movement of a node is calculated by using the locations of its adjacent nodes only. However, Laplacian smoothing sometimes can lead to low quality or invalid elements as well as deformation and shrinkage in the case of surface meshes.

In Laplacian smoothing, we can consider a mesh as a spring system. Each edge connecting the central node with its neighboring node can be seen as a linear-spring. Let v_i be the vector from the central node v (x,y) to the i th neighboring node: $v_i = (x_i - x, y_i - y)$. The sum of the spring forces acting on the central node is: $F = K \sum_{i=1}^{k} v_i$ where K is the spring constant, and k is the number of neighboring nodes. When the central node is located exactly at the geometric center of the polygon, the spring forces are balanced out and the spring system is in equilibrium. Therefore Laplacian smoothing can be considered as an iterative way to find this force-balancing state. Below we can summarize the advantages and disadvantages of Laplacian smoothing:

Advantages:

- Computational efficiency
- Easy implementation

Disadvantages:

- Does not always move the node to the optimal position to get the best element quality
- Generation of inverted elements
- Tends to lose element size uniformity if iterated many times
- Tends to yield lower quality elements if iterated more than a few times

Therefore, modified methods have been proposed to circumvent these problems [29].

1.2.4 Spherical Triangular Area and Angles

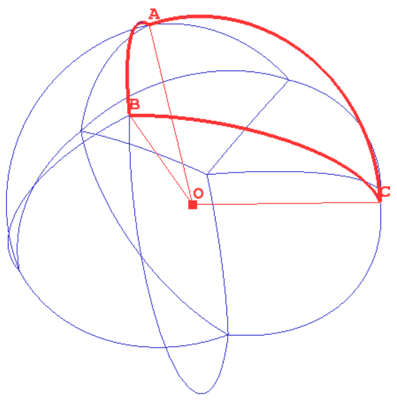


Figure 1.1: A spherical triangle with angles A,B and C on the unit sphere.

A spherical triangle is formed on the surface of a sphere by three circular arcs intersecting pairwise in three vertices. The spherical triangle is the spherical analog of the planar triangle. Let a spherical triangle have angles A,B and C (measured in radians at the vertices along the surface of the sphere) and let the sphere which the spherical triangle sits have radius R. Then the surface area of the spherical triangle is:

$$area(f) = R^2[(A + B + C) - \pi] = R^2 E \tag{1.4}$$

where E is called the spherical excess. The angles A,B and C are dihedral angles between two planes, so that the dihedral angle between planes AOB and AOC is denoted A, the dihedral angle between planes BOC and AOB is denoted B and the dihedral angle between BOC and AOC is denoted C. The dihedral angle between two planes can be calculated using the dot product of the normals of the corresponding planes as shown in equation 1.5 for a spherical triangle with vertices v_0,v_1 and v_2.

$$\begin{aligned} A &= cos^{-1}\left(\left(\frac{v_2 \times v_0}{||v_2 \times v_0||}\right) \cdot \left(\frac{v_1 \times v_0}{||v_1 \times v_0||}\right)\right) \\ B &= cos^{-1}\left(\left(\frac{v_0 \times v_1}{||v_0 \times v_1||}\right) \cdot \left(\frac{v_2 \times v_1}{||v_2 \times v_1||}\right)\right) \\ C &= cos^{-1}\left(\left(\frac{v_0 \times v_2}{||v_0 \times v_2||}\right) \cdot \left(\frac{v_1 \times v_2}{||v_1 \times v_2||}\right)\right) \end{aligned} \tag{1.5}$$

1.3 Structure of this thesis

The rest of chapter 1 presents related work on 3D morphing. Chapter 2 presents the mapping step of our approach and briefly describes the efficient computation of the intersections among the polygons on the sphere and the calculation of the interpolation trajectory. Chapter 3 presents an alternative mapping method that can be applied to one of the morphed objects based on the mapping of the other object and a feature correspondence list of the two solid representations. Chapter 4 contains implementation details and explains chalenges faced during the development of the software. Chapter 5 presents an experimental evaluation of our method and some visual morphing examples which are part of the video that accompanies this work. Finally, chapter 6 provides conclusions.

1.4 Related Work

Most surface-based mesh morphing techniques employ a merging strategy to obtain the correspondence between the vertices of the input model. The merging strategy may be either automatic or user specified. Kent et al. [14] proposed an algorithm for the morphing of two objects topologically equivalent to the sphere. The algorithm works in two steps, first the two objects are mapped to a sphere and then the two projected topologies are merged. A common topology suitable for interpolation is created. The mapping presented is accomplished by a mere projection to the sphere and thus is applicable solely to star shaped objects.

The main problem with 3D parameterization techniques like [14] is how to find an appropriate mapping over the unit sphere for each of the morphed objects. Several techniques have been proposed to overcome this limitation inspired by physics. In [12] the authors use a spring system to model the mesh and gradually force the mesh to expand or shrink on the unit sphere by applying a force field. Our method uses a similar technique for determining an initial mapping over the unit sphere. Methods using springs do not always produce acceptable mappings especially when handling complex non convex objects. We overcome this problem successfully in this work.

[2, 3, 36] use a spring-like relaxation process. The relaxation solution may collapse to a point, or experience foldovers, depending on the initial state. Several heuristics achieving convergence to a valid solution are used.

[28, 25, 11] describe methods to generate a provably bijective parameterization of a closed genus-0 mesh to the unit sphere. The projection involves the solution of a large system of non-linear equations. A set of constraints on the spherical angles is maintained to achieve a valid spherical triangulation.

[27] uses a polyhedron realization algorithm that can transform any general polyhedron into a convex one which is isomorphic to the original. The realization consists of two phases, simplification and re-attachment. During the simplification phase, low valence vertices are detached from the vertex-neighborhood graph of the polyhedron one by one,

and the corresponding graph is re-triangulated. This step is repeated until a 4-clique results. The second phase starts by first creating a tetrahedron and then the vertices are re-attached to the polyhedron, in the reverse order of their detachment, while maintaining the polyhedrons convexity.

[23] presents a similar method that first simplifies the surface mesh to a tetrahedron while creating a progressive mesh favoring triangles with good aspect ratio and then in similar way reattaches the vertices and simultaneously optimizes positions of the embedded vertices. The positions of the vertices are optimized to minimize a stretch metric.

[26] presents a method that directly create and optimize a continuous map between the meshes instead of using a simpler intermediate domain to compose parametrizations. Progressive refinement is used to robustly create and optimize the inter-surface map. The refinement minimizes a distortion metric on both meshes.

[15] also presents a method that relies on mesh refinement to establish a mapping between the models. First a mapping between patches over base mesh domains is computed and then mesh refinement is used to find a bijective parameterization. One advantage of this approach is that it naturally supports feature correspondence, since feature vertices are required as user input for the initial patch mapping.

[18] uses reeb-graphs and boolean operations to extend spherical parameterization for handling models of arbitrary genus. Each genus-n model is represented as a genus-0 positive mesh and n genus-0 negative meshes. Therefore $n + 1$ spheres are required to parameterize these $n + 1$ meshes independently, and thus to accomplish the spherical parameterization of genus-n models. Once a consistent embedding is computed for each model the positive meshes and the negative sets are paired. In the case where the number of negative meshes is not equal in the two models, extra pseudo negative meshes are generated to have an equal number of paired negative meshes. For each pair of meshes the morphing sequence is computed independently. Finally, boolean difference operation is applied to subtract each intermediate negative object from an intermediate positive object to obtain the morphing sequence. Existing methods for producing valid spherical embeddings of genus-0 models can be integrated into their framework.

Another method that use reeb-graphs for morphing topologically different objects of arbitrary genus is [13]. The method specifies the correspondence between the input models by using graph isomorphic theory. The super Reeb graph, which has the equivalent topological information to the Reeb graphs of the two input objects, is constructed and used to conduct the morphing sequence.

[17, 20] provide efficient techniques for morphing 3D polyhedral objects of genus-0. The emphasis of the method is on efficiency and requires definition of feature patches to perform 2D mapping and subsequent merging. Their method does not avoid self intersection and requires embedding merging and user intervention for mapping. Our method overcomes these shortcomings in expense of a considerable preprocessing time for mapping.

An interesting work for volume morphing is based on wavelets is presented in [10].

This is a promising approach whose principle could be applied to surface based morphing. This volume morphing technique yields rather slow algorithm which have time complexity $\Omega(n^3)$ where n is the size of the size of the volume representation.

The method presented in this work overcomes these limitations and allows for a totally automated and appropriate for morphing mapping of an object of genus-0 surface into a 2D space with spherical topology. An initial mapping over the unit sphere is computed and used as an initial state and the mapping is then improved by nonlinear optimization. For smoother morphing that takes advantage of object morphology we introduce a feature-based approach. Feature correspondence is performed automatically without any user intervention.

Chapter 2

Spherical Mesh Mapping

2.1 Introduction

The first step for Morphing arbitrary *genus-0* objects is to find a mapping into a common spherical domain for each object. The aim of the parameterization process is to minimize the distortion of the mapping for the Morphing purposes. In the context of this work this problem is solved by nonlinear optimization. More specifically, the parametrization (*embedding*) of the objects topology in the surface of the unit sphere, can be posed as a constrained optimization problem.

The *variables* of the optimization problem are the coordinates of all vertices. In addition, there are two kinds of *constraints* on the variables. First, the Euclidean norm of the coordinates for any vertex must be 1. This implies that every vertex must lie on the unit sphere during the optimization. Second, the orientation for each facet must remain the same during the optimization. This second constraint guarantees that the final mapping will not contain folded elements provided that the initial mapping did not contain any either. Finally, the *objective* function to be minimized will target on minimizing the distortion of the mapping or matching features in the two objects as described in later sections.

The variables in the optimization are the positions on the unit sphere to which the vertices are mapped. Therefore, the *starting values* are an initial mapping of the object. Consequently, it is important for the optimization process that the initial positions to be a valid solution to the optimization problem. Therefore, the initial mapping should not contain folded elements to satisfy the second constraint. In addition, the initial mapping is helpful to be close to the optimized solution to achieve faster convergence.

2.2 Topology Preserving Sphere Mapping

2.2.1 Initial Mapping

The initial parameterization in Brechbuhler's algorithm [7], which is actually an initial mapping of bounding voxels, is performed in polar coordinates. Two polar coordinates θ and ϕ are determined for all vertices in two steps. Two vertices are selected as the poles (north and south) for this process. The poles must not be too close as this results in a poor initial parameterization. [7] suggests selecting the poles based on the z coordinate in object space and which reasonable due to the fact that they deal only with voxel objects. Instead, we choose as poles the vertex pair with the largest distance between them (diameter of the solid).

The first step of the parameterization involves the calculation of the latitude θ. The latitude should grow smoothly from 0 at the north pole to π at the south pole. The continuous problem is formulated as Laplace's equation $\nabla^2\theta = 0$ (except the poles), with Derichlet conditions $\theta_{north} = 0$, $\theta_{south} = \pi$. The Laplacian is approximated by finite second differences of the available direct neighbors, which in our case implies that every node's latitude (except the poles) must equal the average of its neighbors latitudes. These conditions form a sparse set of linear equations, which can be written in the form $A'\theta' = b'$ where A' is an $n_{vert} \times n_{vert}$ matrix, $\theta' = (\theta_0, \dots, \theta_{n_{vert}-1})^T$ and b' is an n_{vert} vector of constants. The border conditions $\theta_0 = \theta_{north}$ and $\theta_{n_{vert}-1} = \theta_{south}$ supply two equations and results in the reduced $n \times n$ system $A\theta = b$, where $n = n_{vert} - 2$, $A = (a_{1,1}, a_{1,2}, \dots, a_{n,n})$ is symmetric and $\theta = (\theta_1, \dots, \theta_n)^T$. The algorithm that set up the matrix A and the vector b is summarized in Figure 1.

A physical analogy is heat conduction where the south pole has temperature π and the north pole has temperature 0. We then seek the stationary temperature distribution on a heat-conducting surface. The problem is formulated by a sparse set of linear equations. Figure 2.1 shows an example of the application of the thermal conduction on the frog model from [1].

The second step of the parameterization involves the calculation of the longitude ϕ. Unlike latitude, longitude is a cyclic parameter, when we walk around a sphere counter-clockwise (seen from the north), longitude keeps increasing monotonically all the time, but there must be a place where longitude leaps back by 2π. Therefore, a discontinuity spherical path must be established from pole to pole, the choice of the path is immaterial since it just has to connect the two poles. The path is built incrementally in a greedy manner with steepest latitude ascent for each of its nodes (see Figure 2). Consequently, the new system of linear equations is very similar to that of the latitude and the algorithm used to modify the matrices A and b is illustrated in Figure 2.

```
Input: Polyhedral representation for Model $M$
Output: Matrix $A$
for vertex=1...n do
    $a_{vertex,vertex}$ =number of neighbors;
    for the direct neighbors of vertex do
        if the neighbor is not a pole then
            $a_{vertex,neighbor} = -1$;
        end
    end
end
Output: Constant vector $b$
Set all entries of $b$ to 0;
for the direct neighbors of south pole do
    $b_{neighbor} = \pi$;
end
```

Algorithm 1: The algorithm used to set up matrix A and the vector b of the system of linear equations for Latitude

```
Input: Matrix $A$, vector $b$ and Model $M$
Output: Modified Matrix A
for both poles do
    for the direct neighbors of pole do
        $a_{neighbor,neighbor} = a_{neighbor,neighbor} - 1$;
    end
end
$a_{0,0} = a_{0,0} + 2$;
Output: Constant vector b
for row=1...n do
    $b_{row} = 0$;
end
$previous = northpole$;
$here = 1$;
$maximum = 0$;
while $here! = southpole$ do
    for the direct neighbors of here do
        if $\theta_{neighbor} > maximun$ then
            $maximum = \theta_{neighbor}$;
            $nextpos$ = position of neighbor;
        end
        if $neighbor = previous$ then
            $prevpos$ = position of neighbor;
        end
    end
    for the direct neighbors clockwise between prevpos and nextpos do
        $b_{neighbor} = b_{neighbor} + 2\pi$;
        $b_{here} = b_{here} - 2\pi$;
    end
    $previous = here$;
    $here$ = neighbor of here indicated by nextpos;
end
```

Algorithm 2: Lognitude system of linear equations is structurally identical to that of Latitude

Laplacian smoothing is a simple and efficient method for mesh smoothing. Every node is progressively shifted towards the centroid of its adjacent nodes. This is a local operation since at each step the movement of a vertex is determined only by its neighboring vertices.

