MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Technical Report No. 1650 December 1998
C.B.C.L Memo No. 167 December 1998

Three-Dimensional Correspondence

Christian R. Shelton
cshelton@ai.mit.edu

Abstract

This paper describes the problem of three-dimensional object correspondence and presents an algorithm for matching two three-dimensional colored surfaces using polygon reduction and the minimization of an energy function. At the core of this algorithm is a novel data-dependent multi-resolution pyramid for polygonal surfaces. The algorithm is general to correspondence between any two manifolds of the same dimension embedded in a higher dimensional space. Results demonstrating correspondences between various objects are presented and a method for incorporating user input is also detailed.

Contents

List of Figures

Chapter 1

Introduction

1.1 The General Correspondence Problem

This paper presents a solution to one instance of the correspondence problem. Correspondence problems, stated generally, are of the form: given two instances of a class of objects, find a relation from parts of one object to "corresponding" parts on the other object. The most classic domain of objects for such problems is images. Image registration, mosaicing, optical flow, stereo matching, and image morphing are all versions of the correspondence problem with images.

In the particular example of optical flow[1], the correspondence problem can be refined to: given two images of a moving three-dimensional scene taken in temporal proximity, find a vector field which maps points in the first image to their corresponding points in the second image. Here, corresponding points are points whose intensities result from the same real point in the scene. Figure 1-1 shows a typical example of such a pair of images.

In this work, we will explore a new area of correspondence. In particular, we will focus on correspondence between two-dimensional manifolds in three-dimensional space. While much attention has been paid to correspondences between images and some attention has been given to three-dimensional volumetric correspondences (e.g. between MRI scans, see [4] and [6]), there has been little work on surface corre-

[1]see [1] for a good overview of optical flow algorithms

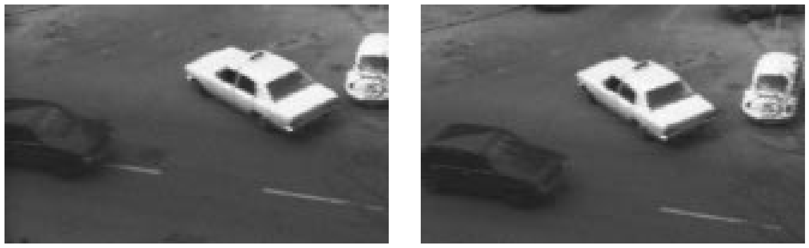


Figure 1-1: Optical flow example

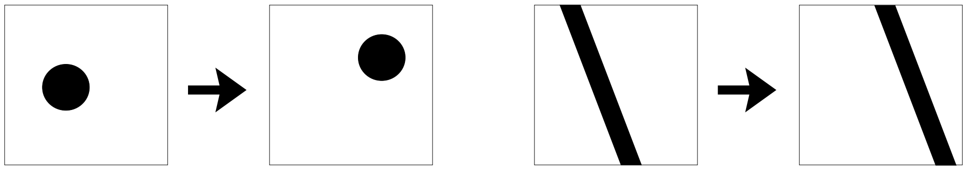


Figure 1-2: The optical flow field for the first image pair is ambiguous since it is uncertain whether the circle rotated about its center as it was translating. The flow field for the second is ambiguous since we cannot recover the translation along the direction of the bar; we can only recover the vector components perpendicular to the edges. These are only simple cases for rigid motions and the assumption that intensity does not change across images. Once we add non-rigid transformation and the ability for the color of a point to change as it moves, the problem becomes much harder.

spondences.

1.2 Correspondence as an Ill-Posed Problem

Correspondence is not a well-posed problem. For example, in optical flow, there are often multiple motions that would explain the same image pair (figure 1-2 gives two such examples) or a point in one image is not visible in the other image (either due to the boundary of the image or occlusion in the image).

The first problem (multiple solutions) is far worse for the case of three-dimensional correspondence. Optical flow is the inverse problem of motion rendering. Thus, while it may be ambiguous which motion caused the image pair, there is a forward problem to which to refer in order to gauge success. In this paper, we are interested in finding correspondences between similar, yet distinct objects. Consider the cars in figure 1-3. There was no process which produced the second car from the first. Thus, whether the correspondence produced by our algorithm is "correct" depends upon a subjective evaluation of what the proper correspondence should be. So, far from it being that there are multiple correct correspondences (as in the case of optical flow), many correspondences are potentially correct. Only once an application which uses the correspondences is chosen, can the quality of the correspondence be evaluated. Although we can not completely remove the subjectivity of the correspondence problem for this paper, in section 1.4 we do give a more concrete definition of what constitutes a "good" correspondence.

The second difficulty in optical flow (points having no correspondence) also exists for three-dimensional correspondence. The rear lights of the two cars in figure 1-3

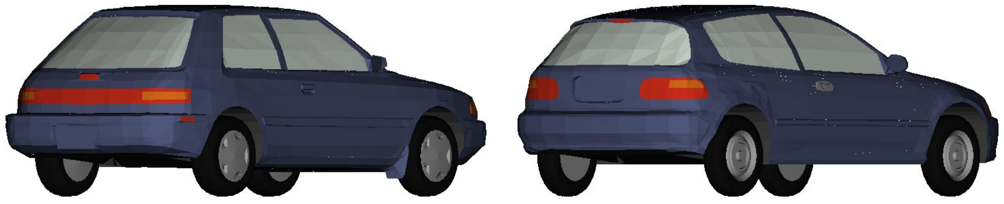


Figure 1-3: An example of two surfaces for which the correspondence is uncertain: one car has one main tail light while the other has two. It is difficult to find the corresponding point for a point in the blue section between the two tail lights on the right car.

provide a good example of this. Since one car has one tail light and the other has two, there is no obviously correct correspondence. We could create some artificial assignment of points on one surface to those on the other, but it would not truly be a correspondence.

Yet, just as these difficulties can be overcome in the image domain to produce useful algorithms, we can also work around the problems of the subjectivity and ambiguity of correspondences to develop a practical algorithm that gives good results.

1.3 Application Areas

There are many applications for a three-dimensional correspondence algorithm. The resulting flow fields can be used to produce a morph between the two objects. This morph is fully three-dimensional and produces viable three-dimensional shapes at each step of the morph. For graphics applications, this allows interesting animation sequences generated automatically or semi-automatically.

Furthermore, by setting a group of shapes from a single class of objects in correspondence with a base shape from the class, we can generate a linear model of that class of objects [13] [12]. Each correspondence between the base shape and another shape represents one flow field. These flow fields can be linearly combined to produce new flow fields which represent novel objects whose shapes are still part of the same class of objects. Thus, if we have a set of surfaces each of a different four-legged animal, by setting them each in correspondence with one arbitrary four-legged animal, we can form a model of four-legged animals in general. Of course this isn't limited to models of four-legged animals; the same technique would work with three-dimensional models of cars, faces, or coffee mugs. Similar work has already been done for the domain of images in the case of faces, handwritten digits, and cars [9].

This parameterized model can be used for a number of interesting applications. New examples of objects in the class can be described in terms of these parameters and easily compared for similarity. Novel models formed from the class can be generated easily by adjusting the parameters (or learned combinations of the parameters). This provides the user with a much simpler method of designing new objects for virtual reality environments or other situations in which novel three-dimensional models are needed.

Lastly, the model provides an easy way of encapsulating information about the shape of a class of objects. This information can be used to aid in three-dimensional reconstruction from images. With the knowledge that the shape being reconstructed is a member of the specified class of objects, the model can be used to help disambiguate or constrain the shape during reconstruction. Thus, we can incorporate prior information about the shape into the reconstruction algorithm.

Just as setting images in correspondence allows for easy comparison of images, finding abnormalities in groups of images, tracking important features across sets of images, or the building of models of types of images, the same operations can be performed on three-dimensional models provided a good correspondence algorithm exists.

1.4 Problem Statement

There are a number of representations for three-dimensional objects. In medical data, often volumetric representations are used: the space is divided into cells and a value given to each cell. By contrast in this work, we will be considering surface models defined by a mesh of polygons, as are often used in computer graphics.

Specifically, we will let a surface consist of a set of triangles in three-dimensional space. These triangles share common vertexes and edges to form one or more two-dimensional manifolds. Further, we will allow a color to be associated with each triangle or, alternatively, each vertex of the object. Triangles, instead of general polygons are used for simplicity since any three points are guaranteed to be coplanar and any polygon can be broken into a set of triangles. Thus, any movement of the vertexes of the triangle will still result in a valid triangle and any surface previously defined in terms of polygons can be converted without changing its shape to a triangle representation.

We will define an object X to be a mesh of triangles defined by the pair (X_K, X_V) where X_V is an ordered set, $\{v_1, v_2, \ldots, v_n\}$, of vertices and X_K is the connectivity of the mesh, or a *simplicial complex* describing the topology of the triangles. For our purposes, we may think of X_K as an ordered set of triangles each described by referencing three vertexes from X_V. [8] has a nice technical description of this mesh representation, but for this paper, we will not need to use that mathematical machinery.

A correspondence algorithm on such meshes will take as input two objects A and B and produce as output a displacement set D of vectors. Each member of D has an associated member in A_V. The object A can be *warped* by the vector set D by adding each member of D to the corresponding vertex location of A. Let us denote the object X warped by the displacement D as $\vec{D}(X)$.

Thus a good correspondence algorithm will produce a D such that $\vec{D}(A)$ will be as similar as possible to B. Furthermore, points on A will be warped by D to be aligned spatially with their corresponding points on B.

The desired correspondence relation which takes a point on the first object and finds the corresponding point on the second object is implied by the output set D. Any point on a triangle of A (not just the vertexes) can be mapped through the deformation of D as described in Appendix A.2. Thus, given a point on A, we map it though the displacement D and then find the closest point on B to this new warped point. Provided that $\vec{D}(A)$ is a good approximation of B and aligns corresponding points well, this will be a good correspondence function.

The algorithm specification above does not allow arbitrary mappings of A to B. Since D only specifies changes to the vertexes, all points on the same triangle in A must be coplanar after A is warped. However, for shapes with enough triangles, this approximation to an arbitrary mapping is sufficient and allows for a tractable method for correspondence computation.

The only task remaining for defining the problem is to specify what is a "correct" correspondence. This requires going back to the application of the correspondences. For this paper, we are going to assume that the purpose of finding the correspondence is to create a morphable model of a class of objects. Thus, since we wish to use the correspondence between A and B to produce new similar objects (e.g. the correspondence between two four legged animals should help us produce a novel four legged animal), we will judge a correspondence by how well the object $\delta\vec{D}(A)$ fits into the class of objects to which both A and B belong for all δ on $[0, 1]$. In this notation, δD refers to the warping field produced by taking D and multiplying all of the vectors by the scalar δ. $\delta\vec{D}(A)$ can be thought of as taking a morph from A to B using the correspondences of D and stopping it in the middle (at the point determined by δ). This "intermediate" object should also look like a reasonable object from the same class as the original two. Although this definition is not concrete enough to directly deduce the proper constraints on an algorithm, it is good enough to evaluate the results and hints at an algorithm.