A mesh can be thought of as a spring system by considering each edge connecting two nodes as a linear spring. Laplacian smoothing is then considered for minimizing the spring forces that are active on each node. Since a balanced spring system over the sphere can not contain folded elements (see Figure 2.5), it turns out that if Laplacian smoothing is applied to every vertex and the vertex is projected on the sphere all folded

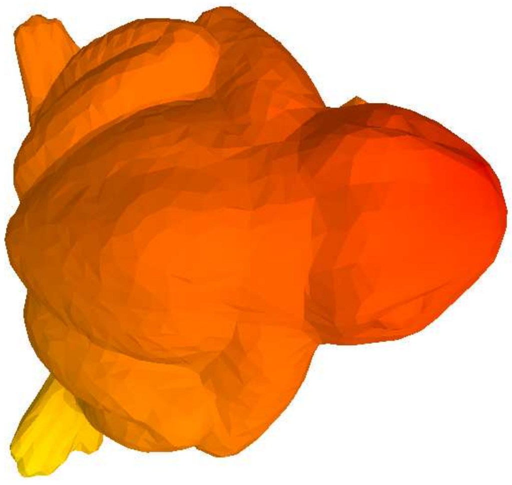


Figure 2.1: The result of thermal conduction at latitude θ applied on the frog. The bottom leg has temperature 0 (north pole) while the upper part of the head has temperature π (south pole).

elements tend to unfold after a sufficient number of iterations (see Figure 2.6). Since Laplacian smoothing does not perform any triangle area balancing certain elements may become degenerate. Besides, the relaxation process may collapse if one or more elements overgrow [2]. We used a simple variation where we determine the position based on the weighted sum of the centroids of the direct neighboring triangles of each vertex. We use as weights the areas of each such triangle. This simple approach yields a smoother mesh with more balanced element area since larger polygons tend to attract more vertices while smaller polygons tend to repulse them. Figure 2.2 shows the results of the initial mapping of the frog from [1] when applying thermal conduction method and Laplacian smoothing. It is generally desirable to use a mapping with uniform sized polygon areas, so in this context other variations of Laplacian smoothing or mesh smoothing can also be considered [29, 33].

The above procedure is expressed concisely by the following two steps: We first project each vertex on the unit sphere:

$$V = \frac{V_0}{||V_0||}, \forall V$$

where V is a vertex of the mesh and V_0 its original position on the mesh. Then, while folded elements still exist,

$$V = \frac{\sum_i^m area_i Centroid_i}{||\sum_i^m area_i Centroid_i||}$$

where $area_i$ is the area of the corresponding i-th adjacent triangle of vertex V, $Centroid_i$ is the centroid of the same triangle and vertex V is connected to m triangles.

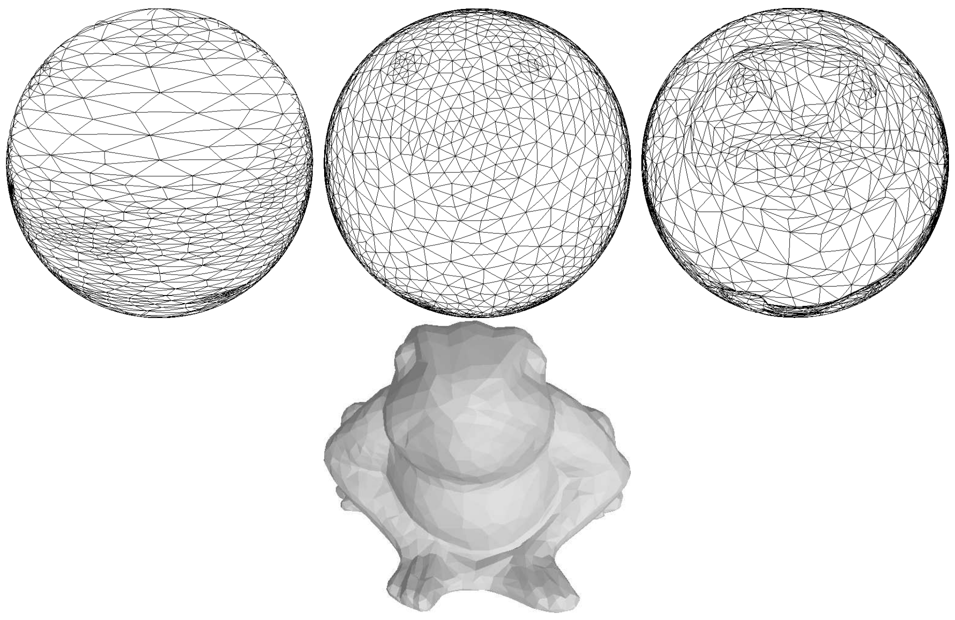


Figure 2.2: (top left) The result of initial mapping with the thermal conduction method, (top center) the result of the initial mapping with the Laplacian smoothing technique, (top right) the result of the mapping after optimization and (bottom) the original frog model.

2.2.2 Optimizing Mapping for Morphing

For the purpose of spherical parameterization [7] uses the following constraints. For each vertex $V(v_x, v_y, v_z)$:

$$v_x^2 + v_y^2 + v_z^2 = 1, \forall V(v_x, v_y, v_z) \tag{2.1}$$

To avoid unequal faces, for each face f the area is constrained to be exactly

$$area(f) = \frac{4\pi}{n}, \forall f \tag{2.2}$$

where n is the number of faces. Finally the angles of each face are constrained to be in $[0, \pi]$ which is compiled to six inequalities per triangular face. The equations to calculate the spherical area and angles are described in section 1.2.4. The objective function that favors short lengths on face edges is:

$$\sum_{\forall f} \sum_{i=1}^{3} cos(s_i^f) \tag{2.3}$$

where s_i^f is the angle (length of the arc) formed by the corresponding edge and the center of the unit sphere. We implemented this method for morphing but found that the results were not appropriate for our purposes. The method is very slow in converging and may place some faces very far away from their original position or place neighboring faces in distant spots on the sphere.

In general, it is desirable the spherical parameterization to preserve some important characteristics of the original model. Since almost always a mapping to a simpler do-

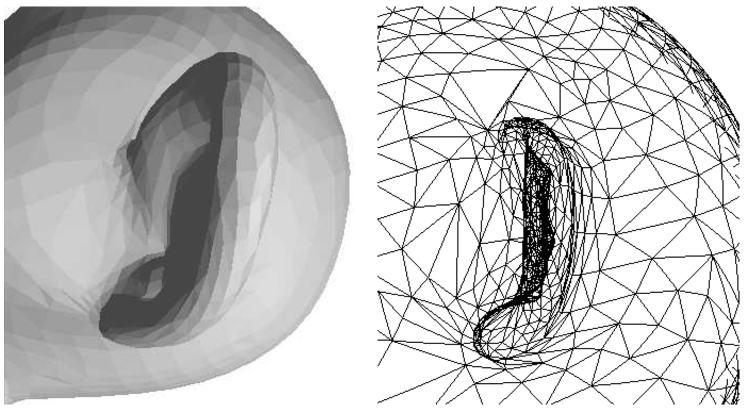

(a) Model (b) Projection

Figure 2.3: Close-up of the mapping from the monkey model. The original projection contains folded and overlapping triangles.

main like a sphere will introduce distortion, it is important to minimize the distortion in significant characteristics of the model.

For this reason we use the following set of constraints and objective functions that are more appropriate for morphing.

Geometric Constraints: For each vertex $V(v_x, v_y, v_z)$:

$$v_x^2 + v_y^2 + v_z^2 = 1, \forall V(v_x, v_y, v_z) \tag{2.4}$$

Topological Constraints: For each face with vertices V_0, V_1, V_2 and for each vertex of this face, each vertex should stay on the same side of the plane defined by the other two vertices and the center of the sphere:

$$\begin{aligned} (V_1 \times V_2) \cdot V_0 > 0 \\ (V_2 \times V_0) \cdot V_1 > 0 \\ (V_0 \times V_1) \cdot V_2 > 0 \end{aligned} \tag{2.5}$$

Objective Function: We use as the objective function to be optimized the sum of all inner products of every mapped vertex $V = map(P)$ with their corresponding original position P

$$\sum_{\forall P} P \cdot map(P) \tag{2.6}$$

The motivation behind this choice is that a good mapping should preserve *locality*, meaning that the projected vertices must not lie far from their original projected (normalized) positions and close vertices in the original model should have close projected positions. This is an important factor since it implies that in a convex model the optimum positions of vertices will be exactly at the projected positions. In non-convex models it leads in introducing distortion in concave areas but keeping important vertices

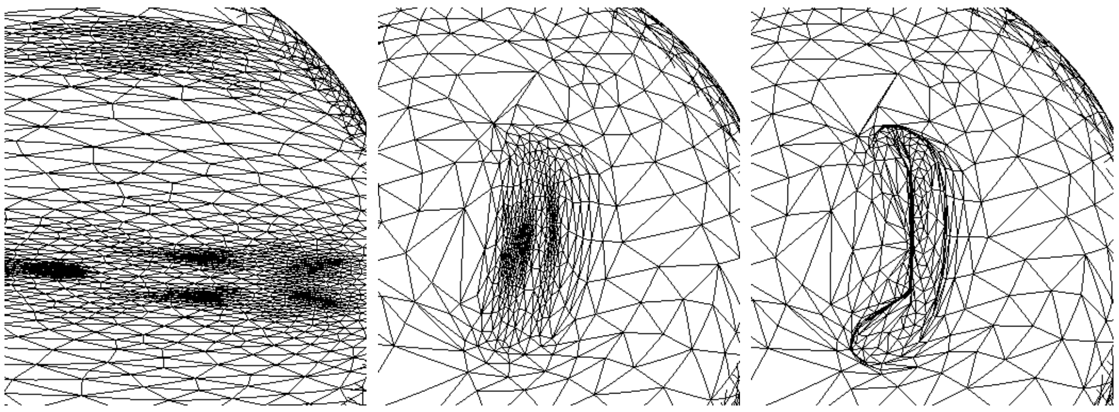

(a) Thermal (b) Laplacian (c) Optimization

Figure 2.4: Close-up of the mapping from the monkey model. In the thermal mapping, although the mapping is valid it deviates from the original projection. The Laplacian mapping contains good quality triangles resulting from the smoothing process but loses some locality. Finally, the optimization balances between good quality triangles and preserving the mapping close to the original projected mapping. Note how the mapping is naturally distorted in concave areas around the ear.

(for example those over the convex hull) in their corresponding projected positions. The vertices of the convex hull will give a basic skeleton for the model to lead the rest of the mapping. Finally, to avoid introducing collapsed and thin elements in the mapping a suitable $\epsilon > 0$ value must be chosen for equation 2.5. Figures 2.3 and 2.4 illustrate a comparison of various projection methods.

Figure 2.2 illustrates the optimized mapping for the frog, while Figure 2.7 illustrates the final optimized mapping on the sphere for the monkey from [5].

2.3 Surface Correspondence and Interpolation

Following the successful mapping of the boundary of the two objects on the sphere, a merging process of the two topologies is performed. The purpose of this step is the creation of a final merged topology that is suitable for navigating back and forth to the original models.

This process requires each projected edge of one model to be intersected with each projected edge of the other. The search for intersections among edges can be computed exhaustively by checking all possible pairs of edges, but this brute force technique suffers from two problems: time complexity and numerical inaccuracy. More specifically, for two solids a and b the time complexity of this operation is $O(E_aE_b)$, where E_a and E_b are the number of edges of each solid. Additionally, small numerical inaccuracies in the

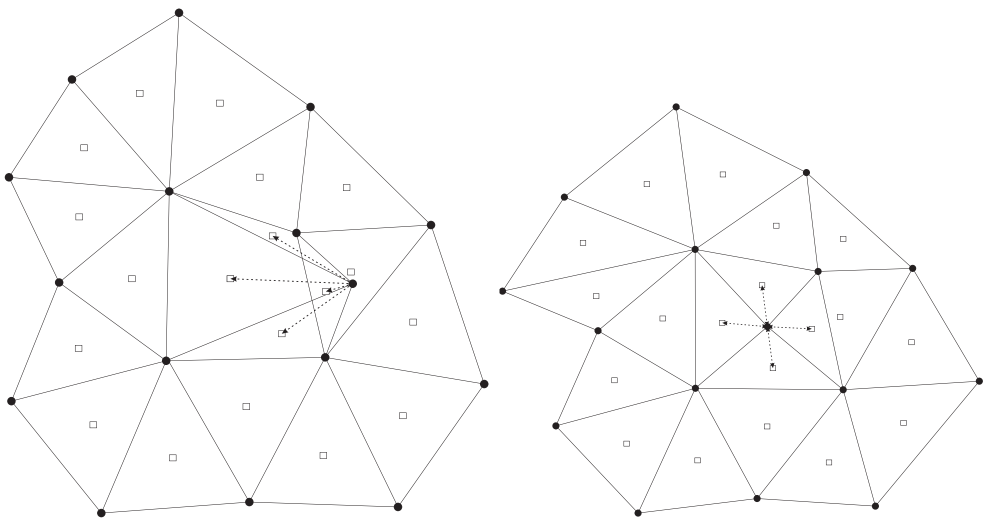


Figure 2.5: Geometric explanation of why Laplacian smoothing works. The original mesh (left) contains a node with folded elements, all the adjacent elements will act forces on that node forcing it to move in a balanced position (right). Consequently, all the folded elements will unfold after a certain number of iterations.

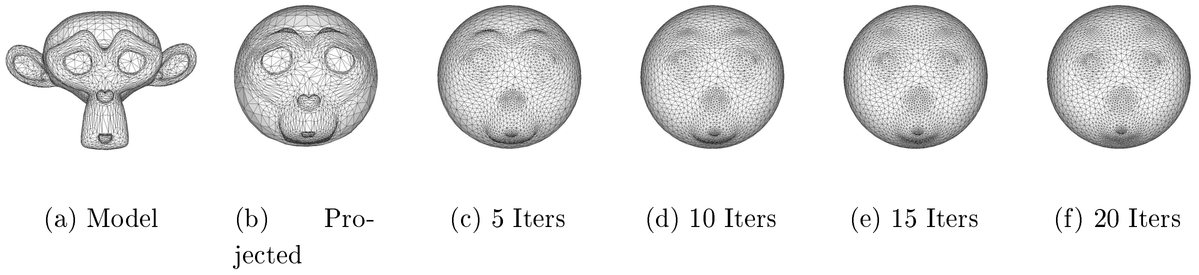


Figure 2.6: Laplacian smoothing run.

intersection computations may lead to an improper ordering of the intersections lying on an edge.

A method similar to the method of Kent et al. [14] was used to compute the intersections of the merged topology. The algorithm is based on the idea that starting from an intersection over an edge we can traverse all the remaining intersections by exploiting the topological information contained in the models. For each edge of the first model a list of edges of the second model is constructed (candidate list) which could actually intersect the processed edge. The candidate list is constructed using the mapping of the vertices of the first model over the second one and using neighboring information of the second model. More specifically, we need to find the triangle of the second model which contains one vertex of the starting edge and use it to construct a candidate list which contains all the edges of that triangle. This is required for only one vertex of the first mesh and can be computed directly in $O(n)$. Since from the candidate list containing edges of the second model we can traverse all the intersections in constant time by traversing to

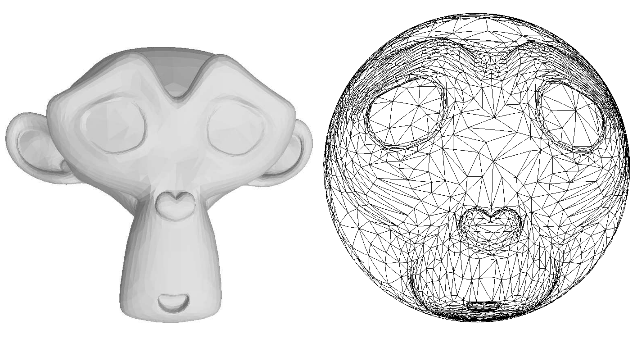


Figure 2.7: The final result of mapping (right) applied to the monkey with 5600 faces (left).

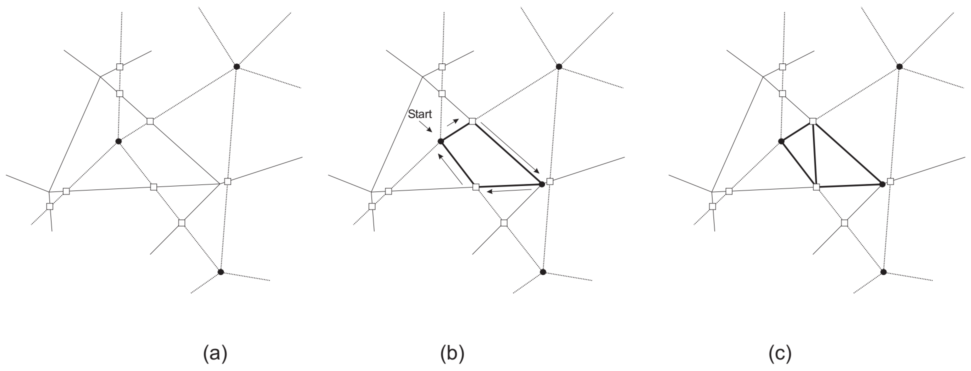


Figure 2.8: (a) Finding Intersections (b) Curve faces visited in clockwise manner (c) Triangulation

adjacent elements as long as intersections exist in the specific edge (see Figure 2.8 for an example) and intersections and edges from the first model are visited exactly one time, the complexity of this algorithm is $O(E_a + I)$ where I is the total number of intersections. Generally, it is expected that I is much smaller than $E_a * E_b$ therefore this algorithm is more efficient than the brute force approach. In addition, this method correctly sorts the intersections over the edge and avoids improper ordering that could be caused by small numerical inaccuracies in the calculation of the intersections.

From the intersections found along with the vertices of the two models a set of spherical regions bounded by circular arcs is determined. These regions are always convex, therefore it is straightforward to triangulate them and compute the final triangulated merged topology. First, for each edge the list of intersections that belong to that edge is sorted by the distance from each vertex of the edge. Additionally, for each vertex a list of the edges incident to the vertex in clock-wise order is calculated. Based on the aforementioned geometrical data we traverse each closed bounded region in a clock wise order and compute the triangulated merged topology. A new triangular face is created for every two continuous edges. If there is no edge connecting the endpoints of the two edges then a new edge is added to the merged topology. This operation continues until all edge fragments created from splitting the edges at the intersection points are visited exactly twice, once in clock-wise order and once in counter clock-wise order. Figure 2.8

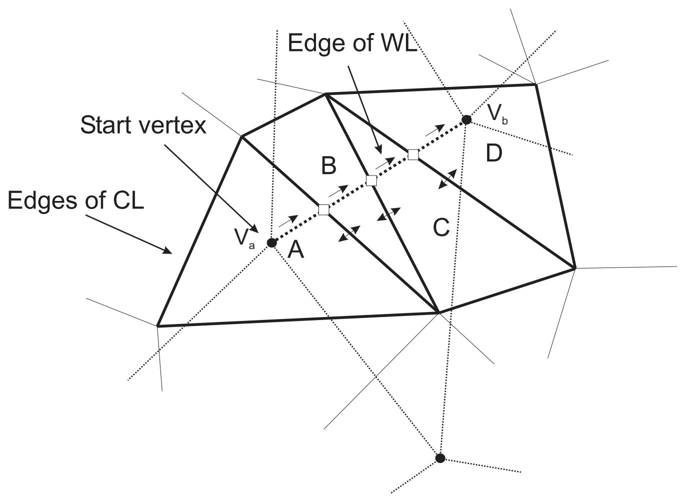


Figure 2.9: Calculating the Intersections of an Edge. Light edges are from M_a, Dark edges are from M_b Bold edges are those inserted in the Candidate List (CL).

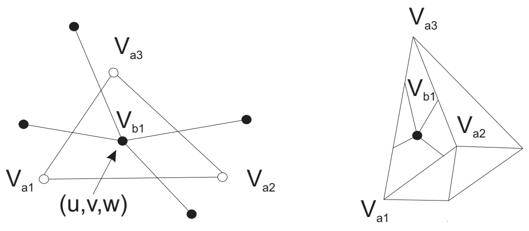


Figure 2.10: Mapping vertex V_{b_1} of the second model to a face of the first model $(V_{a_1}, V_{a_2}, V_{a_3})$

illustrates this process.

The final step of the algorithm involves the projection of the merged topology back to the original models. For each model A the vertices of the other model B along with the intersection points are projected on A. For intersections, the parametric value of the intersection over the projected edge is used to determine the position in the original edge. In the case of interior vertices a similar method is used. In particular the barycentric coordinates of the vertex in the projected triangles are used to calculate the position in the original model. This is illustrated in Figure 2.10.

The first step involves the search for intersections and is proportional to the number of intersections $O(E_a + I)$. The traversal of the bounded regions has an $O(I \log I)$ time complexity, since it depends on the number of intersections and the effort to sort them. Finally, the triangulation of the faces of the merged topology has $O(N)$ time complexity.

Following the successful establishment of a correspondence between the source and target vertices the vertex positions are interpolated to acquire the final morphing sequence. To this end, we use simple linear interpolation. The interpolated vertex positions are calculated as

$$V_m = (1-t)V_a + tV_b, t \in [0,1]$$

where V_a is a vertex of solid A and V_b is a vertex of solid B. The advantage of linear

```
Input: Two polyhedral representations for objects $M_a$ and $M_b$
$v1_a \leftarrow$ first vertex of $M_a$;
$MapToB[v1_a] \leftarrow$ face of $(M_b)_p$that contains $(v1_a)_p$;
Add the edges originating at $v1_a$ to Work List ($WL$);
Mark those edges Used;
while WL is not empty do
    $e_a \leftarrow$ next edge of $WL$;
    $v1_a, v2_a \leftarrow$ endpoints of $e_a$;
    $f_b \leftarrow MapToB[v1_a]$;
    Add the edges of $f_b$ to Candidate List ($CL$);
    while CL is not empty do
        $e_b \leftarrow$ next edge of CL;
        Intersect $e_a$ and $e_b$;
        if Successful then
            Add intersection point i, to Model;
            Create links from $e_a$ and $e_b$ to i;
            $f_b \leftarrow$ Face of $M_b$ on other side of $e_b$;
            Add the two other edges of $f_b$ to $CL$;
        end
    end
    $MapToB[v2_a] \leftarrow f_b$;
    Add the unused edges originating at $v2_a$ to $WL$;
    Mark those edges Used;
end
```

Algorithm 3: The algorithm used to compute the intersections of the merged topology in $O(E_a + I)$.

interpolation besides its simplicity is that it can be easily and very efficiently adapted to GPU techniques since it can be implemented as a simple morphing shader interpolating vertices and features (lighting, textures) in real-time. Nevertheless, linear interpolation may not always be desirable especially in very complex meshes where self-penetrations may appear during the morphing sequence of the models. More advanced interpolation techniques are applied in such cases. Some of them are also implemented in shaders but their performance may vary depending on the limits set by the GPU.

Chapter 3

Feature Based Morphing

3.1 Introduction

After the successful mapping of the two models as described in the previous section, we can directly overlay the spherical embeddings and retrieve a morphing sequence. While the result is a smooth transformation between the models sometimes it fails to keep basic characteristics of the models stable in the intermediate frames, see for example Figure 3.8. It is generally desirable basic characteristics and distinguished features of the models to be matched during the morphing sequence (ears,head,legs,etc). A way to assure that these features are preserved during the transformation is to make the corresponding vertices at (approximately) the same positions on the spherical embeddings. Although these features can be manually defined by the user, the procedure can become tedious and error-prone in complex models. In general, it is desirable these features to be automatically detected and matched.

3.2 Feature-based Morphing

3.2.1 Detecting Feature Regions

To detect feature regions in a point cloud we employ an approach [31, 30] developed earlier for reverse engineering based on discovering features on the point cloud by detecting local changes in the morphology of the point cloud. We use region growing, detection of rapid

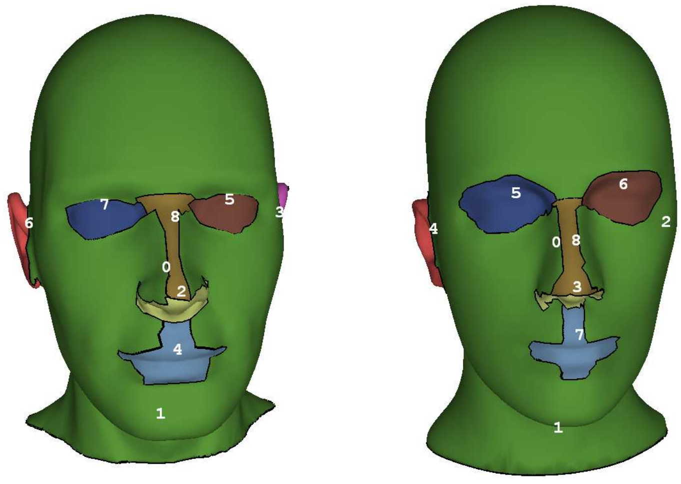


Figure 3.1: Detecting and matching feature regions in two head meshes (matched regions have similar color).

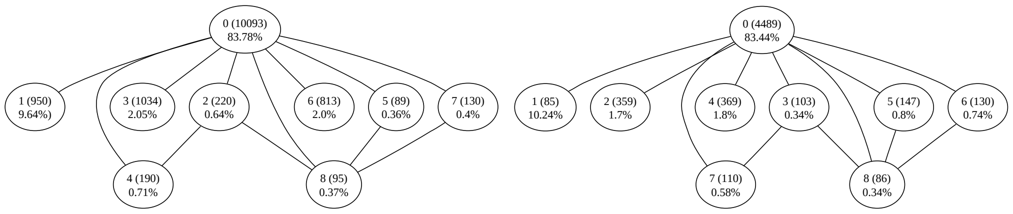


Figure 3.2: Original graphs the two head objects.

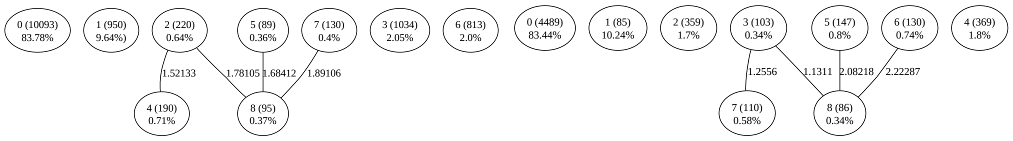


Figure 3.3: Reduced graphs of the two head models.

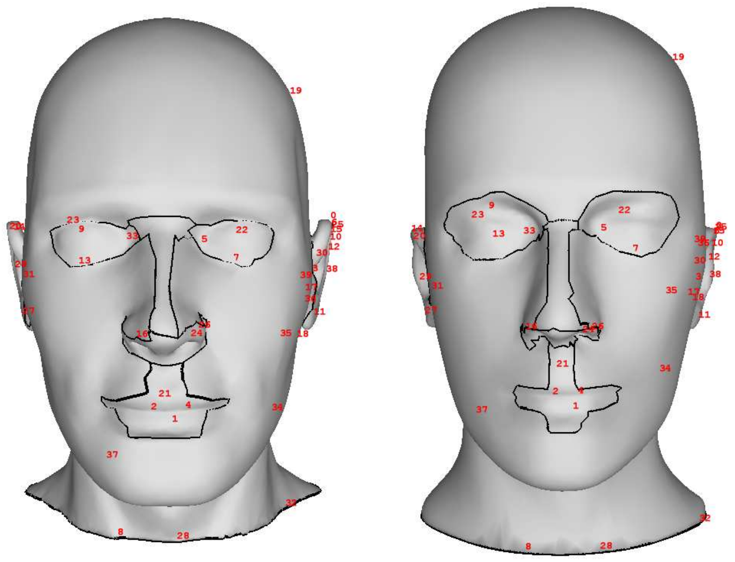


Figure 3.4: Detecting and matching feature points inside feature regions.