In section 2, we describe a "colorless" version of the algorithm which assumes the two surfaces are not colored and thus only the geometry need be matched. In section 3, we add two extensions to the basic algorithm: the color component of the shapes is added to the algorithm as a natural extension of the “colorless” version in section 3.1 and user input is optionally incorporated in section 3.2. Section 4 shows some examples of the algorithm on different surfaces and section 5 describes related and future work. The appendix details most of the mathematical derivations: The first section derives a necessary derivative; the second extends coefficients from the first to warp triangles; and the third derives a new operator which extends the notion of a scalar product from vectors to subspaces.

Chapter 2

The Basic Algorithm

In this section we will consider the case where the triangles are uncolored. Thus the problem becomes one of only trying to match two geometric surfaces embedded in three dimensions.

2.1 Energy Minimization

Given the definition of section 1.4, the problem is to find new positions of the vertices of A that best set A in correspondence with B.

We will try to place an ordering on different displacement sets D and then find the "best" D in terms of this ordering. To specify this ordering, we will associate an energy function $E(D)$ with D such that lower values of $E(D)$ correspond to better values of D.[1]

There are two qualities a solution, D, must have. First of all, $\vec{D}(A)$ must have the same shape (or as similar as possible) as B. Secondly, D must represent a "plausible" movement of the shape of A; we would like D to represent a motion from one surface to the other that preserves the common structures between the two shapes throughout the motion. If we were to apply only half of D to A (i.e. instead of applying the displacements in D to the corresponding vertices, apply $1/2$ of the displacements to the vertices), we would like the resulting shape to appear as similar as possible to both A and B and not to be an arbitrary shape having little relationship to the structure of the two input shapes. This comes from our problem definition where we specified that the correspondences will be used to produce a morphable model.

To this end, the energy term describing the quality of a potential solution will have two terms. The first term will measure the similarity of the two surfaces in terms of distance. The second term will measure the changes in the structure of the object.

[1]The energy function also implicitly depends on A and B

2.1.1 Similarity Term

Ultimately, we want every point of $\vec{D}(A)$ to coincide with a point on B and vice-versa. Merely requiring that every point of $\vec{D}(A)$ lie on the surface B is not sufficient as it allows degenerate solutions (e.g. D maps all points to a single point on B). Similarly, only requiring that all points of B lie on $\vec{D}(A)$ will also allow trivial solutions.

For a given point p let us define $\mathrm{d}_X(p)$ to be the square of the distance from p to the closest point on the manifold X. $\mathrm{d}_X(\cdot)$ is a continuous, but not smooth, function over all space. It can be defined as follows:

$$\mathrm{d}_X(p) \triangleq \min_{x \in X} \|p - x\|^2 \tag{2.1}$$

Given p, $\mathrm{d}_A(p)$ can be efficiently computed for the case where A is composed of a set of triangles by placing the triangles of A in a geometric hash-table.

We might like to compute $\int_B \mathrm{d}_A(b)$ as a measure of the total distance from the manifold A to the manifold B. Unfortunately, this can take a lot of computation time. Therefore, instead we chose to approximate the integral with a sum over randomly sampled points. If we let $S_n(A)$ be a set of n points sampled uniformly from the manifold A, we can let our similarity term be:

$$\sum_{s \in S_n(\vec{D}(A))} \mathrm{d}_B(s) + \sum_{s \in S_n(B)} \mathrm{d}_{\vec{D}(A)}(s) \tag{2.2}$$

In practice, we have found that it is best to modify this slightly by adding all of the vertices of $\vec{D}(A)$ and B to the sets of points. Since the vertices are the "most extreme" points on the manifold, it makes sense to insure that their distances are being counted in the sum. Otherwise, this formula tends to "round" the corners of the surfaces. As well, we modify the $\mathrm{d}(\cdot)$ function slightly (renaming it $\mathrm{d}'(\cdot)$). This yields the final formula of

$$E_{sim}(D) = \sum_{s \in S'_n(\vec{D}(A))} \mathrm{d}'_B(s) + \sum_{s \in S'_n(B)} \mathrm{d}'_{\vec{D}(A)}(s) \tag{2.3}$$

where $S'_n(X)$ is the set $S_n(X)$ with the addition of X_V. Letting $N_X(x)$ be the normal to the point x on the surface X, $\mathrm{d}'(\cdot)$ is

$$\mathrm{d}'_X(y) \triangleq \min_{x \in X} \|y - x\|^2 + \rho(1 - (N_Y(y)^T N_X(x))^2) \tag{2.4}$$

assuming that y is from the surface Y.

The addition of the second term in the definition of $\mathrm{d}'(\cdot)$ captures the orientation of the surface at the two points being considered. The square of the inner product of the normals is one if the surfaces are parallel at the point and zero if the surfaces are perpendicular. One minus this term therefore penalizes matching two points on the surfaces whose orientations do not match well.

2.1.2 Structure Term

In order to assure that the correspondence found isn't arbitrary, we will add a structure term. This term should try to preserve the structure and features of the original shape of the manifold. To aid in our explanation of this term, we will define the idea of a *directional spring*. While a normal spring tries to keep the distance between its end points constant with a quadratic energy function, a directional spring tries to keep the vector of the difference of its endpoints a constant with a quadratic energy function. Thus, if a_0 and b_0 are the original endpoints of a directional spring and a and b are the current endpoints of the same directional spring, then the energy associated with that spring is

$$\frac{k}{\|a_0 - b_0\|} \left\|(a_0 - b_0) - (a - b)\right\|^2 \tag{2.5}$$

The fraction in front is the spring constant divided by the original length of the spring. The denominator insures that springs combine in the proper fashion: we would like the energy associated with one spring of length 2 to be the same as the energy of two springs of length 1 placed end-to-end.

While a spring does not penalize rotation or translation, a directional spring allows only translation without an increase in energy. It might seem odd that we are going to use directional springs in our definition of the structure term when rotations of an object should probably be considered as acceptable transformations that do not change the shape. The reason for our choice of directional springs is that they provide the rigidity necessary and help to preserve the volume of the object, which will become more clear later. Our energy term (as a whole) will already have enough local minima that it would be implausible for our minimization technique to be able to match arbitrary rotations of an object. Thus, we will give up our ability to match large rotations for the ability to better match the volumetric shape under small rotations.

Our structure term for the energy function will be composed of the sum of energies of directional springs each connecting two vertices of A. We will add a directional spring from each vertex in A to all adjacent vertices in A (two vertices are adjacent if and only if some triangle of A contains both vertices as vertices of the triangle.). Then, for each vertex in A, we will add directional springs to every other vertex which is closer than the most distant adjacent vertex.

To be more precise, if we let the preposition $\mathrm{ad}_A(\cdot)$ evaluate to true if and only if the arguments are adjacent vertices on the surface A, we can write the set of all pairs of vertices on a surface A connected by directional springs as

$$\begin{aligned} C_A \triangleq \{(a,b) \mid a \neq b \ \wedge\ (\exists c)\ [\ (\mathrm{ad}_A(a,c) \vee \mathrm{ad}_A(b,c))\ \wedge \\ (|a-b| \leq |a-c| \ \vee\ |a-b| \leq |b-c|)]\} \end{aligned} \tag{2.6}$$

If we furthermore define C_A^x as

$$C_A^x \triangleq C_A \cap \{(a,b) \mid a = x\} \tag{2.7}$$

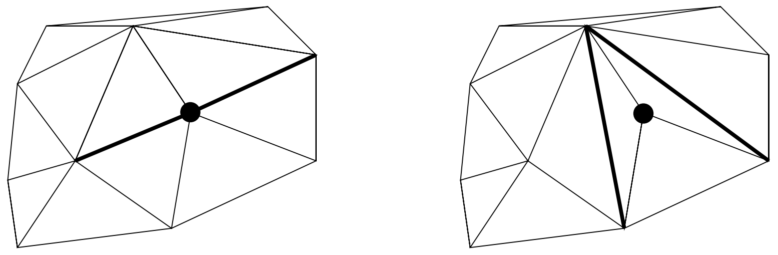


Figure 2-1: Two flat surfaces each tessellated differently (the thick edges mark the difference). The point marked with the circle will have different forces on it due to the different triangulations.

then the energy of the structure term is (taking the sum over the energies of the directional springs)

$$E_{str}(D) = k \sum_{(a,b)\in C_A} \frac{\left\|(a-b) - (\vec{D}(a) - \vec{D}(b))\right\|^2}{\|a-b\|\,|C_A^a|} \tag{2.8}$$

The term $|C_A^a|$ normalizes the energy due to a point by the number of directional springs connected to that point. This helps to insure that a point on the surface does not contribute more to the structural energy solely because it is connected to more points. If we look at the two surfaces defined by the triangles in figure 2-1, they clearly define the same surface and should deform in the same way given the same force on the marked point. Without this normalizing term, the second surface with more connections to the marked point would deform more slowly simply due to a difference in the triangulation of the surface. This normalization does not exactly account for differences in triangulations, but it does an approximate job that has been good enough in practice.

This formula could easily have been written in terms of regular springs.[2] However, tests showed that such an energy term provided far worse results. Since a regular spring has no sense of direction, placing springs along the surface of an object will not, in general, help to keep the shape of the surface intact. Each spring only attempts to keep one end point on a sphere centered around the other end point. This means that if a flat surface is subjected to a compressive force, it will prefer to "buckle" and produce a ridge rather than remain flat and compressed. This is undesirable for most surfaces. Similarly, angles and bends in the surface will not be held in the proper relation to each other to produce the overall surface shape. Each spring is

[2]While regular springs lead to a natural and understood physical model of the properties of a surface, it is an open and interesting question what physical model directional springs describe.

completely local and has no sense of what orientation it should keep relative to the points around it. One solution might be to add "higher order" springs which look at more than two points. This becomes excessively computationally expensive and it is difficult to balance and set all of the parameters correctly. Rather, it is simpler to observe that rotation of the entire object will seldom be realizable and to change to directional springs instead of patching regular springs.

Lastly, there is the question of why we use directional springs between points which are not adjacent. The answer is that we wish to preserve the volume of the shape and the relative positions of the features. If you consider the cars shown in section 4, the tires and bodies of these cars are each independent manifolds. Yet, we would like to try to preserve their relationships to each other. Similarly, for the animals shown in section 4, the feet are close together and their relative positions are important for the overall look of the animals despite the fact that the distance between the feet along the manifold is much farther. Along small thin sections, like the tails of the animals, non-adjacent connections help to preserve the volume on the shape and keep it from collapsing or expanding.