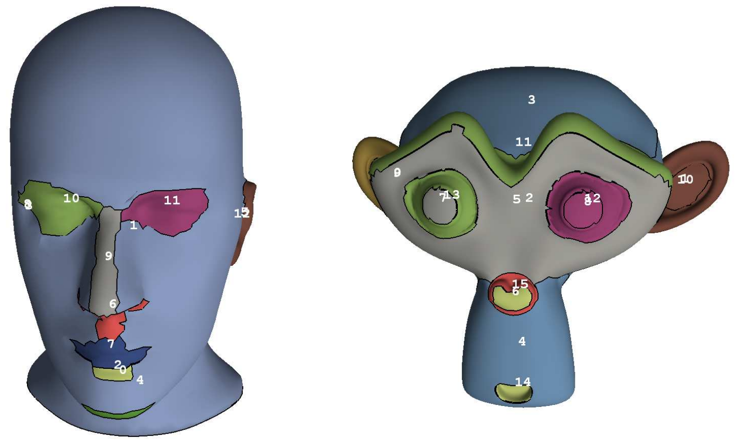


Figure 3.5: Detecting feature regions in a head and a monkey model.

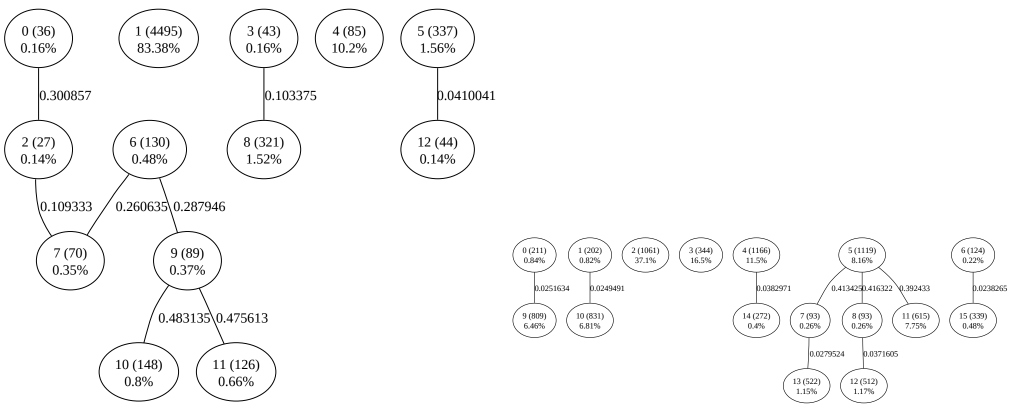


Figure 3.6: Reduced graphs of the head and monkey models.

variations of the surface normal and the concavity intensity, i.e. the distance from the convex hull. This results in a number of regions that represent object feature areas (for example see Figure 5.3). In the context of this work we employ this method to detect features in models for the purposes of matching and alignment of the two morphed solids.

More specifically, morphological features in the point cloud are detected using a characteristic called *concavity intensity* of a point which represents the smallest distance of a point from its convex hull.

Definition 1: *Concavity intensity* of a vertex V denoted by $I(V)$ is the distance of V from the convex hull of the solid.

This characteristic is used to detect concave features in the point cloud. Figure 3.7(a) presents the point cloud of a screwdriver and its convex hull facets, whereas Figure 3.7(b) displays a greyscale mapping of the concavity intensity value of each point (white color corresponds to the maximum distance whereas black corresponds to points located on the convex hull). Feature regions are also detected by rapid variations of the surface normal. These two characteristics are combined in a region growing method that results in sets of points corresponding to individual features (Figure 3.7(c)).

```
Input: Two polyhedral representations for objects $S_1$ and $S_2$
for each vertex $V$ of $S_1$ do
    calculate $I(V)$
end
for each vertex $U$ of $S_2$ do
    calculate $I(U)$
end
for $S_1$ and $S_2$ do
    compute the corresponding feature region sets $F_1$ and $F_2$
end
for $F_1$ and $F_2$ do
    compute the corresponding connectivity graphs and perform graph reduction on
    them
end
Perform a 3D alignment of $F_1$ and $F_2$ up to scaling,rotation and translation;
for each feature point in $F_2$ do
    find the closest feature point in $F_1$
end
Perform the sphere mapping on $F_1$;
Perform the sphere mapping of $F_2$ under the additional constraint that each
mapped point has to be close to the corresponding point of the first object;
```

Algorithm 4: The algorithm for feature based morphing.

3.2.2 Region Matching

The first step for Feature based morphing is to establish a correspondence between the region patches of the two models. Since the region patches detected can differ in number and in morphology we need more high level information to find a good matching. In this context, we create the connectivity graph that captures adjacency information as illustrated in Figure 3.2. For each feature region node we also compute the percentage of the corresponding area that it covers and the number of points it contains. For every edge we calculate the geodesic distances between the centroids of the corresponding feature regions.

The graphs are then simplified by reducing edges that correspond to large geodesic distances (see Figure 3.3). In addition, small regions that can introduce noise and are not significant are merged. In general, it is desirable to have a small number of regions each one covering a significant area of the original model.

$$Distance(i,j) = ||Centroid_i - Centroid_j|| * \frac{\max\limits_{i,j} Area}{\min\limits_{i,j} Area} \tag{3.1}$$

The reduced graphs resulting from the elimination of graph edges with large geodesic distances can be used to establish a correspondence between the patches. In the case like in Figure 3.3 the match is trivial, in more complex cases like in 3.6 we match nodes with equivalent or almost equivalent degree and edges, for example node 9 in the first model and node 5 in the second. We also take into consideration the area covered by each region patch in the case there are more than one candidates in the first model for matching a single one in the second model. In that case a similarity metric is used 3.1, favoring the matching of regions covering equal areas. Moreover, many patches of the second model can be matched with a single one in the first model. This is especially true in the case where a patch of the first mesh covers a large area, for example in Figure 3.5 the patch 1 (back of the head covering 83%) is matched with patches 2,3 and 4 of the second model (covering a total of 65%).

Since some region patches in the second model can be left unmatched when there are no suitable candidates in the first model, the number of unpaired regions can be used as a metric of quality for the final matching.

3.2.3 Feature Point Matching

For each feature area we detect points with certain characteristics that provide a high level description of specific structural characteristics of the solids. For each object we have:

Definition 2: A vertex V is called a *feature point* if and only if $I(V)$ exhibits a local extremum at V.

We then process the feature point set to eliminate feature points that results from local noise and feature points that are very close to each other. The resulting point set,

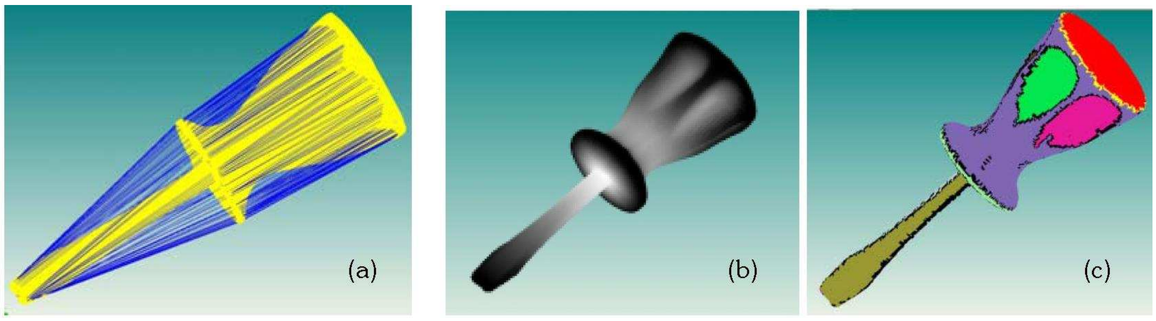


Figure 3.7: Feature region detection.

called a *feature point set*, provides a high-level description of the maximum convexities and concavities of the object. Note that normally convex hull vertices (distance 0 from the convex hull) are part of the feature point set. The algorithm for feature-based morphing is illustrated in Algorithm 4.

Following the establishment of a correspondence between the region patches of the two models the feature points of the corresponding patches are paired according to the distance between them. Since the patches may be in different locations in each model the two regions must first be translated so that their corresponding centroids coincide. An example of a match of the feature points is illustrated in Figure 3.4.

3.2.4 Feature-based Optimization

For the feature based mapping of the second model we use the following set of constraints and objective functions to obtain a more appropriate mapping based on the feature point correspondence of the models :
Geometric Constraints: For each vertex $V(v_x, v_y, v_z)$:

$$v_x^2 + v_y^2 + v_z^2 = 1, \forall V(v_x, v_y, v_z) \tag{3.2}$$

Topological Constraints: For each edge (circular arc over the sphere) the length must stay the same during the optimization:

$$V_{i1} \cdot V_{i2} = OriginalLength, (V_{i1}, V_{i2} \in E_i), \forall E \tag{3.3}$$

Objective Function: We use as the objective function to be optimized the sum of all inner products of every mapped feature vertex $f(V_a) = V_b$ of the second model with their corresponding mapped feature vertex of the first model V_a

$$\sum_{\forall V_a} V_a \cdot f(V_a) \tag{3.4}$$

For equation 3.3 an ϵ value must be selected, this value controls the distortion that it will be introduced in the optimized mapping in order to match the feature points. Since equation 3.3 compares circular arches over the unit sphere the ϵ value represents the degree of freedom (in degrees) that the mapping can be distorted. A very large ϵ will lead in a perfect feature point matching at the cost of a very distorted mapping, while a small ϵ will lead in no distortion but in no good feature point matching. Usually, an ϵ that allows a change up to 0.5 degree at each edge is sufficient unless the feature points are too far away.

3.3 Comparison of Morphing Results

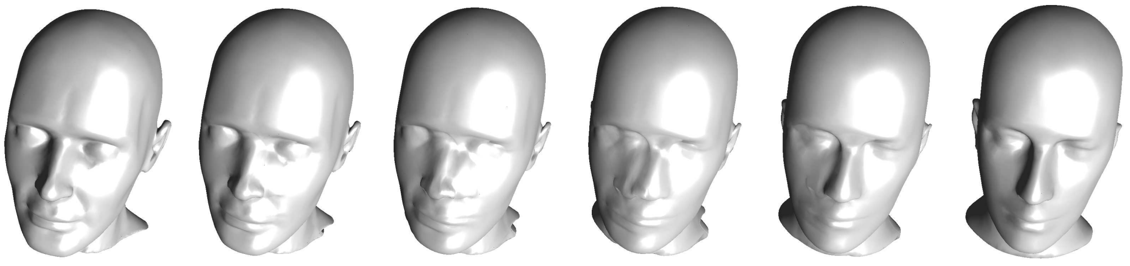


Figure 3.8: Morphing with no alignment.

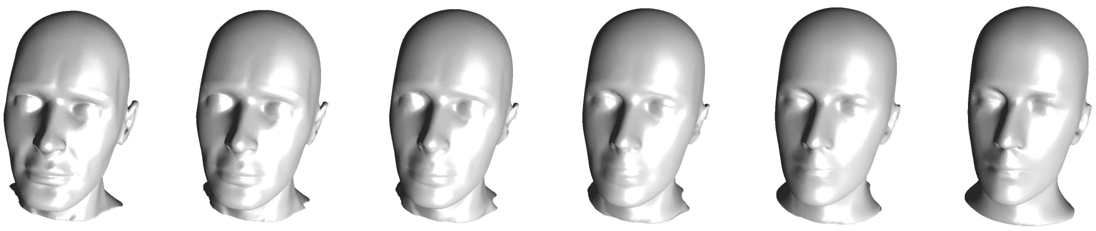


Figure 3.9: Morphing with alignment but no feature point matching.

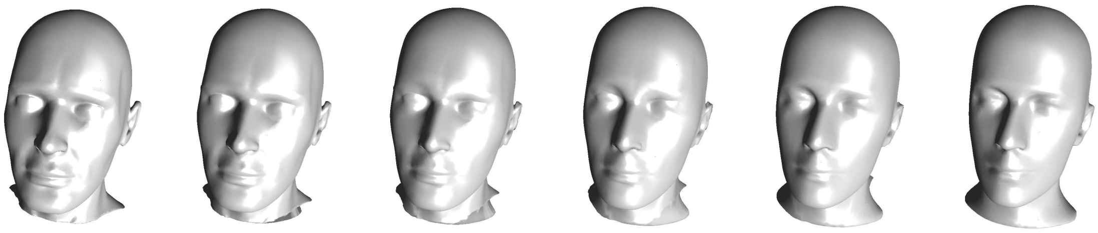
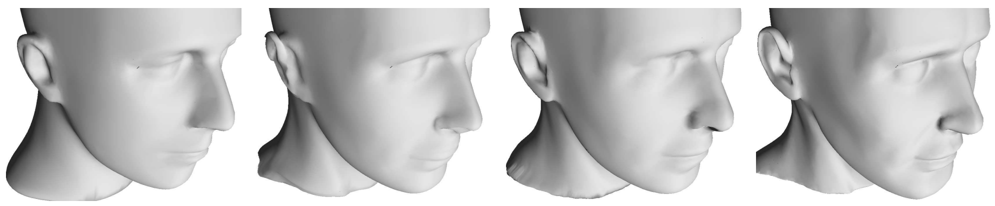


Figure 3.10: Morphing with alignment and feature point matching. The improvement is apparent in the details of the characteristics of the intermediate frames.



(a) M_1

(b) 50% Morph with alignment but no feature point matching

(c) 50% Morph with alignment and feature point matching

(d) M_2

Figure 3.11: Close-up of the morphing sequence.The improvement around the ear area is noticeable.

CHAPTER 4

IMPLEMENTATION

4.1 Overview

In this chapter we briefly describe the various tools and technologies used to implement the algorithm described in the previous chapters.

4.2 Programming language and tools

C++ was used as the development language of choice since the main requirements for our application was speed of execution, availability of scientific libraries and rapid-prototyping. C++ has a wide range of well established and tested scientific libraries like boost [6] and LAPACK [16].

Since C++ is not the best language when considering testing algorithms and quick changes, various scripts in other more high level languages like python were implemented and used during the development. These scripts were used mainly through applications that supported scripting like Blender [5]. Although python is an excellent language for rapid prototyping, it suffers from slow speed of execution sometimes in the factor of one tenth of the equivalent C++ code. Consequently all the scripts were finally ported to C++ for the sake of speed.

4.3 Shaders and special effects

4.3.1 GPU Shaders

Modern graphic hardware provides greater control over the rendering process through a higher and flexible hardware abstraction model. The various stages of the rendering process can be replaced by programs called shaders written by the user in a corresponding shading language. Although shaders are designed to be executed directly on the GPU on the proper point in the pipeline, they can be also used successfully in general processing.

Some examples of shading languages are Cg, HLSL and GLSL. These languages usually are based on the C programming language with the addition of graphic specific extensions. GLSL was used for all the shaders of the application. GLSL is a standardized high level shading language meant to be used in conjunction with OpenGL.

4.3.2 Normal Mapping

Normal mapping is computer graphics technique where the normal (the way the surface is facing) of a model is replaced according to a multichannel image(r,g and b channels) derived from a set of more detailed versions of the objects. The values of each channel usually represent the xyz coordinates of the normal in the point corresponding to that texel. The result is a richer, more detailed surface representation that more closely resembles the details inherent in natural world, see figure 4.1. This technique is used to enhance the appearance of low poly models. We have used normal mapping to enhance the final resulting morphing sequence. The normal maps of the two models are also blended through the morphing sequences and interpolated surface normals are calculated on the fly in GPU shaders, see figures 5.17,5.16.

4.4 Optimization software

IPOPT software was used as a non-linear solver for the optimization problem. IPOPT is an open source package available from COIN-OR [8], [34] under the CPL (Common Public Licence) for large-scale nonlinear optimization. This means, it is available free of charge also for commercial purposes. IPOPT implements an interior point line search filter method that aims to find local solution of the optimization problem defined at 1.2.2. IPOPT has been designed to be flexible for a wide variety of applications and provides various code interfaces, namely the AMPL modeling language interface, and the C++, C and Fortran interfaces.

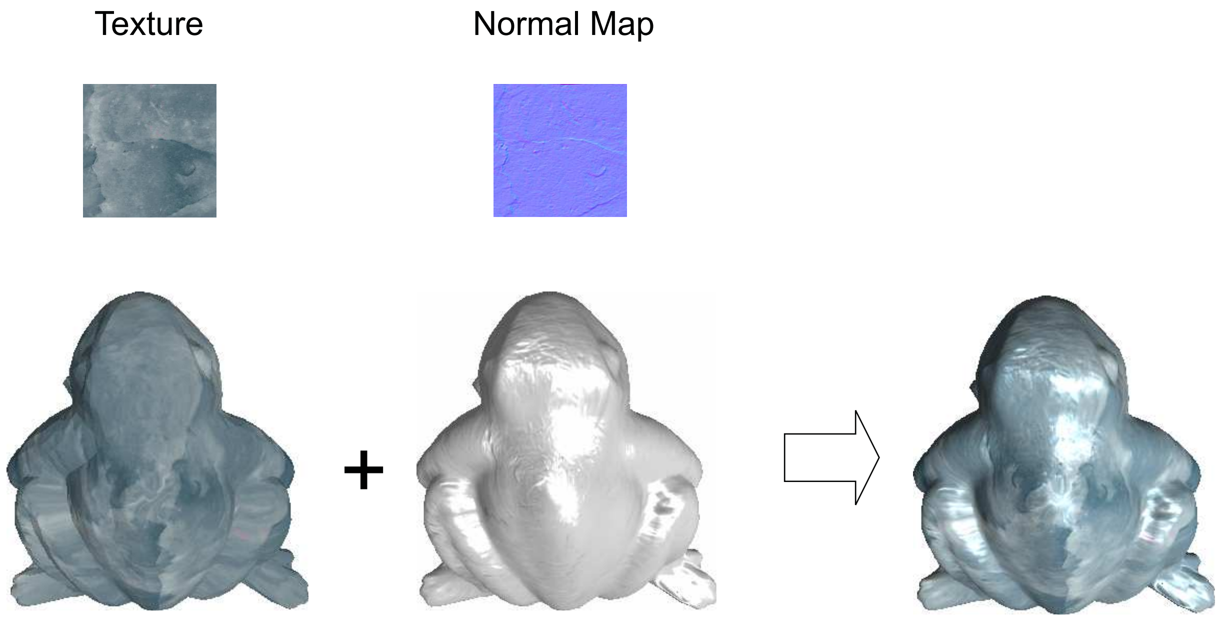


Figure 4.1: Normal mapping.

4.5 Additional software

During the development of the software for the implementation of the method described in the previous sections, we used some additional software. The software used was both in form of libraries for C++ and stand-alone software.

Mathematica software [35] allowed us to calculate analytical derivatives for very complex functions in form suitable for C++ applications. In addition a great part of the boost library was used extensively, some notable libraries used from boost are BGL for implementing graph algorithms and uBLAS Linear algebra library that can be used to easily manipulate matrices and contains many linear algebra algorithms.

We have also performed the experiments for out method on a variety of well-known models such as the blender monkey and models originating from 3D scanning. Those models usually required some pre-process in order to be suitable for morphing. The pre-process is usually cleaning of duplicated faces, filling of open regions (holes) and fixing self-penetrating elements, the aforementioned operations are necessary in order to be able to gain a valid mapping of the model over the parameterization domain. For the meshing and pre-processing of the all those models blender software [5] was used.

4.6 Application

The application that we created to implement the morphing algorithm was made with portability in mind. In Figures 4.3,4.4 and 4.5 the application is running under Windows XP and Windows Vista. All the libraries and tools used are portable across platforms and consequently the application can run in both Windows and Linux.

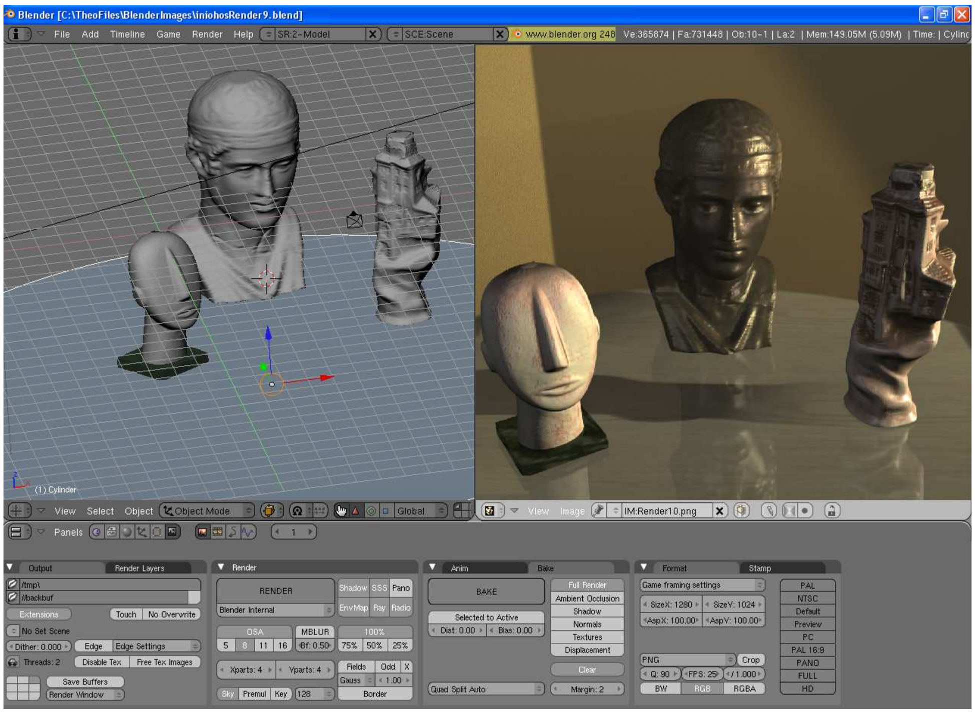
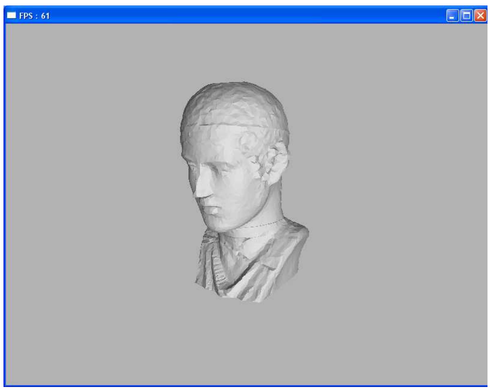
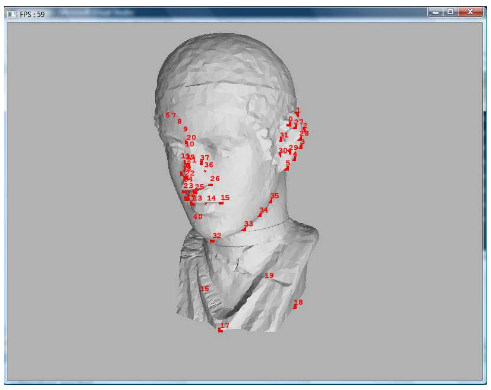


Figure 4.2: Models from 3D-scanner inside Blender Software [5].

The application can produce the morphing sequence of two models each time. The two models are given as command line parameters along with the desired number of frames in the final sequence. The application supports some popular file formats for the input models, namely Wavefront file format (OBJ), 3D Studio Graphics format (3DS) from 3D Studio Max and VRML file format (WRL). The application apart from the morphing sequence, can also visualize several stages of the algorithm like the parameterization of the models over the sphere the convex hull distances and the region clustering of the de-featuring process described in 3.2.1. In addition, an interface is offered to modify the automatically calculated pairs of *feature points* in order to improve the final result, in the case the user is not satisfied with the automatic matching.

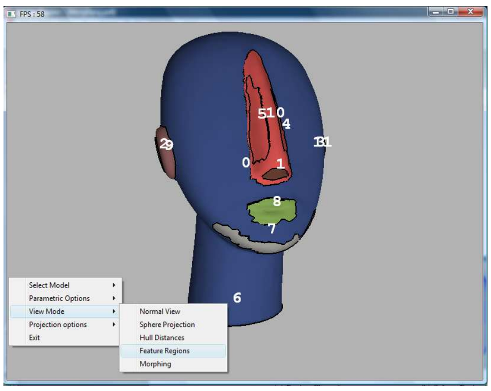


(a) Normal flat shaded view of the original model

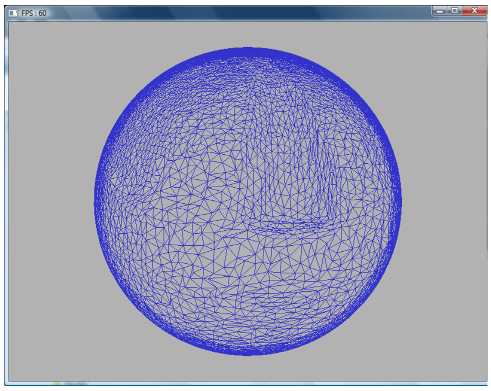


(b) *feature point* matching

Figure 4.3: Application.

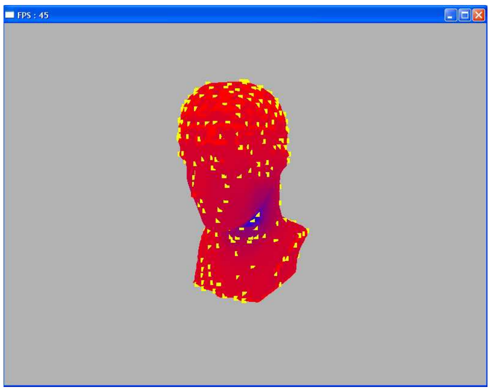


(a) Region Clustering

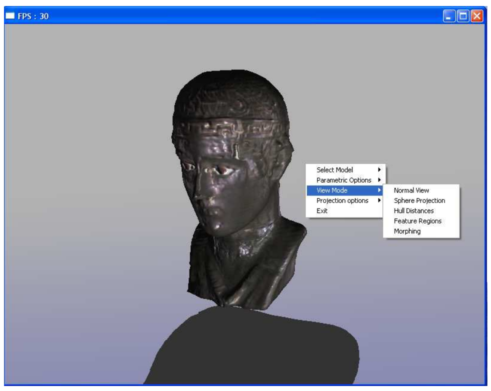


(b) Sphere Projection

Figure 4.4: Application.



(a) Convex hull distances and *feature point* detection



(b) Morphing View with normal mapping and shadows

Figure 4.5: Application.

Chapter 5

Experiments and Performance Evaluation

5.1 Introduction

We have developed software for implementing mapping, merging and interpolation as described in the previous sections. The platform used for development was a Windows XP Professional based system running on a Intel Pentium Q6600 Core 2 at 2.4GHz, 2GByte of RAM, with NVIDIA GeForce 8600GT. We have developed the system on Visual Studio 2005, using OpenGL 2.0 (Shader Model 3.0) and GLUT.

We have also implemented tools for visualizing all steps of morphing: map to the sphere, merging, and interpolation.

5.2 Experiments with no Feature Point Matching

We have tested our algorithm described in previous sections to generate morph sequence without any feature matching to various standard models like the blender monkey (*Suzanne*) and the Stanford bunny. Figures 5.1, 5.2 illustrates the resulting morphing sequence. Although the final result is quite satisfactory and the algorithm can be used to produce morphing sequences in arbitrary models, there is no high level feature correspondence that would make the final result more visual pleasing.

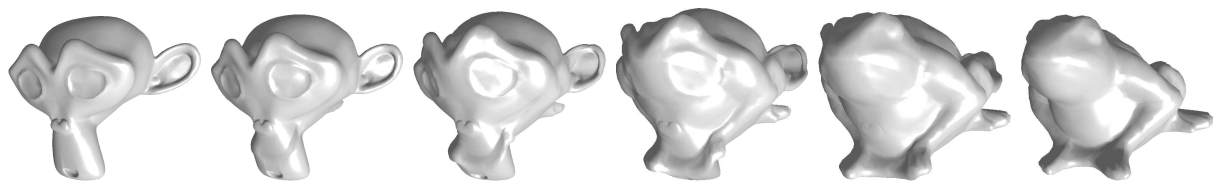


Figure 5.1: Morphing of Monkey (5600 faces) to frog (3924 faces), merged topology has 43794 faces.

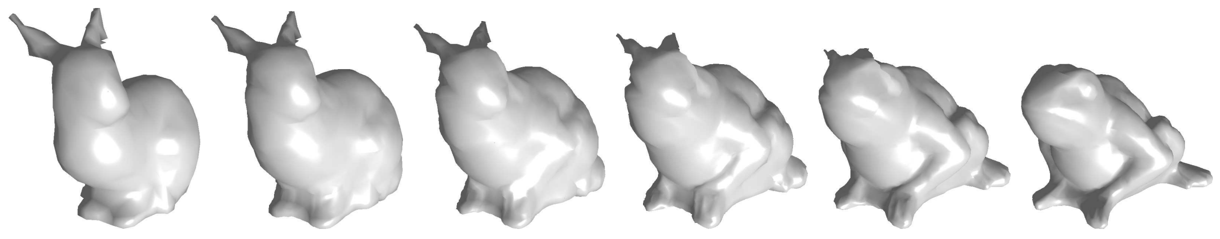


Figure 5.2: Morphing of Stanford bunny (876 faces) to frog (3924 faces) with no feature matching. The merged topology has 18142 faces.