2.1.3 Total Energy Minimization

If we combine the two energy terms from above, we end up with

$$E(D) = E_{sim}(D) + \alpha E_{str}(D) \tag{2.9}$$

where α is used to control the tradeoff between matching the objects and preserving the structure. We will start α off high and gradually anneal, or reduce, its value during the minimization until the two surfaces match well enough. This annealing parameter is common in situations where one would like to minimize one function subject to the minimum of another. In this case, we would like to make E_{str} as small as possible subject to the constraint that E_{sim} is at its global minimum. Clearly the above equation for E does not guarantee it, but it does provide a way of computing a suitable approximation.

We will use gradient descent to minimize $E(D)$. Note that since $E(D)$ already involves a random sampling of the two surfaces, such a minimization already has a stochastic element.

In order to write down the derivative of E with respect to the vector D_p (the vector of D associated with the point p from A), we must first introduce two new definitions. First, we will let $\mathrm{n}_A(x)$ be the set of triangles of A which contain x as a vertex (the neighborhood of x). Second, we will let $\overline{\mathrm{d}'}_\cdot(\cdot)$ be defined as

$$\overline{\mathrm{d}'}_X(y) \triangleq \arg\min_{x \in X} \|y - x\|^2 + \rho(1 - (N_Y(y)^T N_X(x))^2) \tag{2.10}$$

Note that $\overline{\mathrm{d}'}_X(y)$ is exactly the same as $\mathrm{d}'_X(y)$ except that we are taking the arg min instead of the min.

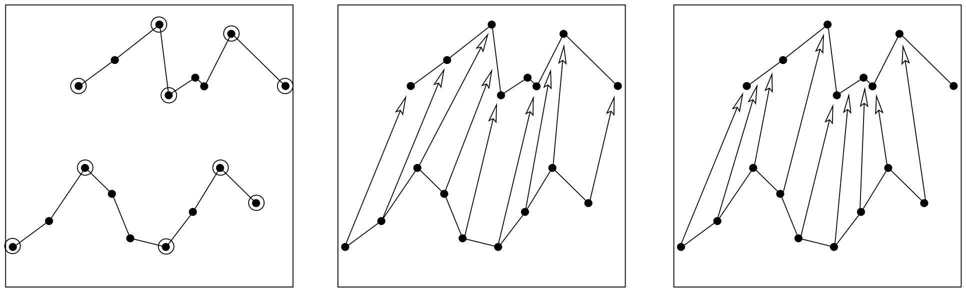


Figure 2-2: Consider the two lines shown in the first frame. We would like to produce the correspondence shown in the second frame. However, due to local minima, we might end up getting stuck with a correspondence more like the third frame. By finding the most important control points (those points circled in the first frame) and only working with them in the initial stages, we can avoid many such local minima.

With those definitions out of the way, we can now derive that

$$\begin{aligned}
-\frac{d}{dD_p}E(D) \;\propto\; & \sum_{s \in (S'_n(\vec{D}(A)) \cap \mathrm{n}_{\vec{D}(A)}(\vec{D}(p)))} (\overline{\mathrm{d}'}_B(s) - s)\frac{L_{p,s}}{K_{p,s}} \\
& + \sum_{s \in \{s \;|\; s \in S_n(B) \wedge \overline{\mathrm{d}'}_{\vec{D}(A)}(s) \in \mathrm{n}_{\vec{D}(A)}(p)\}} (s - \overline{\mathrm{d}'}_{\vec{D}(A)}(s))\frac{L_{p,\overline{\mathrm{d}'}_{\vec{D}(A)}(s)}}{K_{p,\overline{\mathrm{d}'}_{\vec{D}(A)}(s)}} \\
& + \frac{\alpha k}{|C^p_A|}\sum_{(p,a) \in C^p_A} \frac{(p-a) + (\vec{D}(a) - \vec{D}(p))}{\|p-a\|} \qquad (2.11)
\end{aligned}$$

where the ratio L/K is the same ratio derived in Appendix A.1.

Thus, we can minimize $E(D)$ by following the gradient of equation 2.11 with standard gradient descent techniques. We will choose an initial large value for α and gradually reduce it by a constant multiple η (i.e. $\alpha_{t+1} = \eta\alpha_t$) at each iteration of the gradient descent. Initial high values for α will force the algorithm to concentrate on moving A in a consistent fashion. As the algorithm continues and manages to match A to B approximately while keeping the shape roughly the same, α will decrease allowing the algorithm to concentrate on matching the places that could not be matched before with the higher values of α. The algorithm terminates after a fixed number of iterations or after $\vec{D}(A)$ is "close enough" to B.

2.2 Polygon Reduction

The energy minimization described in the above section is only half of the algorithm. There are two major problems with the algorithm as described thus far. First, it takes too long to complete. For objects involving thousands of vertices, it takes a long time to compute the gradient each iteration and many iterations are needed to move the vertices to their final positions. And secondly, the energy function has too many local minima. This is also due to the large number of vertices. Consider the two-dimensional case shown in figure 2-2. Although the best match would be as shown by the arrows in the second frame, clearly there are many solutions like the arrows in the third frame that correspond to local minima of the energy function.

Both of these difficulties can be reduced by finding a smaller set of vertices of the shapes which capture the "essential control points" of the surface. Thus, given a shape as the one in figure 2-2, we would like to find just the most important points to the surface (those marked in the first frame) and work with them first. They will only give an approximate solution since flexibility is more limited when working with this smaller set of vertices. However, such an approximate solution can be used as a starting point for finding a solution involving more control points. If we can pick the proper set of important control points, we will gain a speed up in running time since we can compute much of the movement due to D with fewer vertices, and we will also reduce the number of local minima the energy minimization algorithm might find.

This problem of reducing the complexity of the description of a surface has already been studied in great lengths in the computer graphics literature. Polygon reduction algorithms essentially solve the exact problem we have. The idea is to create a new mesh with fewer triangles and vertices that looks as close as possible to the original mesh. For our work, we chose to use the mesh simplification algorithm of Hoppe called Progressive Meshes [7]. Hoppe's algorithm has a number of nice properties but most importantly for this work is the fact that it tries to preserve the surface shape and not just geometry: it pays attention to the discontinuity curves of the surface and attempts to keep their topology constant throughout the reduction. We will not detail the progressive mesh algorithm here but rather just refer to [7]. It is important to note that polygon reduction algorithms in general (and Hoppe's progressive meshes in particular) can give as output meshes which contain no vertices in the same positions as those of the original object. So, unlike the original formulation where we were to pick out existing control points and use them, we may end up with a completely new set of control points.

2.2.1 Polygon Pyramid

In order to be able to describe the role of polygon reduction more concretely, we will introduce a little more notation. Considering the polygon reduction algorithm as a black box, if we are given a mesh X with v vertices, we will let ${}^{\beta}X$ to be the result of running the reduction algorithm on X to produce a mesh with $2^{-\beta}v$ vertices. Thus, ${}^{0}X$ is the same as X and ${}^{i}X$ has half as many vertices as ${}^{i-1}X$.

To begin the process of creating a correspondence, we first create a "pyramid"[3]

[3]We call this ordered set of shapes a pyramid since it plays a similar role to Gaussian pyramids

of shapes for both A and B, $\{{}^0A, {}^1A, \ldots, {}^{l_1}A\}$ and $\{{}^0B, {}^1B, \ldots {}^{l_2}B\}$. This involves running the polygon reduction algorithm repeatedly until we have a set of reduced versions of both A and B each with half as many vertices as the previous member of the set. The simplest version (a value for l_1 or l_2) is determined by selecting the last reduction which still looks reasonable. Either the user can manually pick the last usable shape or the distance between the reduce shape and the original shape can be used along with a threshold to choose the number of reduction steps automatically.

We then proceed "down" the pyramid creating correspondences between successively more complex meshes using the results from the previous correspondence. We start by creating the correspondence between ${}^{l_1}A$ and ${}^{l_2}B$, which we will denote as ${}^{l_1}_{l_2}D$. We then continue producing each ${}^{i}_{j}D$ for successively smaller values of i and j, decrementing i and j by one each time. At each stage, we use the value of the last produced D as a starting point for our algorithm at the current stage.

Let us first assume that we can extend our definition of D so that it can be used to warp any object (and not just the one for which is was created). This will be shown in the next section. We can then formulate our pyramid scheme as such: Assuming we have ${}^{i+1}_{j+1}D$, we can produce ${}^{i}_{j}D$ with the energy minimization described previously except that we will not take D to be all zeros for the starting position of the gradient descent; we will warp ${}^{i}A$ by ${}^{i+1}_{j+1}D$ to produce the starting position for the energy minimization.

We continue in this fashion, until we have produced ${}^{i}_{j}D$ for $i \leq 0$ and $j \leq 0$. Note that if $l_1 \neq l_2$ then we must define ${}^{k}X$ to be equal to ${}^{0}X$ for all $k < 0$. This final D will be the desired displacement field for the full versions of A and B.

2.2.2 Extending Warps

The above description of the algorithm works well provided we can use D to warp a mesh other than the one for which it was produced. If the vectors of D correspond to the vertices of X, then $\vec{D}(X)$ is simple to compute. But now we wish to expand the warp field to fill the entire space and not just to be defined on the vertices of X.

We can use the triangle warping mechanism described in Appendix A.2 to extend the warp field of D across the surface of X. Namely, given a point, p, on X, we find the triangle which contains p. We then warp the vertices of the triangle using D. We now have two triangles in correspondence and the point p can be mapped from one to the other using the coordinate transformation in Appendix A.2. The difference in the two points (p warped and p) we will call the extension of D across the surface of X.

We can then easily extend D to cover all of space. Given a point p which does not lie on X, we find the closest point on X to p and use the displacement at this projected point as the displacement for p. In this way, any point can be warped

of images used in some optical flow algorithms. Note, however, that the reduction described in this section is data-dependent whereas Gaussian pyramids are constructed in a data-independent way.

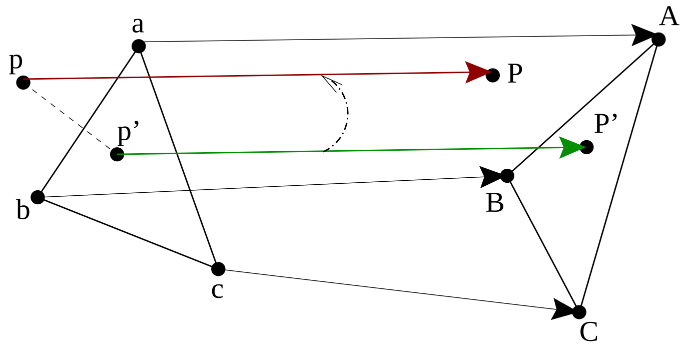


Figure 2-3: How to extend a warp field from the vertices of a shape to all of space: First point p is projected onto the nearest triangle, $\triangle abc$, to produce p'. The point p' is then mapped through the coordinate transformation described in Appendix A.2 producing a new point P'. Finally, the difference $P' - p'$ is taken to be the translation vector for the point p. The green vector is the extension of the warp field across the surface and the red vector is the extension of the warp field to all space (taken by translating the green vector). The black vectors are the original warp field as defined on the vertices of the triangles.

using the displacement field D. Thus, we redefine $\vec{D}(X)$ to be the new, more general, warping which does not require X to be the same as the mesh for which D was produced. Figure 2-3 pictorially demonstrates this extension.