5.3 Comparison with Feature Point Matching

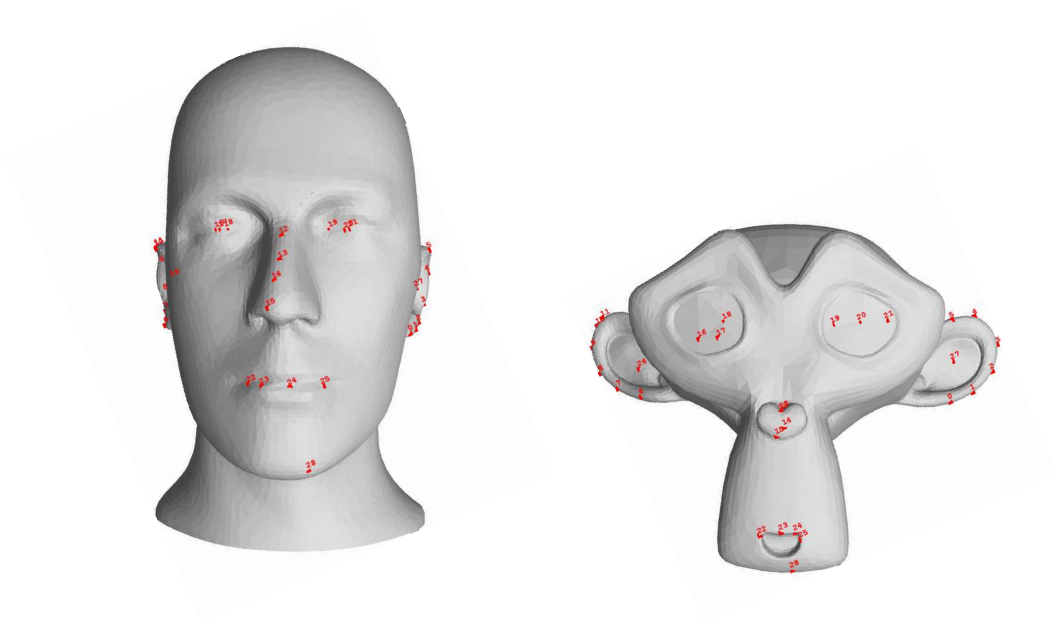


Figure 5.3: Feature point matching of Man. Head model and Monkey model.

Figures 5.3, 5.4, 5.5, 5.6, 5.7 and 5.8 illustrates the results of our method in a series of head models. From the final morphing sequences it is obvious that the results with feature point matching has several favorable characteristics. More specifically, geometrically important features of the models are matched and morphed correctly, especially the ears, mouth and the nose of the head models in the second case.

Figures 5.9, 5.10 and 5.11 illustrates the morphing sequence for two models of very different topology like the Fish Model and the duck Model from [1]. Again it should be

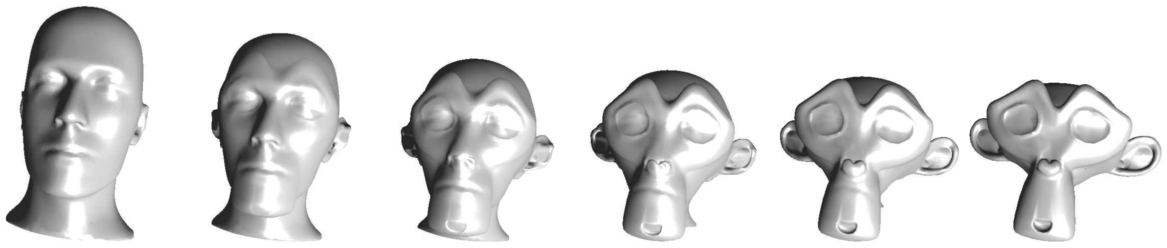


Figure 5.4: Morphing with alignment but no feature point matching.

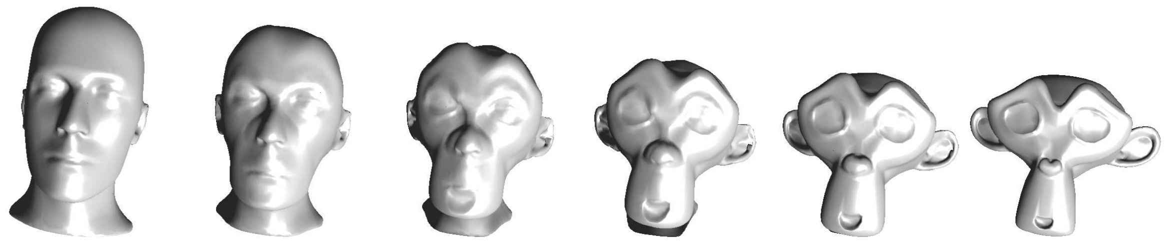


Figure 5.5: Morphing with alignment and feature point matching.

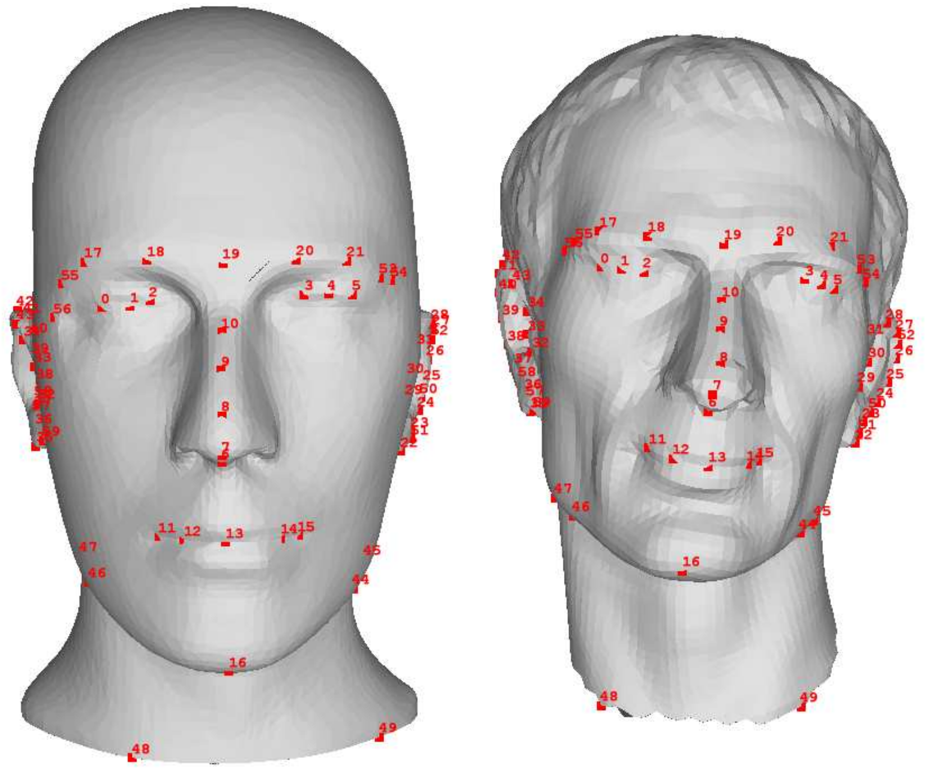


Figure 5.6: Feature point matching of Man. Head model and Ceasar model

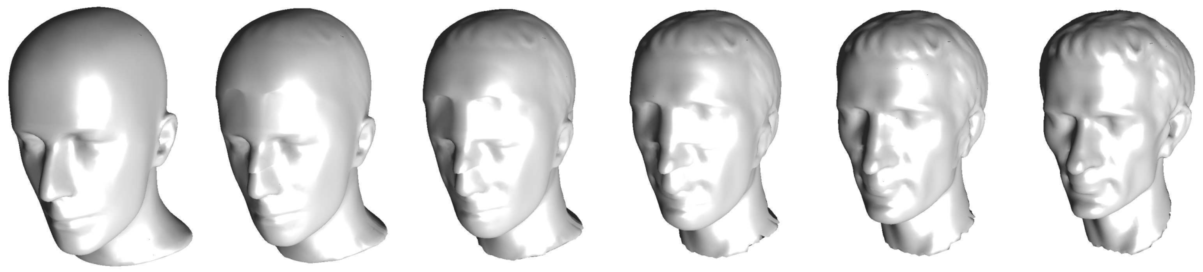


Figure 5.7: Another example of Morphing without alignment of Man. Head (11042 faces) to Ceasar model (13530 faces), merged topology has 109849 faces.

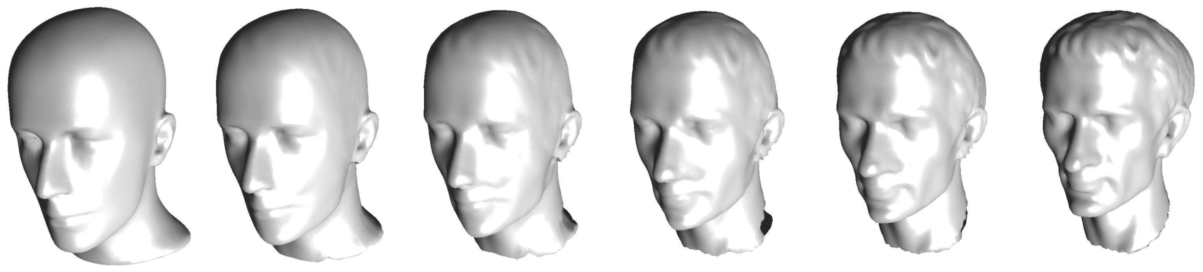


Figure 5.8: Morphing with alignment, merged topology has 127161 faces.

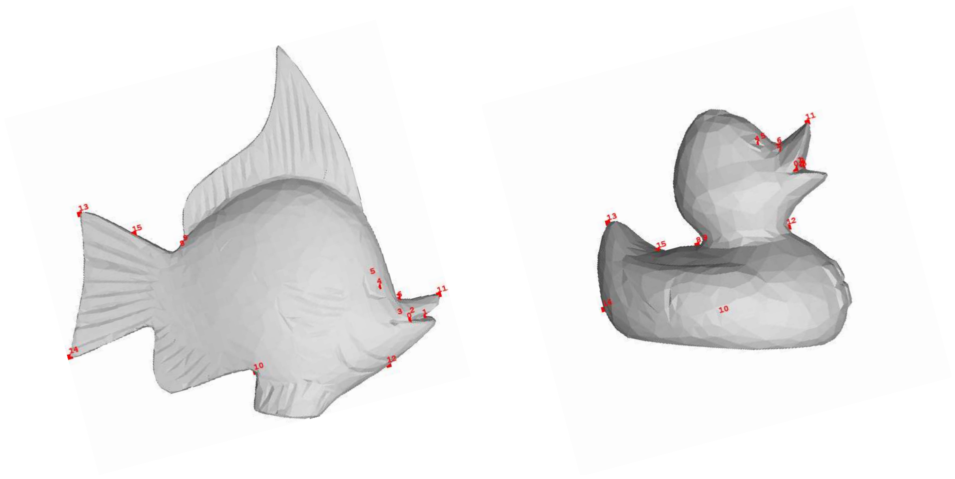


Figure 5.9: Feature Mapping

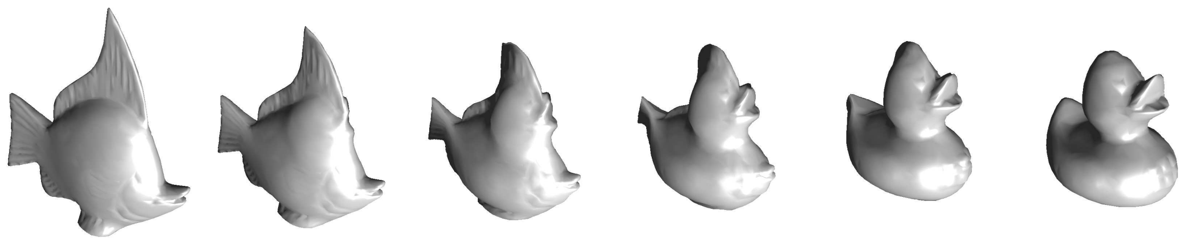


Figure 5.10: Morphing with alignment but no feature matching of Fish (4994 faces) to Duck (1926 faces), merged topology has 28526 faces.

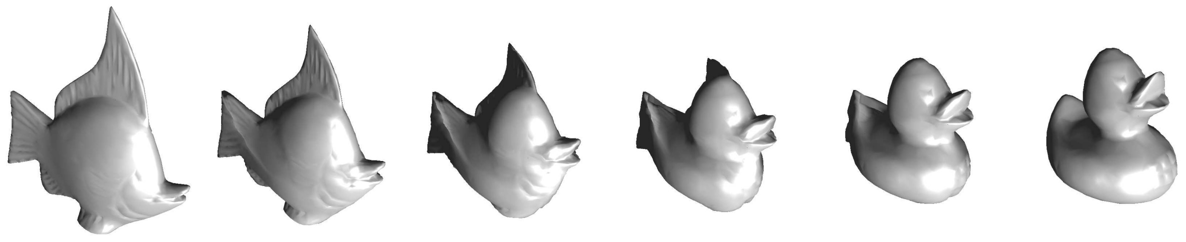


Figure 5.11: Morphing with alignment and feature matching of Fish (4994 faces) to Duck (1926 faces), merged topology has 33038 faces.

noticed, that morphing with feature point matching produce a considerably more visual pleasing sequence compared to the case with no matching. Therefore in the latter case, many features of the two models are not matched correctly, namely the head and the tail parts of the models. Nevertheless, the sequence produced is smooth and valid.

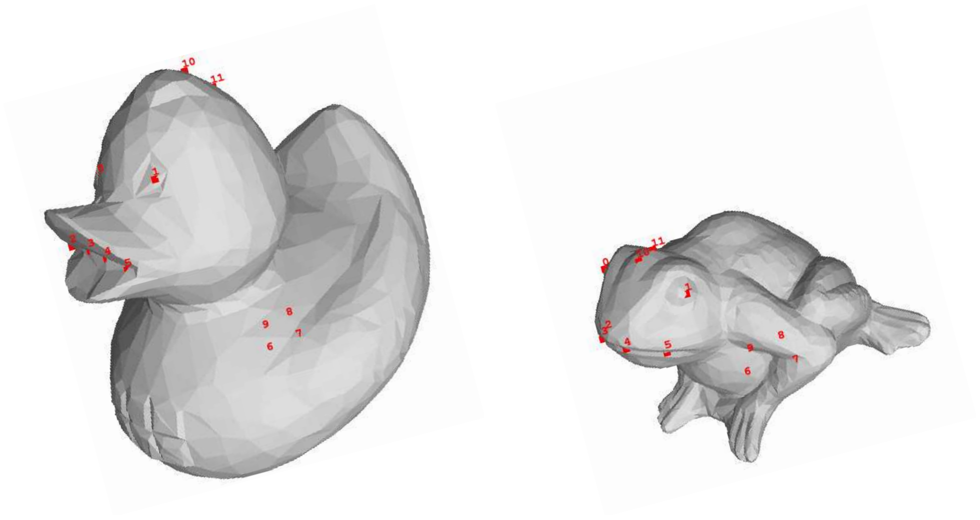


Figure 5.12: Feature Point Matching of Duck and Frog models

We also provide in figures 5.13, 5.14 and 5.15 a comparison of morphing sequences for the case of not aligned models. We observe that although the sequence with no alignment 5.13 is valid the result is improved considerably after the alignment and feature point matching in 5.15.

Finally, we have applied the algorithm described in previous sections to generate the morph sequence for two models obtained from 3D scanner 5.16, 5.17 and 5.18. Since the original models contained a lot or redundant vertices a level of detail reduction was applied on the models to reduce the computational cost in *Blender* application. In addition, the textures of the model along with their corresponding normal maps were blended to produce the final morphing sequence. The normal maps were generated in *Gimp* application.

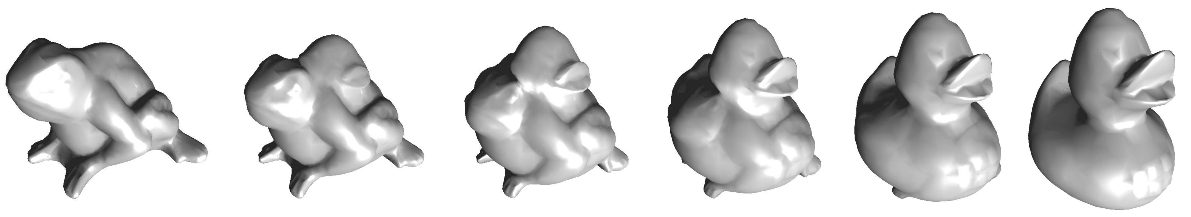


Figure 5.13: Metamorphosis with no alignment of Duck (1926 faces) to Frog (3924 faces), merged topology has 25800 faces.

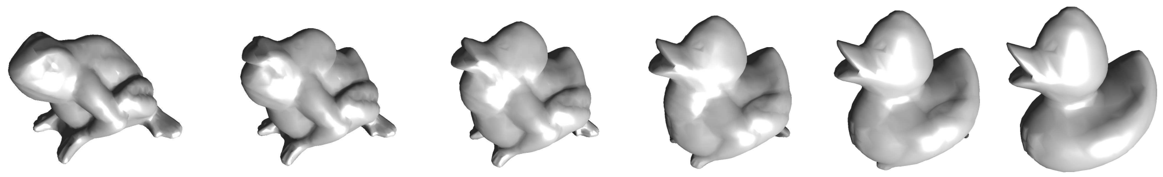


Figure 5.14: Metamorphosis with alignment but no feature point matching, merged topology has 27862 faces.

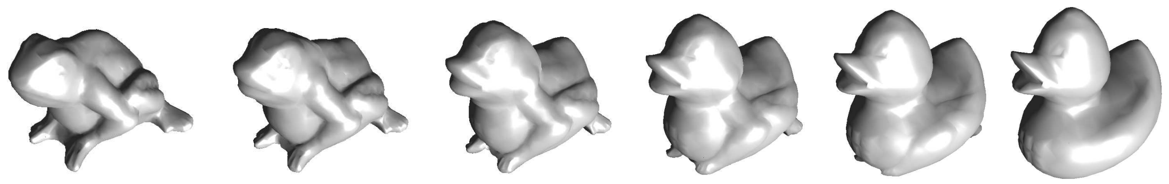


Figure 5.15: Metamorphosis with alignment and feature point matching, merged topology has 28332 faces.

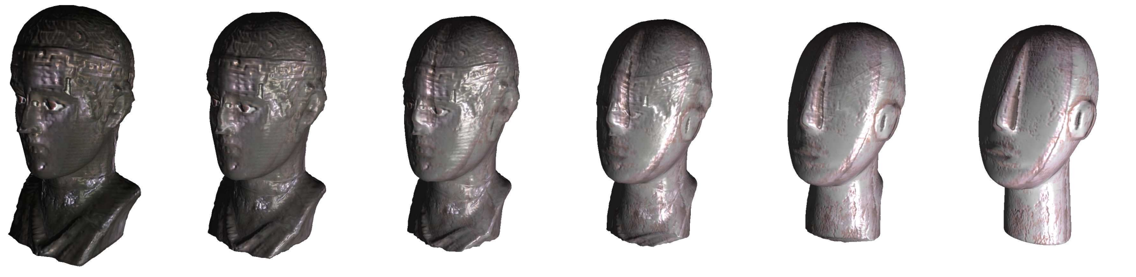


Figure 5.16: Morphing with alignment but no feature point matching of the Iniohos model (11098 faces) to Kykladitiko model (16798 faces), merged topology 142422 faces.

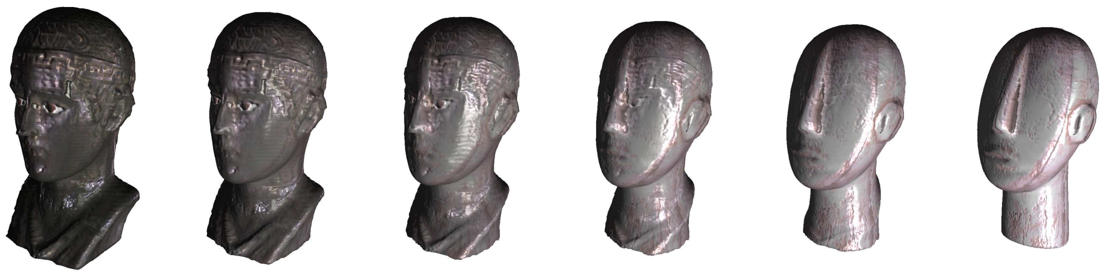
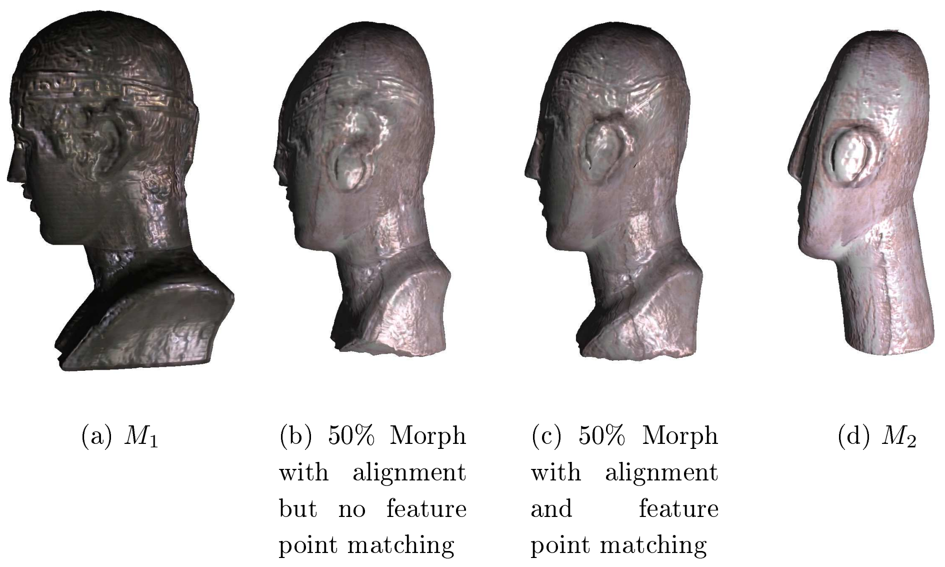


Figure 5.17: Morphing with alignment and feature point matching, merged topology has 142512 faces


(a) M_1

(b) 50% Morph with alignment but no feature point matching

(c) 50% Morph with alignment and feature point matching

(d) M_2

Figure 5.18: Comparison of the morphing results. The improvement around the ear area and the outline of the model is noticeable.

5.4 Performance Evaluation

Table 5.1 summarizes the results of some of our experiments on mapping for different models (Stanford bunny, monkey and frog) using both the Thermal and the Laplacian smoothing initialization. The number of iteration refers to the optimization phase, while the time refers to the total time for both deriving the initial mapping and for performing optimization. We observe that the Laplacian smoothing initialization yields a much faster convergence in the optimization phase (half the number of iterations and 50% faster). Our extensive experiments indicate that the number of iterations increases quadratically over the number of faces of the polyhedral representation for triangular models. This is a considerable overhead but it can be calculated off line during a preprocessing phase and stored along with the polyhedral representation. Table 5.2 show the results for the same set of experiments for the same model with different LODs ranging from 854 faces up to 5610 for the monkey model. This set of experiments confirms the above observations.

As mentioned in Section 2.2 merging takes in average $O(I \log I)$ time, where I is the

Table 5.1: Experimental results of mapping with different models of various level of detail

model	method	# vertices	# faces	# constraints	# iterations	time (secs)
Monkey	Laplace	429	854	2991	36	10.9
Monkey	Thermal	429	854	2991	78	22.6
Bunny(Lod1)	Laplace	440	876	3068	94	24.3
Bunny(Lod1)	Thermal	440	876	3068	165	52.0
Frog(Lod1)	Laplace	1964	3924	13736	70	422.2
Frog(Lod1)	Thermal	1964	3924	13736	152	895.8

Table 5.2: Experimental results with the same model with different levels of detail

model	method	# vertices	# faces	# constraints	# iterations	time (secs)
Monkey(Lod1)	Laplace	429	854	2991	36	10.9
Monkey(Lod1)	Thermal	429	854	2991	78	22.6
Monkey(Lod2)	Laplace	703	1402	4909	26	21.3
Monkey(Lod2)	Thermal	703	1402	4909	70	54.8
Monkey(Lod3)	Laplace	1404	2804	9816	49	151.3
Monkey(Lod3)	Thermal	1404	2804	9816	91	271.3
Monkey(Lod4)	Laplace	2807	5610	19637	79	934.0
Monkey(Lod4)	Thermal	2807	5610	19637	136	1578.6

number of intersections. For all cases in Tables 5.1 and 5.2 this step took less than 2.5 sec. Finally, the interpolation step is implemented in GPU so it is very fast and can accommodate almost unlimited number of frames.

Figures 5.1, 5.2, 5.5, 5.11, 5.15, 5.8 and 5.17 illustrate different cases of metamorphosis. We have performed the experiments on well-known models such as the Stanford bunny [32], the Blender monkey [5] and the Aim@shape frog [1].

Table 5.3: Feature alignment optimization

model1	model2	# features	# faces1	# faces2	# constraints	# iters	time (secs)
fish	duck	23	4994	1926	9632	123	14.34
Head	Man. Head	26	25990	11040	55202	34	39.9
Man. Head	Caesar	60	11040	13530	67652	182	321.1

Table 5.4: Experimental results

model	method	# vertices	# faces	# constraints	# iterations	time (secs)
Duck	Laplace	965	1926	6743	19	29.4
Fish	Laplace	2499	4994	17481	108	1007.9
Man. Head	Laplace	5522	11040	38642	28	1377.9
Kykladitiko	Laplace	8401	16798	58795	61	6504.1

Chapter 6

Conclusions

6.1 Conclusions and Future Work

We have presented a method that performs morphing between arbitrary genus-0 objects (homeomorphic to the sphere) without any user intervention. The final morphing sequence blends not only the topologies of the corresponding models but also the textures and the normal maps resulting in more visual pleasing result. The sphere mapping can be considered as preprocessing and stored along with the representation of the solid. The merging is very fast in the average case, and the interpolation is implemented with GPU shaders. Finally, we have presented a fully automated technique for feature matching and alignment that greatly improves the visual effect by matching geometrically important features of the models. We have used our method very successfully on object pairs of similar topology (for examples busts) or of very different topology (fish and duck). In addition we have tested successfully our method in real models from 3D scanners.

We are currently exploring the feasibility of parallelization through GPUs of the merging and the optimization phase. Modern graphic hardware can be used in general processing and several options exists that could improve the computational efficiency of the implementation. More specifically, the initial mapping computation on the sphere can be moved on the GPU along with the merging process. In addition, the optimization problem can be simplified by adding more constraints on the vertices. In particular, certain vertices belonging to the convex hull can be fixed in their corresponding normalized positions and reduce the degrees of freedom for the optimization problem. In the context of optimization efficiency a tailored solver instead of a general one can be used optimized for the particular problem that could cut down the times required for optimization drastically.

Finally, concerning the interpolation step more advanced methods can be used for more visual pleasing results and to avoid the self intersection problem during the morphing sequence. There exist methods inspired from physics that can treat the models as soft bodies and the movement of the vertices as pressure forces acting on the vertices that can totally eliminate the problem of self intersections.

BIBLIOGRAPHY

[1] Aim@shape. AIM@SHAPE Shape Repository v4.0, Department of Genova, Institute for Applied Mathematics and Information Technologies, CNR, http://shapes.aimatshape.net, AIM@SHAPE Project.

[2] M. Alexa. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1):26–37, 2000.

[3] M. Alexa. Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2):173–197, 2002.

[4] T. Beier and S. Neely. Feature-based image metamorhosis. In *Proceedings of SIGGRAPH 1992*, volume 26(2), pages 35–42. New York, Published as Computer Graphics, July 1992.

[5] Blender. Blender Suite, Open Source Suite, http://www.blender.org, Blender Foundation.