Chapter 3

Extensions to the Basic Algorithm

3.1 Adding Color

3.1.1 Dimension Extension

The algorithm presented in chapter 2 can be extended naturally to handle colored surfaces by moving the vertex positions from three-dimensional space to six-dimensional space. The three original dimensions remain, but now we add an additional three dimensions for color.

In general, any number of dimensions can be used to describe color. In computer graphics, color is often parameterized in terms of three quantities since the human eye perceives only three axes of color. Hue-saturation-value, hue-lightness-saturation, and red-green-blue are all well used axes along which to measure color. The algorithm below will work well with any of these (or any other) axes for color,[1] but we shall use the red-green-blue coordinate system for these examples.

Thus, each vertex now has six values describing its position: x, y, z, red, green, and blue. To place all six dimensions in the same space, we need a conversion factor (or scaling) of color units to spatial units. We will define this to be the *color:shape ratio* and denote it by the symbol γ. γ is the value by which to multiply the color components to set them in the same units as the spatial components. Thus it has units of distance per energy. γ has the natural interpretation in this context of being the relative importance of matching color verses shape. It is a fairly simple parameter to set based on the user's knowledge about the coloring of the surface.

We can now perform the algorithm exactly as described in chapter 2 except that the geometry will be performed in $\mathbb{R}^6$ instead of $\mathbb{R}^3$. But, *why is this a good representation for color?* Clearly it is simple and mathematically elegant, but that does not mean it will produce correct results. Yet, let us first consider how we would change the algorithm if color were not automatically encoded as three extra dimensions. In the previous algorithm, instead of matching a point to the other surface based solely on spatial distance, we would add a penalty for matching based on the colors of the

[1]The results may be different since the conversions among these different color coordinates are non-linear.

two points. However, this would not be enough. The color boundaries on surfaces are very important visually and contain a lot of information about the surface; we would like to try to specifically match those as well. Thus, we might sample an extra set of points from along color boundaries and attempt to match those to other color boundaries (again, with a penalty if we could not find a color boundary whose color or orientation exactly matched).

Constructing a surface in $\mathbb{R}^6$ accomplishes exactly those goals. By matching points in a space which also includes color, we automatically include a natural penalty term for matching two points whose colors don't agree: the squared distance in this six-dimensional space has three terms corresponding to the squared distance in the original three-dimensional space plus three terms which penalize differences in color. More subtly, and more importantly, in constructing a complete surface in $\mathbb{R}^6$, we add explicit surfaces along the color boundaries which we then must match (this is demonstrated clearly in the next section). These surfaces are sampled with points just as before and thus lead to a direct mapping of color boundaries. Since we have an explicit term in the energy term for matching orientation of surfaces, this automatically takes into account the different colors along the color boundary and the orientation of the boundary.

As an important extra bonus, by having both color and position in the same coordinate system, the algorithm can change the colors and color boundaries as needed to ensure a match. The algorithm is free to create or remove color boundaries simply by changing the color coordinates of the vertices along with the spatial coordinates. Once again, the structure term of the energy function keeps these changes reasonable.

3.1.2 Constructing the Surface

There are two primary ways of specifying colored surfaces for computer graphics. The easiest case, from our stand point, is when each vertex is assigned a color. In this case, a point in a polygon is colored based on a linear combination of the colors of the vertices. Thus, our surface already naturally lies in $\mathbb{R}^6$ and nothing else needs to be done.

The other case is where each polygon has a single color. Therefore, distinct color boundaries exist between polygons of different colors. In this case, if we look at each triangle in the original model and construct a new triangle in our 6D version such that each vertex of the triangle has spatial coordinates as given in the original model and all three vertices have the same color coordinates as the triangle itself, we will be close but not quite done. After "pulling" the vertices into $\mathbb{R}^6$ by this method, there will now be gaps in the surface where there weren't before. In particular, vertices which previously coincided (since they only had spatial coordinates) will now be distinct if the two triangles from which they came had different colors. This will produce gaps along the lines in the original model where color boundaries were before.

To complete the surface, we add triangles along these gaps. For every color bound-

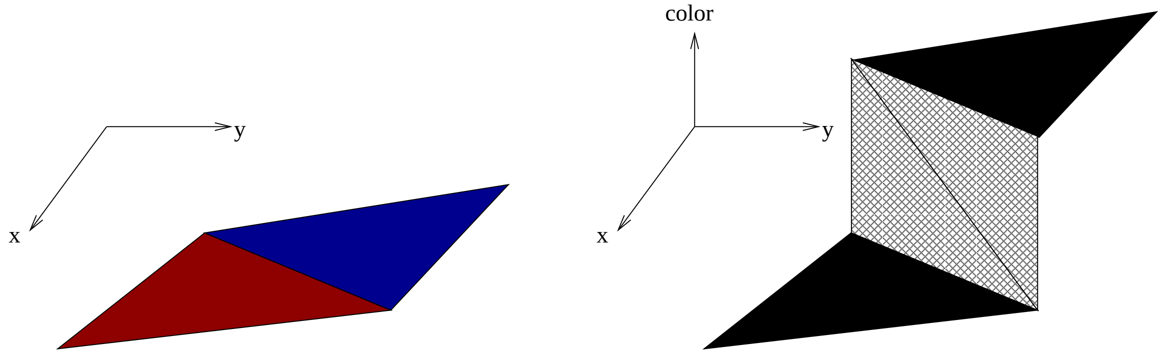


Figure 3-1: Filling the gap caused when triangles are "pulled" into six dimensions. For this simple example, the two triangles lie in two dimensions and there is only one dimension of color. Thus, when the triangles on the left are "pulled" into the third dimension of color the gap shown on the right appears which can be patched by the checked triangles. They have a spatial projection of the color boundary on the surface and a color projection of a line from one color to the other.

ary in the original surface, we will add one rectangle (or rather two triangles) whose spatial projection is the line of the color boundary but who stretches the color difference between the vertices on either side. Figure 3-1 shows this solution for a simple case. This will cover all of the gaps except those at vertices where more than two differently colored polygons touch. In the surfaces used for this paper, no more than three different colors ever met at a single vertex. Thus, it was sufficient for each such point to add a single triangle all of whose vertices shared the same spatial coordinates but each of which had different color coordinates. If more than three colors join at a single point, some decision must be made about the connectivity of those colors and the triangles arranged appropriately since, using only planar objects, it is not possible in general to construct a single polygon which will connect all of the colors.

3.1.3 Algorithmic Implications

The only major difference between working with a two-dimensional surface in $\mathbb{R}^3$ and a two-dimensional surface in $\mathbb{R}^6$ is that there no longer exists a single normal vector for each surface triangle. Although points can still be projected to the nearest point on the surface, distances and derivatives can be computed as before, and the directional springs continue to provide structure, the definition of $\mathrm{d}'()$ as used in section 2.1.1 no longer holds since $N_A(x)$ no longer exists as a single vector.

In general, we would like this technique to extend to arbitrary dimensional surfaces in higher dimensional spaces. Thus, just as we don't want to restrict the algorithm to three-dimensional space, we don't want to restrict it to two-dimensional manifolds either. Returning to the definition of $N_A(x)$ in equation 2.4, we see that we do not explicitly need a normal but rather just a method of measuring the rotational similarity between two subspaces. In Appendix A.3 we describe a new operator, $\diamond$, which is a generalization of the scalar product to subspaces. In equation 2.4, we replace the scalar product of the normals with this new operator and obtain a version which will work for any dimensional subspace:

$$\mathrm{d}'_A(p) \overset{\triangle}{=} min_{a \in A} \|p - a\|^2 + \rho(1 - V_b \diamond V_a) \tag{3.1}$$

where V_x is the matrix of basis vectors (as in the Appendix) describing the linear subspace on which the point x lies on the surface X.

3.2 Adding User Input

Sometimes it is not desirable for the algorithm to be completely automatic. Although the best way of incorporating user knowledge about the desired correspondence would be to encode the types of allowable deformation of the surface into the structure term in the energy function (thus changing the generic one given in Section 2.1.2), sometimes it is easier to specify specific points on the surface and their correspondences. Often the algorithm works for all but a small section of the shape. In this case, it is usually sufficient to mark a few points on each of the two surfaces and specify their correspondences.

Let us denote this type of user input as the set U $\{u_1, u_2, \ldots, u_n\}$ where u_i is the pair (u_i^a, u_i^b): u_i^a is a point on A and u_i^b is the corresponding point on B. We incorporate this user input into the energy function by adding a third term:

$$E_{user} = \zeta \sum_{(u_i^a, u_i^b) \in U} \left\| \vec{D}(u_i^a) - u_i^b \right\|^2 \tag{3.2}$$

This essentially adds a spring of zero rest length from each point picked on surface A to the corresponding point on the surface B. The parameter ζ dictates how rigorously the algorithm will follow the input of the user. Note that when taking the derivative of this new term, the derivative of the summation term involving u_i^a will be spread over the three vertices of the triangle on which u_i^a lies (as detailed in Appendix A.2) in a similar fashion as the sampled points were in the derivative of E_{sim}.

Chapter 4

Results

The algorithm described in the previous chapters has a number of parameters. Figure 4-1 shows the settings for these parameters for each of the examples given in this section. We have tried to keep all of the parameters the same across all experiments. There are only two differences in the setting of the parameters. For the car models, we set γ to be 5.0 instead of 1.0. This is because the car models are all colored in the same fashion so the color of the surface contains a lot of information about correspondences. The only other change is in the energy minimization parameters for the second car correspondence. Due to the larger shape differences between the models in figure 4-3 we needed to run the algorithm for longer. This meant changing t and t_0 which in turn required changing η so that the springs would still decay at the same relative rate. None of these parameters were found to be sensitive to changes; they only needed to be within a factor of 2 or more for the algorithm to work as well as the results seen in this section.

Instead of drawing D directly for each correspondence (which would result in an uninterpretable diagram), we have drawn a sequence of frames of a morph from the first object to the second. Thus, if δD is the result of taking the scalar δ and multiplying each element of D by it, we have drawn $\delta\vec{D}(A)$ for evenly varying values of δ between 0 and 1. These shapes should represent a smooth transition between A and B, and when $\delta = 1$, $\delta\vec{D}(A)$ should be as similar as possible to B.