[6] Boost. Boost, www.boost.org.

[7] Ch. Brechbuhler, G. Gierig, and O. Kubler. Parametrization of closed surfaces for 3d shape description. *Computer Vision and Image Understanding*, 61(2):154–170, August 1995.

[8] COIN-OR. IPOPT, http://www.coin-or.org.

[9] D. Eberly. Estimating a Tangent Vector for Bump Mapping, www.geometrictools.com/,2003.

[10] T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In *Proceedings of Vizualization 1994*, pages 85–92, Washington D.C., October 1994. ieee.

[11] Ilja Friedel and Peter Schröder and Mathieu Desbrun. Unconstrained spherical parameterization. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 134, New York, NY, USA, 2005. ACM.

[12] T. Kanai, H. Suzuki, and F. Kimura. 3d geometric metamorhosis based on harmonic map. In *Proceedings of the 5th Pacific Computer Graphics and Applications*. IEEE, 1997.

[13] P. Kanonchayos, T. Nishita, S. Yoshihisa, and T. L. Kunii. Topological morphing using reeb graphs. In *CW '02: Proceedings of the First International Symposium on Cyber Worlds (CW'02)*, page 0465, Washington, DC, USA, 2002. IEEE Computer Society.

[14] J. R. Kent, W. E. Carlson, and R. E. Parent. Shape transformation for polyhedral objects. In *Proceedings of SIGGRAPH 1992*, volume 26(2), pages 47–54. New York, Published as Computer Graphics, July 1992.

[15] Vladislav Kraevoy and Alla Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, 23(3):861–869, 2004.

[16] LAPACK. http://www.netlib.org/lapack/.

[17] T. Y. Lee and P. H. Huang. Fast and intuitive metamorphosis of 3d polyhedral models using smcc mesh merging scheme. *IEEE Transactions of Visualization and Computer Graphics*, 9(1):85–98, 2003.

[18] Tong-Yee Lee, Chih-Yuan Yao, Hung-Kuo Chu, Ming-Jen Tai, and Cheng-Chieh Chen. Generating genus-n-to-m mesh morphing using spherical parameterization: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):433–443, 2006.

[19] A. Lerios, C. D. Garfinkle, and M. Levoy. Feature-based volume metamorhosis. In *Proceedings of SIGGRAPH 1995*, pages 449–456. ACM SIGGRAPH, 1995.

[20] C. H. Lin and T. Y. Lee. Metamorphosis of 3d polyhedral models using progressive connectivity transformations. *IEEE Transactions of Visualization and Computer Graphics*, 11(1):2–12, 2005.

[21] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH 87, published as Computer Graphics*, pages 163–169. ACM SIGGRAPH, 1987.

[22] M. Oka, K. Tsutsui, O. Akio, K. Yoshitaka, and T. Takashi. Realtime manipulation of textured mapped surfaces. In *Proceedings of SIGGRAPH 83, published as Computer Graphics*, volume 21(4), pages 181–188. ACM SIGGRAPH, Anaheim, 1983.

[23] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 340–349, New York, NY, USA, 2003. ACM.

[24] M. Rosenfeld. Special effects production with computer graphics and video techniques. In *Proc. SIGGRAPH 87, Course Notes #8.* ACM SIGGRAPH, July, Anaheim 1987.

[25] Shadi Saba, Irad Yavneh, Craig Gotsman, and Alla Sheffer. Practical spherical embedding of manifold triangle meshes. In *Proceedings of the International Conference on Shape Modeling and Applications 2005*, pages 258–267, 2005.

[26] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. *ACM Trans. Graph.*, 23(3):870–877, 2004.

[27] Avner Shapiro and Ayellet Tal. Polyhedron realization for shape transformation. *The Visual Computer*, 14(8/9):429–444, 1998.

[28] A. Sheffer, C. Gotsman, and N. Dyn. Robust spherical parameterization of triangular meshes. *Computing*, 72(1-2):185–193, 2004.

[29] T. Shou and K. Shimada. An angle-based approach to two-dimensional mesh smoothing. In *Proceedings of the 9th International Meshing Roundtable*, pages 373–384, 2000.

[30] V. Stamati. Reconstructing feature based cad models based on point cloud morphology. In *PhD Thesis*. Department of computer Science, University of Ioannina, October 2008.

[31] V. Stamati and I. Fudos. A feature-based approach to re-engineering objects of freeform design by exploiting point cloud morphology. In *Proc. SPM 2007*. ACM, Beijing, China 2007.

[32] Stanford. The Stanford 3D Scanning Repository, Stanford University, http://graphics.stanford.edu/data/3Dscanrep, Stanford Computer Graphics Laboratory.

[33] D. Vartziotis, T. Athanasiadis, I. Goudas, and J. Wipper. Mesh smoothing using the geometric element transformation method. *Computer Methods in Applied Mechanics and Engineering*, 197:3760–3767, 2008.

[34] Andreas Wachter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. In *Mathematical Programming 106*, pages 25–57, 2006.

[35] WolframResearch. Mathematica, http://www.wolfram.com.

[36] M. Zwicker and C. Gotsman. Meshing point clouds using spherical parameterization. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, Zurich, June 2004.

Appendix

A. Tangent Vectors for Normal Mapping

This section briefly describes the estimation of a tangent vector for normal mapping based on the analysis in [9]. Consider a triangle with vertices P_0, P_1 and P_2 and with corresponding texture coordinates (u_0,v_0),(u_1,v_1) and (u_2,v_2). Any point on the triangle may be represented as $P(s,t) = P_0{+}s(P_1{-}P_0){+}t(P_2{-}P_0)$ where $s \geq 0, t \geq 0, s{+}t \leq 1$. The texture coordinate corresponding to this point is similarly represented as $(u(s,t), v(s,t)) = (u_0, v_0) + s((u_1, v_1) - (u_0, v_0)) + t((u_2, v_2) - (u_0, v_0)) = (u_0, v_0) + s(u_1 - u_0, v_1 - v_0) + t(u_2 - u_0, v_2 - v_0$. Consequently, the surface defined by $P(s,t)$ is implicitly dependant on the parameters u and v. The problem is to estimate a tangent vector relative to u or v. To estimate the tangent with respect to u, we must compute the rate of change of P as u varies, more specifically the derivative $\frac{\partial P}{\partial u}$. Using the chain rule from calculus,

$$\frac{\partial P}{\partial u} = \frac{\partial P}{\partial s}\frac{\partial s}{\partial u} + \frac{\partial P}{\partial t}\frac{\partial t}{\partial u} = (P_1 - P_0)\frac{\partial s}{\partial u} + (P_2 - P_0)\frac{\partial t}{\partial u} \tag{6.1}$$

The equation that relates s to u can be written as a system of two linear equations with two unknowns

$$\begin{bmatrix} u_1 - u_0 & u_2 - u_0 \\ v_1 - v_0 & v_2 - v_0 \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \tag{6.2}$$

Inverting this leads to

$$\begin{bmatrix} s \\ t \end{bmatrix} = \frac{1}{(u_1 - u_0)(v_2 - v_0) - (u_2 - u_0)(v_1 - v_0)} \begin{bmatrix} v_2 - v_0 & -(u_2 - u_0) \\ -(v_1 - v_0) & u_1 - u_0 \end{bmatrix} \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \tag{6.3}$$

Computing the partial derivative with respect to u produces

$$\begin{aligned} \begin{bmatrix} \frac{\partial s}{\partial u} \\ \frac{\partial t}{\partial u} \end{bmatrix} &= \frac{1}{(u_1 - u_0)(v_2 - v_0) - (u_2 - u_0)(v_1 - v_0)} \begin{bmatrix} v_2 - v_0 & -(u_2 - u_0) \\ -(v_1 - v_0) & u_1 - u_0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \\ &\frac{1}{(u_1 - u_0)(v_2 - v_0) - (u_2 - u_0)(v_1 - v_0)} \begin{bmatrix} v_2 - v_0 \\ -(v_1 - v_0) \end{bmatrix} \end{aligned} \tag{6.4}$$

Combining this into the partial derivative for P from 6.1, we finally have

$$\frac{\partial P}{\partial u} = \frac{(v_2 - v_0)(P_1 - P_0) - (v_1 - v_0)(P_2 - P_0)}{(u_1 - u_0)(v_2 - v_0) - (u_2 - u_0)(v_1 - v_0)} = \frac{(v_1 - v_0)(P_2 - P_0) - (v_2 - v_0)(P_1 - P_0)}{(v_1 - v_0)(u_2 - u_0) - (v_2 - v_0)(u_1 - u_0)} \tag{6.5}$$

B. Layout of the data in the GPU

We use simple linear interpolation to produce the final morphing sequence. The advantage of linear interpolation besides its simplicity is that it can be easily and very efficiently adapted to GPU techniques since it can be implemented as a simple morphing shader interpolating vertices and features (lighting, textures) in real-time. The required interpolated attributes are vertices,texture coordinates, normals and tangent vectors and are calculated as

$$\begin{aligned} V_m &= V_a + t * (V_b - V_a) = V_a + t * V_{ba}, t \in [0,1] \\ UV_m &= UV_a + t * (UV_b - UV_a) = UV_a + t * UV_{ba}, t \in [0,1] \\ N_m &= N_a + t * (N_b - N_a) = N_a + t * N_{ba}, t \in [0,1] \\ T_m &= T_a + t * (T_b - T_a) = T_a + t * T_{ba}, t \in [0,1] \end{aligned} \tag{6.6}$$

Bi tangent vectors are not required to be stored and can be computed as the cross product of the normal and the tangent vector. From 6.6 8 vectors (3 floats each) should be stored for each interpolated vertex of the mesh. To improve the efficiency of the rendering process and reduce the memory overhead resulting from continuously feeding the GPU with data and since our data are static and can be precomputed, we store the data required in a static Vertex Buffer Object (*VBO*). A VBO is an OpenGL extension that provides methods for uploading data (vertex,normal vector,color, etc) to the video device. VBOs offer substantial performance gains over directly rendering the data, mainly because the data are stored in the video memory rather than the system memory so it can be rendered directly by the video device. This is especially true for static data like in our case. Since in OpenGL a lot of standard vertex attributes are already defined (normal,color,etc) we can use that in our advantage by assigning our data in unused attributes. To this end we store the 8 vectors in a VBO object as follows

Attribute	GPU Attribute	Description
UV_a	Texture Unit 0	Texture Coordinates
UV_{ba}	Texture Unit 1	Texture Coordinates
T_a	Texture Unit 2	Tangent Vector
T_{ba}	Texture Unit 3	Tangent Vector
V_{ba}	Texture Unit 4	Vertex Coordinates
N_{ba}	Color	Normal Vector
N_a	Normal	Normal Vector
V_a	Vertex	Vertex Coordinates

The use of GPU shaders and modern rendering techniques allowed us to effectively handle and render millions of morphed triangles.

C. GPU Shaders Code

Fragment Shader code:

```
uniform sampler2D texture1;
uniform sampler2D texture2;
uniform sampler2D texture3;
uniform sampler2D texture4;

//Param variable controls the percentage of the morphing (0-1.0)
uniform float Param;
varying vec3 lightDir;
varying vec3 halfVector;

void main()
{
    const float overBright = 1.0;

    vec3 halfV, lightColor ;
    float NdotHV, NdotL;
    vec4 tex0 = texture2D(texture1, gl_TexCoord[0].st);
    vec4 tex1 = texture2D(texture2, gl_TexCoord[1].st);

    //Compute the normal vectors from the normal maps
    vec3 n0 = 2.0 * texture2D(texture3, gl_TexCoord[0].st).rgb - 1.0;
    vec3 n1 = 2.0 * texture2D(texture4, gl_TexCoord[1].st).rgb - 1.0;

    //Calculate the interpolated normal
    vec3 n = normalize(mix(n0,n1,Param));

    NdotL = max(dot(n,normalize(lightDir)),0.0) * overBright;
    lightColor = gl_LightSource[0].ambient.rgb;

    if (NdotL > 0.0) {
        lightColor += gl_LightSource[0].diffuse.rgb * NdotL;

        halfV = normalize(halfVector);
        NdotHV = max(dot(n,halfV),0.0);
        lightColor += gl_LightSource[0].specular.rgb *
        pow(NdotHV, gl_FrontMaterial.shininess);
    }

    gl_FragColor = mix(tex0,tex1,Param) * vec4(lightColor,1.0);
}
```

Vertex Shader code:

```
uniform float Param;
varying vec3 lightDir;
varying vec3 halfVector;

void main()
{
    gl_TexCoord[0] = gl_MultiTexCoord0; //Material of object 1 coordinates
    gl_TexCoord[1] = gl_MultiTexCoord1; //Material of object 2 coordinates

    //Calculate new interpolated world pos
    vec4 worldPos = gl_Vertex + vec4(gl_MultiTexCoord4.xyz,0.0)*Param;
    vec3 aux;

    //Calculate the interpolated TBN matrix
    vec3 n = gl_NormalMatrix * (gl_Normal + gl_Color.xyz*Param);
    vec3 t = gl_NormalMatrix * (gl_MultiTexCoord2.xyz + gl_MultiTexCoord3.xyz * Param
    vec3 b = cross(n,t);

    if( gl_LightSource[0].position.w > 0.0 ) //Point Light
        aux = (gl_LightSource[0].position - worldPos).xyz;
    else //Directional Light
        aux = normalize(gl_LightSource[0].position.xyz);

    //Transform light space to tangent space
    lightDir.x = dot(aux,t);
    lightDir.y = dot(aux,b);
    lightDir.z = dot(aux,n);

    halfVector = normalize(gl_LightSource[0].halfVector.xyz);

    halfVector.x = dot(halfVector,t);
    halfVector.y = dot(halfVector,b);
    halfVector.z = dot(halfVector,n);

    gl_Position = gl_ModelViewProjectionMatrix * worldPos;
}
```

Index

Author's Publications

1. D. Vartziotis, T. Athanasiadis, I. Goudas, and J. Wipper. Mesh smoothing using the geometric element transformation method. *Computer Methods in Applied Mechanics and Engineering*, 197:3760–3767, 2008.

Short Vita

Theodoros Athanasiadis was born on May 14, 1983 in Thessaloniki. He received his BSc degree from the Department of Computer Science of the University of Ioannina in 2005. He is currently pursuing his MSc thesis. His reasearch interests include graphics, CAD/CAM systems, finite elements analysis and software engineering.