It should be noted that all of the images in this section are two-dimensional renderings of three-dimensional shapes. The two-dimensional renderings were produced solely for the purposes of inclusion in this paper. Only the full three-dimensional representations are used by the computer for computation and warping. Thus, the correspondences and shapes are three-dimensional and only on a per image basis do we convert them back to a two-dimensional image. The images are rendered using a Lambertian shading model with a single light source at the same position as the camera.

		figure number				
	description	4-6	4-7	4-8	4-9	4-13
γ	color:shape ratio	5.0	5.0	1.0	1.0	1.0
$\alpha_0 \cdot k$	starting spring constant	50.0	50.0	50.0	50.0	50.0
η	spring constant annealing rate	0.995	0.9988	0.995	0.995	0.995
n	number of sampled points	3000	3000	3000	3000	3000
ρ	orientation weighting	1.0	1.0	1.0	1.0	1.0
ζ	user control spring strength	N/A	N/A	50	50	N/A
t	number of gradient steps	100	400	100	100	100
t_0	number of steps on last level	1000	4000	1000	1000	1000
ϵ	gradient step size	0.0005	0.0005	0.0005	0.0005	0.0005

Figure 4-1: Parameter settings for the correspondences shown in this chapter. γ is explained in section 3.1.1, α and η in 2.1.3, n and ρ in 2.1.1, and ζ in 3.2. At each level of the pyramid scheme, we take t steps with the gradient step size at ϵ, except for the final layer (to compute ${}^0_0D$) where we take t_0 steps.

4.1 Colored Surfaces

We chose two different classes of objects for testing the colored surface algorithm. Figures 4-2 and 4-3 show the cars we used as one set of surfaces and figures 4-4 and 4-5 show the four-legged animals we used as another set of surfaces.

For the cars, the correspondences were generated as normal. For the animals, we added 11 user specified points as in the description in section 3.2. These points are shown in figures 4-4 and 4-5: one for each ear, eye, and foot, one for the nose, one for the end of the tail, and one for the start of the tail.

Figures 4-6 and 4-7 both show a generated correspondence between cars. Figures 4-8 and 4-9 show correspondences between animals.

In figure 4-6 the objects appear to be very similar. However, there are some important differences that the algorithm manages to deal with properly. The Mercedes has three side windows, whereas the Lexus has only two. The algorithm nicely removes the extra window by collapsing it to a line instead of changing the color of the middle blue bar (which would have produced unnatural looking middle frames). As well, the rear lights also require change. Again, the algorithm moves the lights instead of changing the color of the surface.

Figure 4-7 shows a morph between two very different cars. In particular, there are many unclear correspondences (e.g. how does one match the orange front lights in a consistent manner with the surrounding body? or how does one make three windows from only two? or what does one do with the rear view mirrors?). The algorithm does a nice job of matching smooth consistent changes (like the side of the car where a step and large wheel hubs must be created) and it does an acceptable job of finding some way of matching the very different front lights and grill structures.

The morph shown in figure 4-8 shows some problems. The eyes are not properly matched and the ears of the dog cannot be formed from the ears of the cat. The tail shows yet another difficulty as the shrinking of the tail is not generally allowed by the directional springs used to control the shape. However, the overall shape and stance of the dog is well matched (including the asymmetric rear legs).

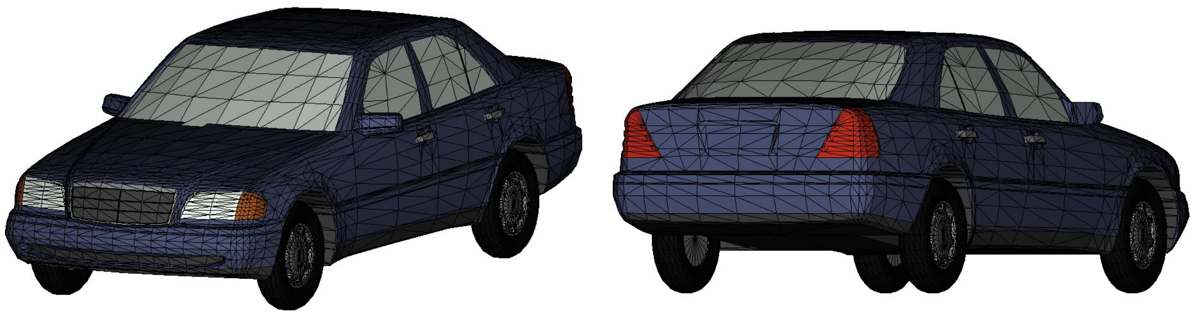
Finally, figure 4-9 demonstrates the most difficult correspondence attempted. The neck shape gives the algorithm clear problems. However, this is most due to a lack of needed flexibility in the starting shape's polygons. Yet, the change in leg stance, facial shape, as well as the obvious difference in back shape are all matched as well as can be expected.

To give a sense of the complete running of the program on this final example, figure 4-10 shows the output of the program at each stage. In this figure, the "base" column shows the polygon reduced versions of the giraffe. The "target" column shows the polygon reduced versions of the camel. Since the camel had fewer polygons, the last three shapes in this column are all the same. The middle two columns show the output of the algorithm after each stage in the pyramid scheme. The "after warping" column is the result of warping the left shape by the correspondence produced at the previous level and the "after energy minimization" is the result of the energy minimization using the "after warping" shape as a starting point. The first row has no "after warping" shape since there was no previous level in the pyramid scheme to use.

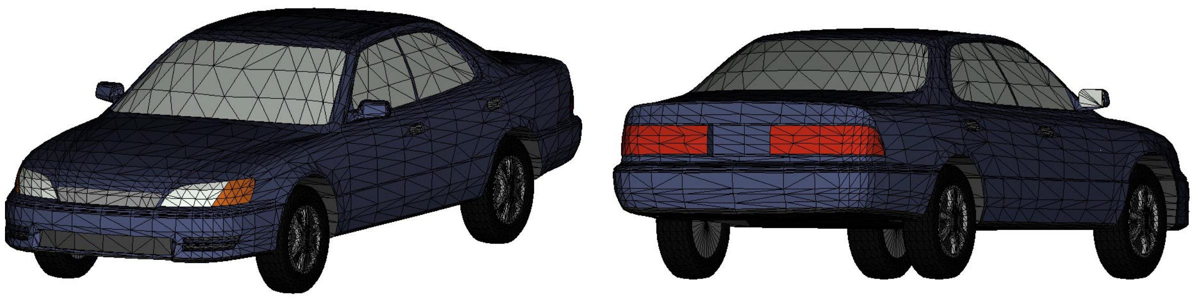
4.2 Images

As a final example, we consider the special case of images. An image is degenerate form of a colored surface in that it lies in a 2D subspace of $\mathbb{R}^3$. We took the two images shown in figure 4-11 and ran exactly the same algorithm as in the other examples. The polygon reduction algorithm produced the results shown in figure 4-12 and the final correspondence is shown in figure 4-13.

Aside from the artifact in the nose, we feel this represents a very good correspondence. Looking at the morph, we can see the light move from the left side to the right side, the stripes are added to the shirt, and there is a smooth transition between the facial pose and expression.

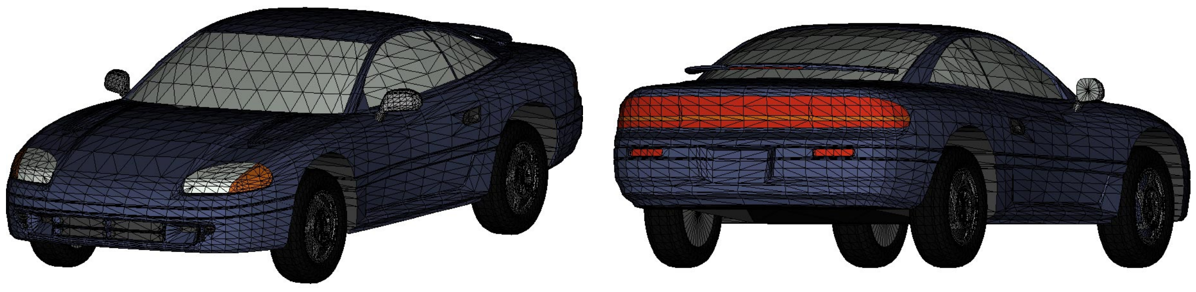


Mercedes 280C '94 (6860 vertices and 13684 faces)



Lexus ES300 '93 (6989 vertices and 13958 faces)

Figure 4-2: Two of the car models


Dodge Stealth ’94 (10568 vertices and 21104 faces)

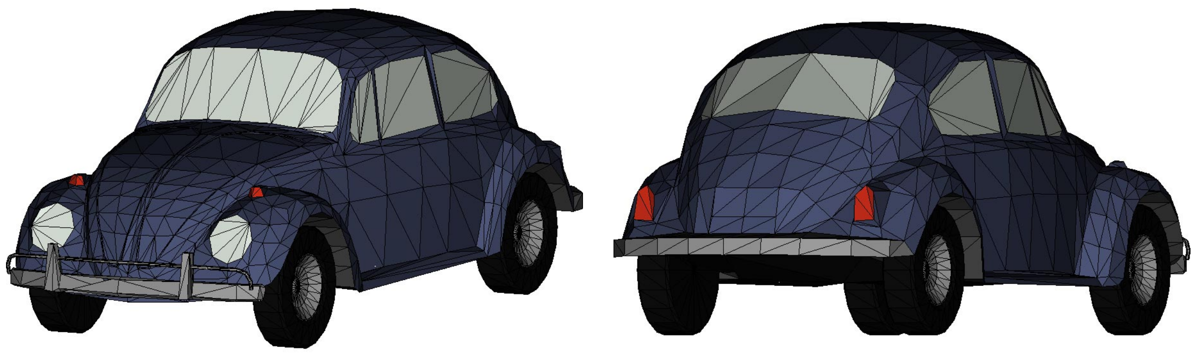
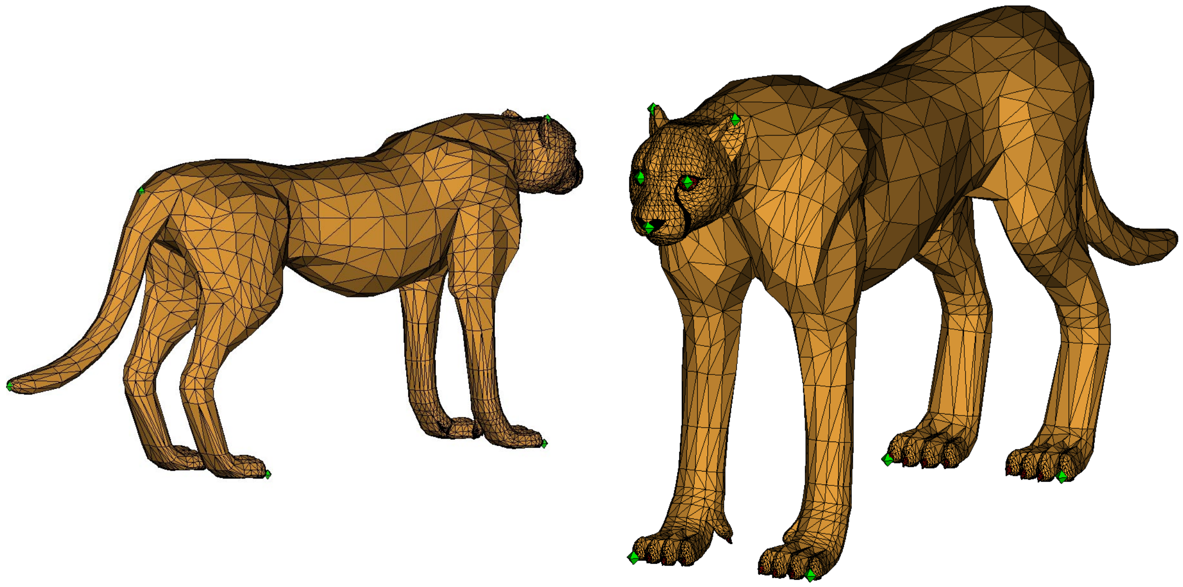
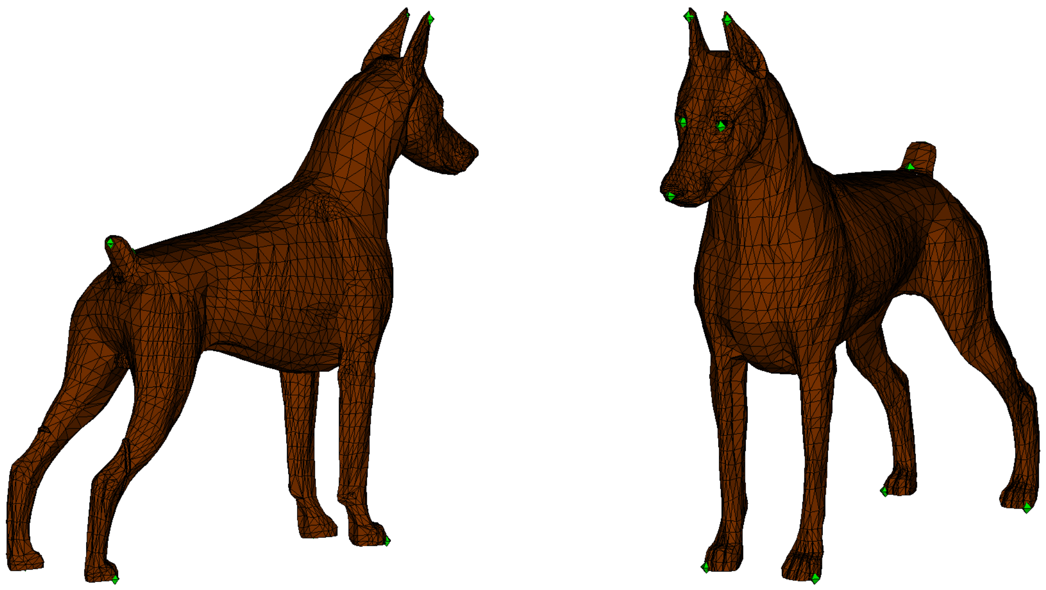

VW Beetle ’70 (1888 vertices and 3748 faces)

Figure 4-3: Two more of the car models

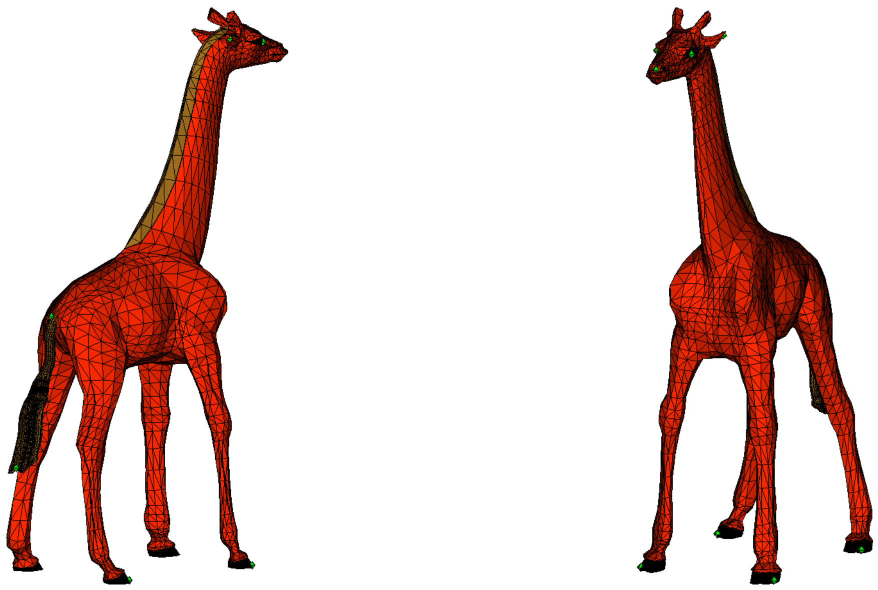


Cheetah (5702 vertices and 11402 faces)

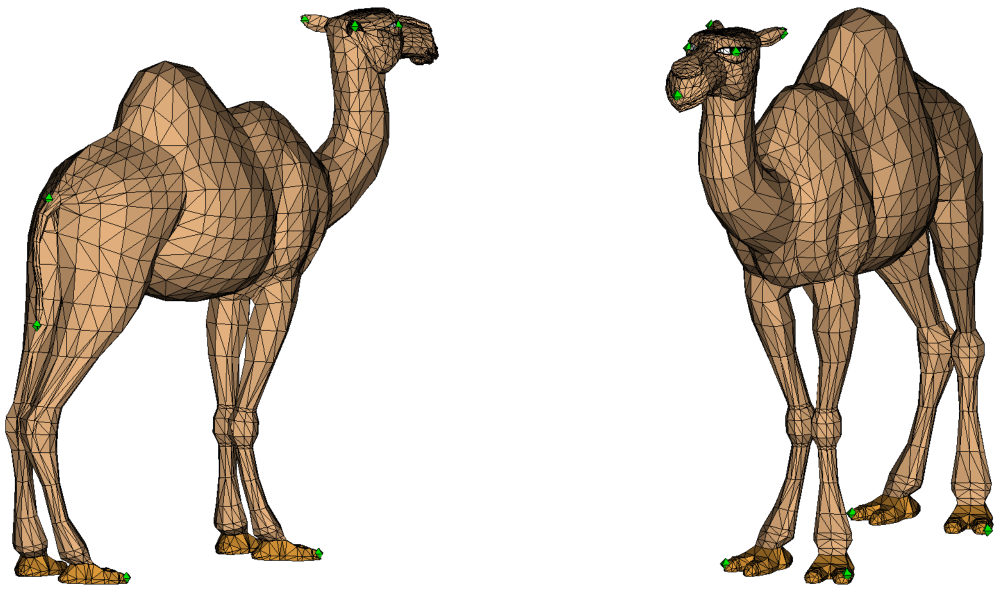


Doberman Pincher (4640 vertices and 9268 faces)

Figure 4-4: Two of the animal models: The green octahedra show the points added as user control points



Giraffe (4595 vertices and 9174 faces)



Camel (2567 vertices and 5132 faces)

Figure 4-5: Two more of the animal models: The green octahedra show the points added as user control points

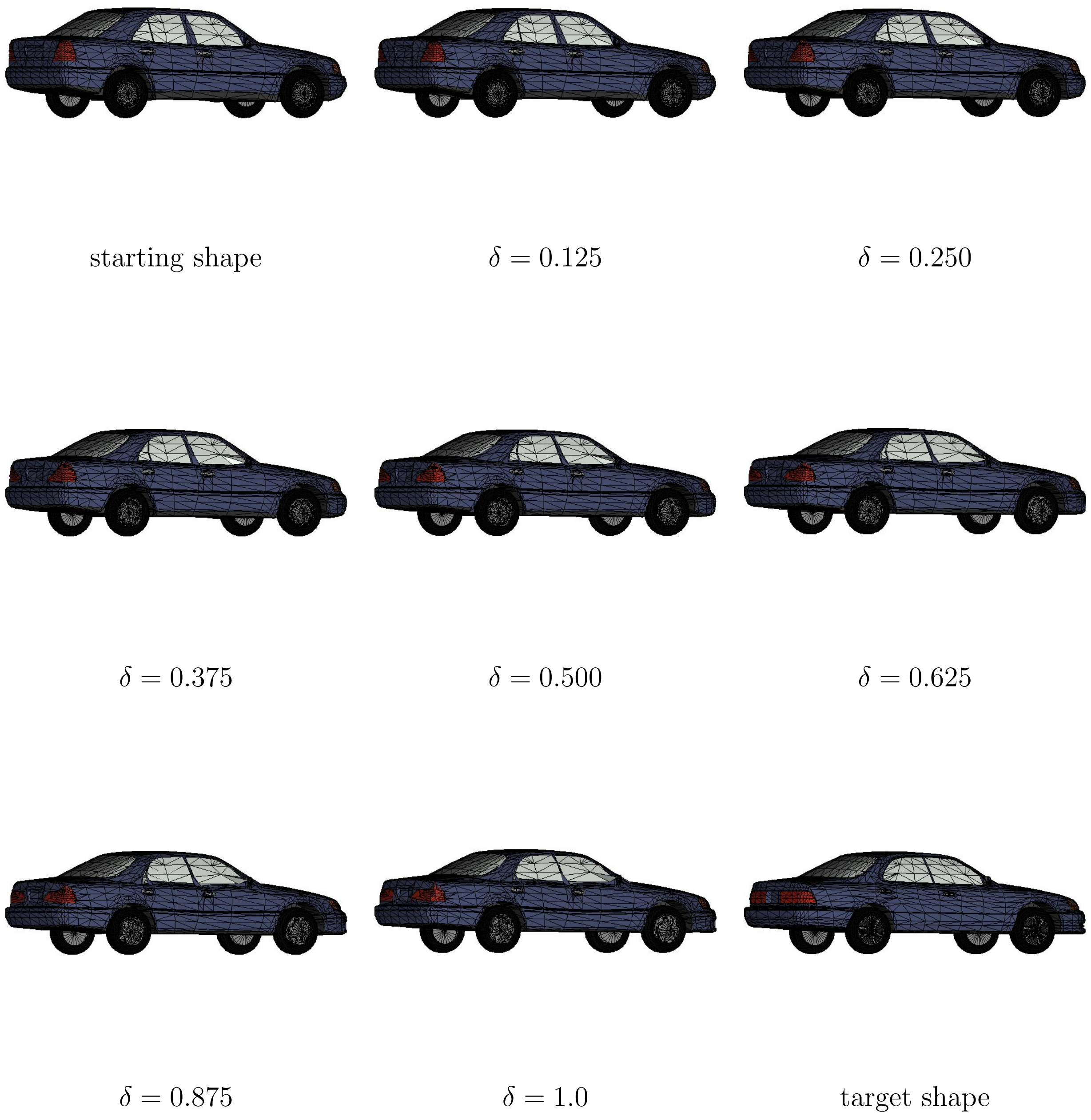


Figure 4-6: Morph showing the generated correspondence between the two cars shown in figure 4-2. The shape in the upper left was specified as A and the shape in the lower right was specified as B. The frames in between represent $\delta\vec{D}(A)$.

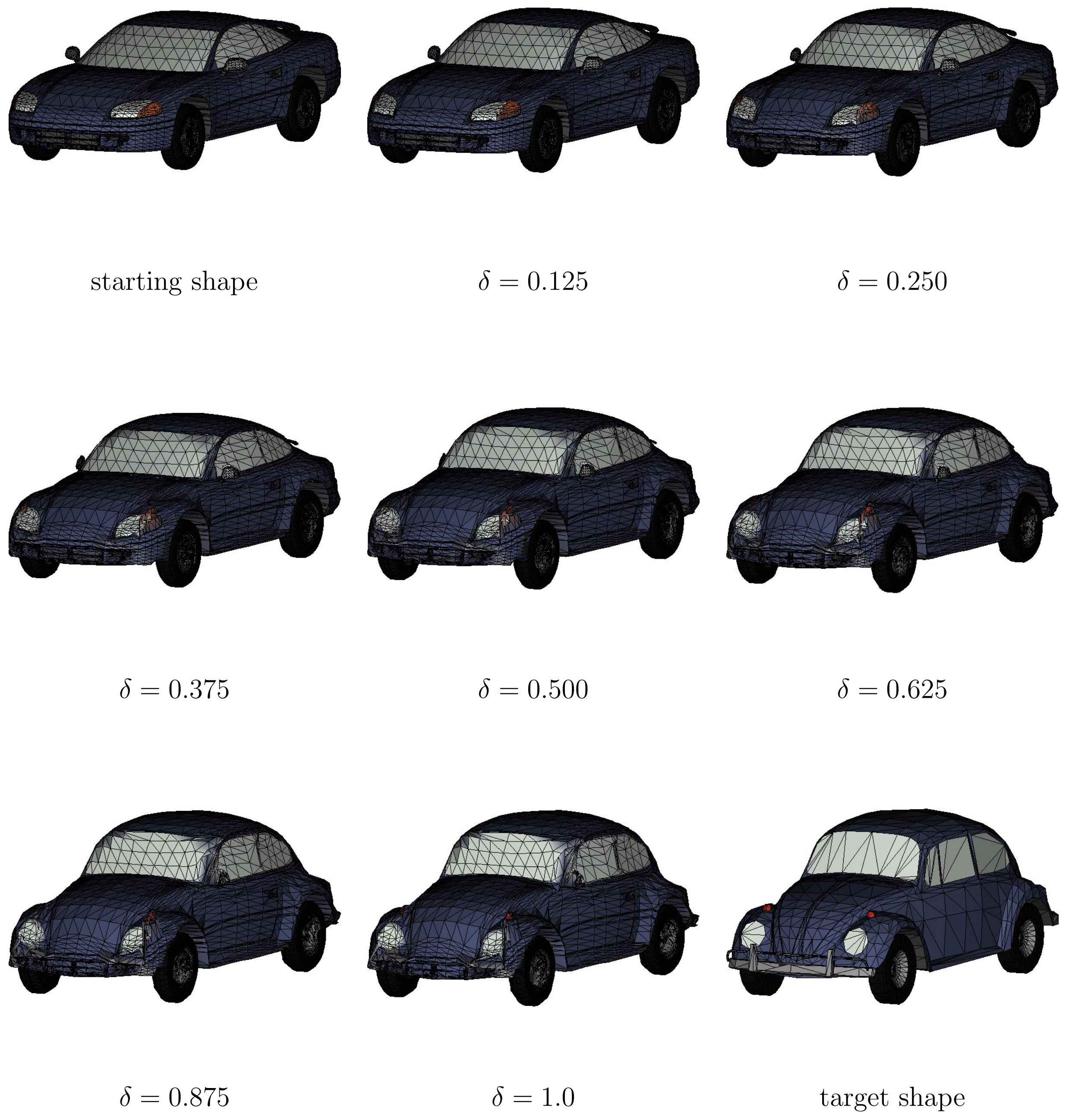


Figure 4-7: Morph showing the generated correspondence between the two cars shown in figure 4-3. The shape in the upper left was specified as A and the shape in the lower right was specified as B. The frames in between represent $\delta\vec{D}(A)$.

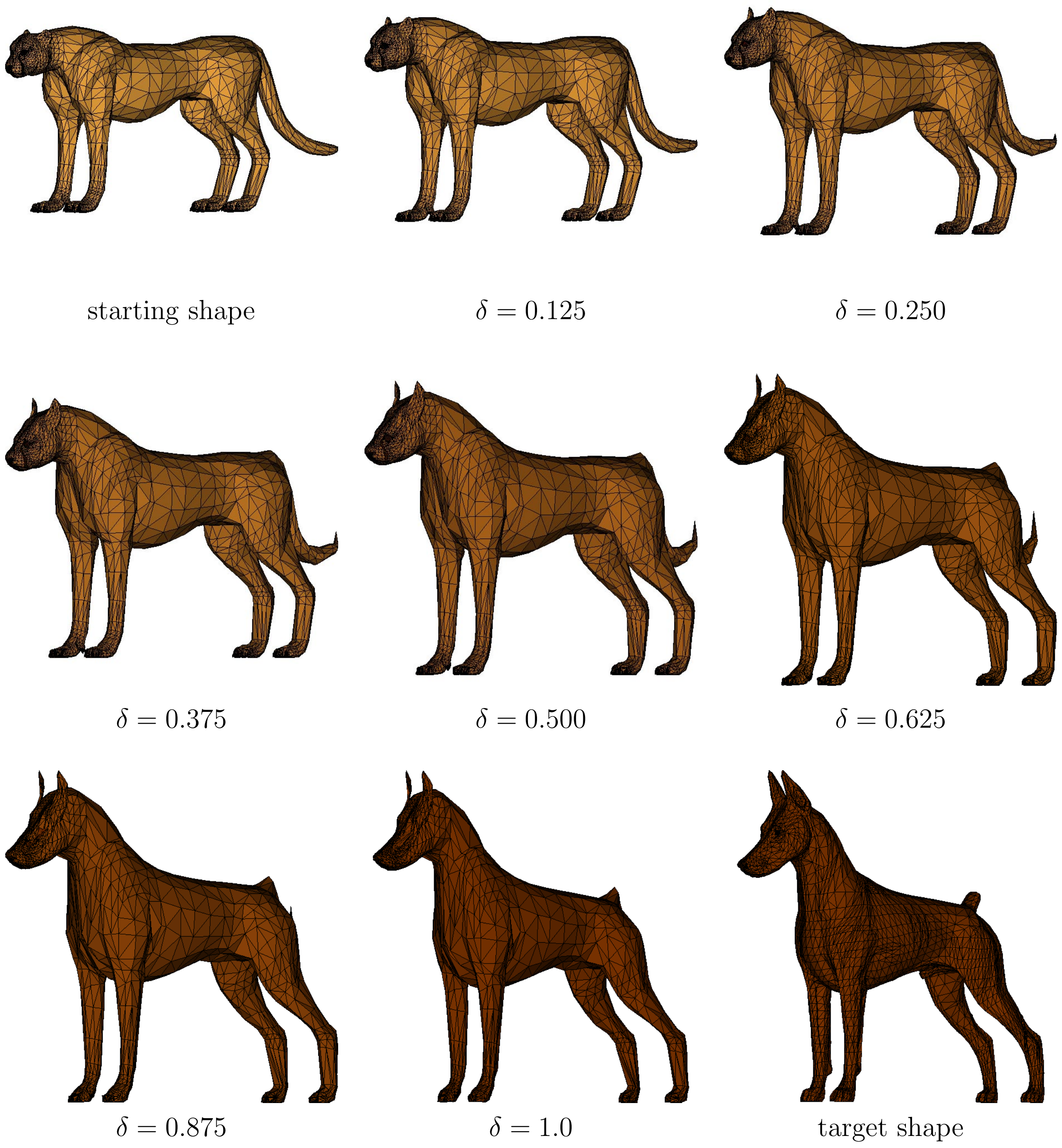


Figure 4-8: Morph showing the generated correspondence between the two animals shown in figure 4-4. The shape in the upper left was specified as A and the shape in the lower right was specified as B. The frames in between represent $\delta\vec{D}(A)$.

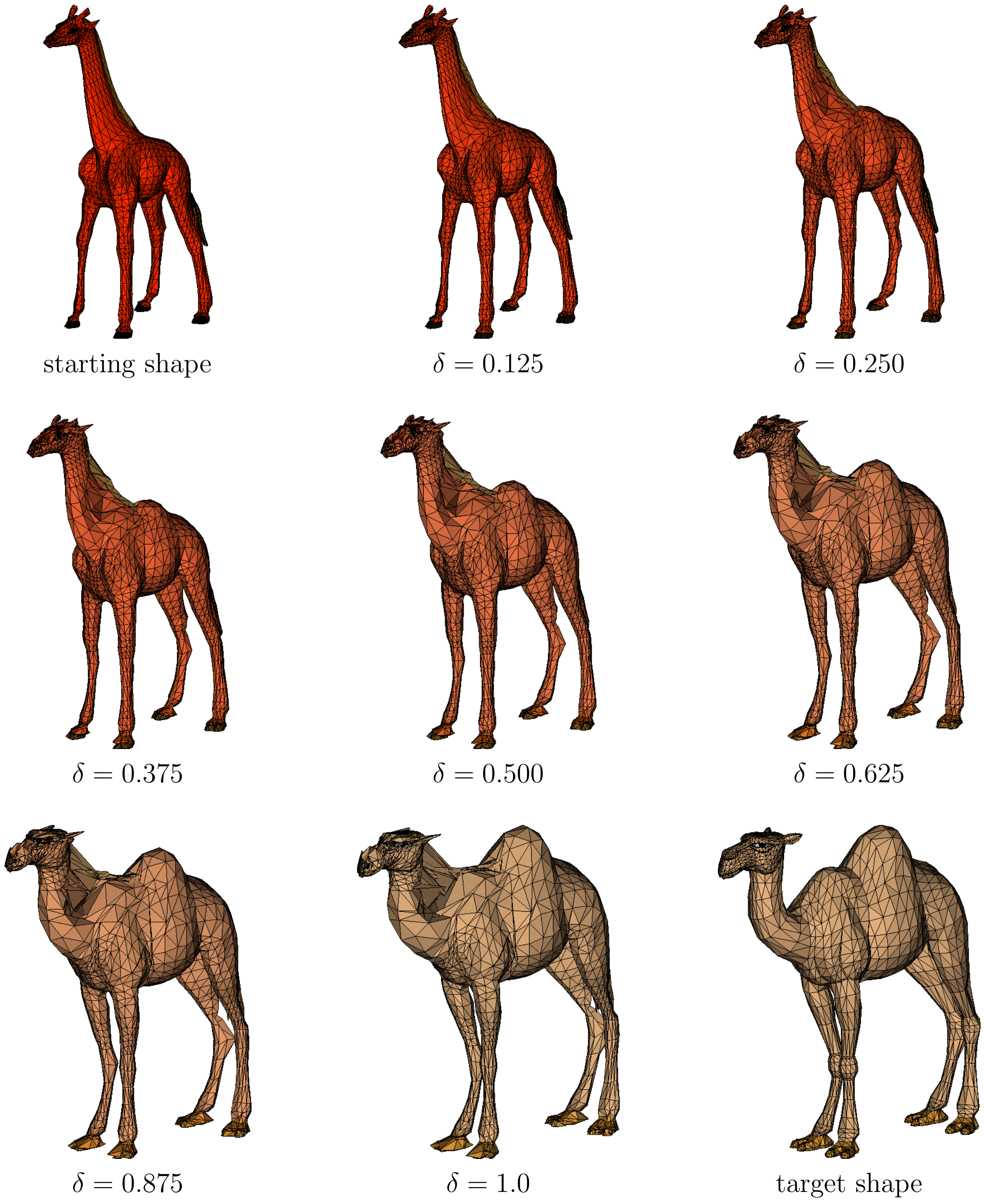


Figure 4-9: Morph showing the generated correspondence between the two animals shown in figure 4-5. The shape in the upper left was specified as A and the shape in the lower right was specified as B. The frames in between represent $\delta\vec{D}(A)$.

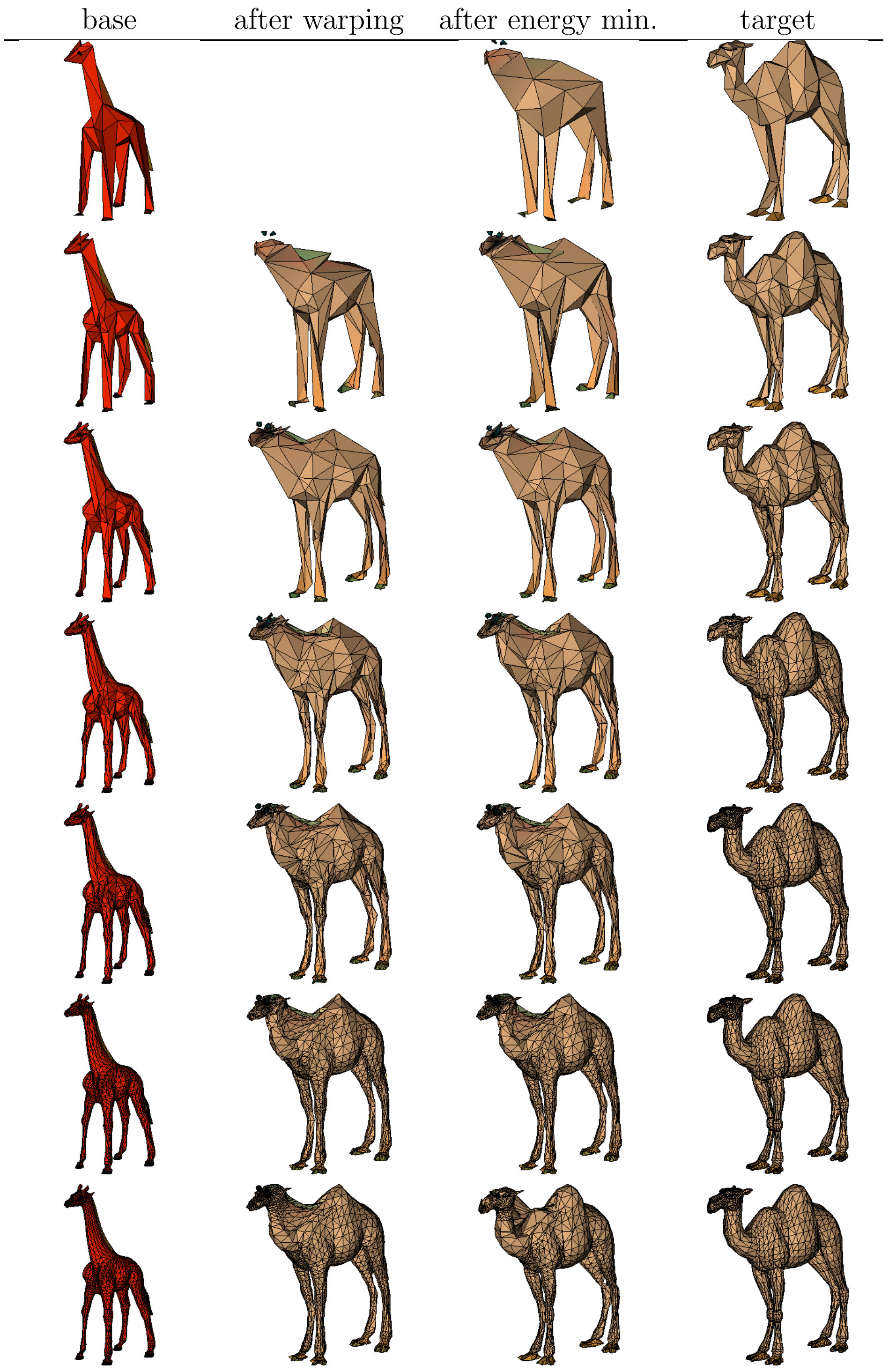


Figure 4-10: The output of the system at various points on each level of the algorithm. Please see the text for a description of the images.

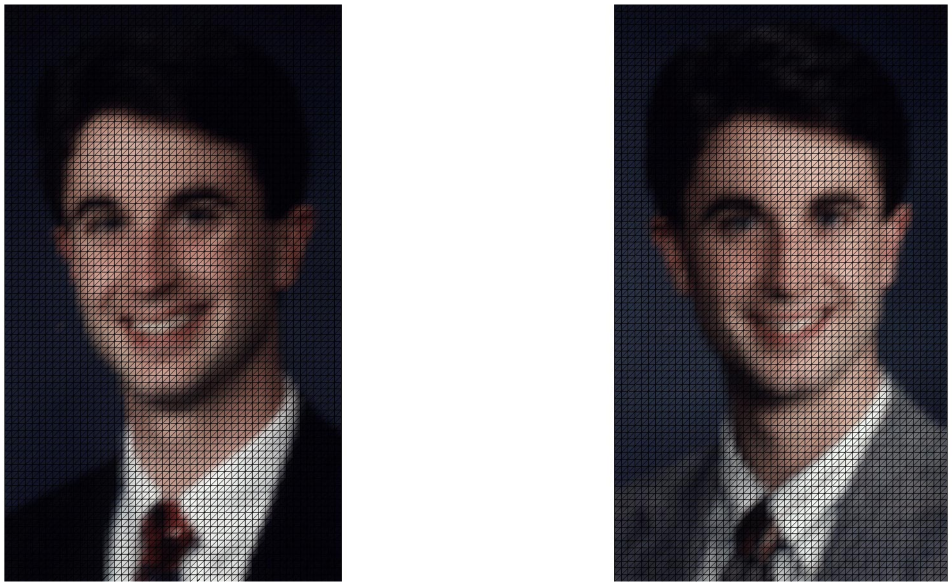


Figure 4-11: Two images converted into a triangle mesh lying in a single plane. Each mesh has 4250 vertices and 8232 faces.

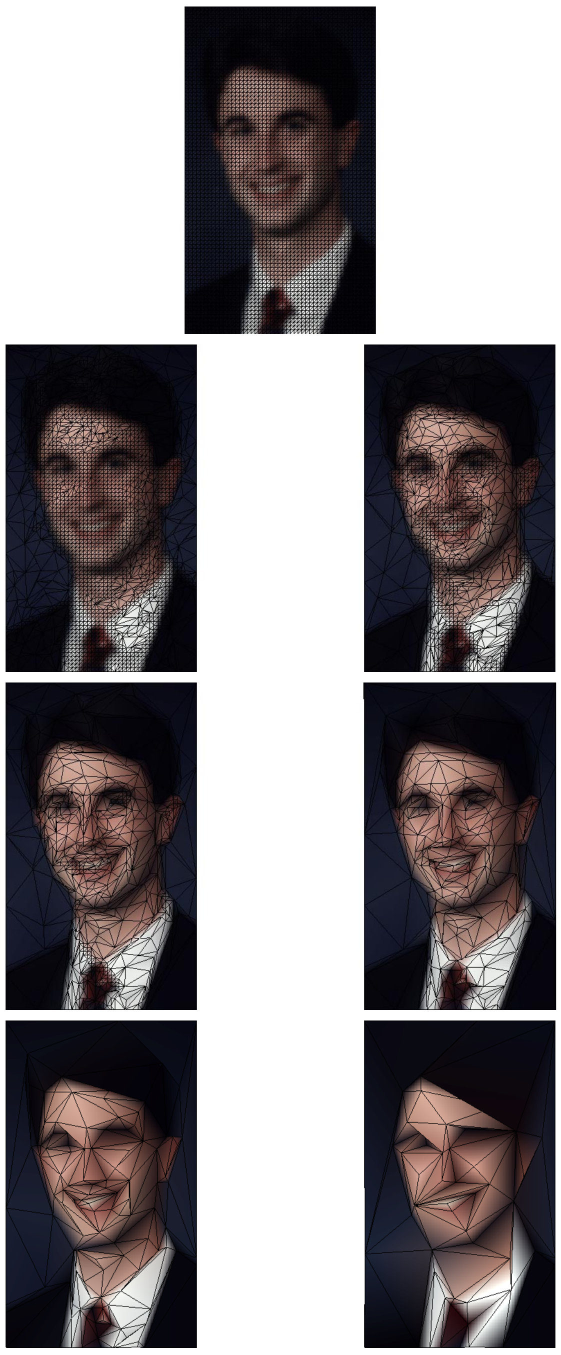


Figure 4-12: The original surface (at the top) and the reduced versions each with successively fewer polygons and vertices (each surface has half as many vertices as the previous)

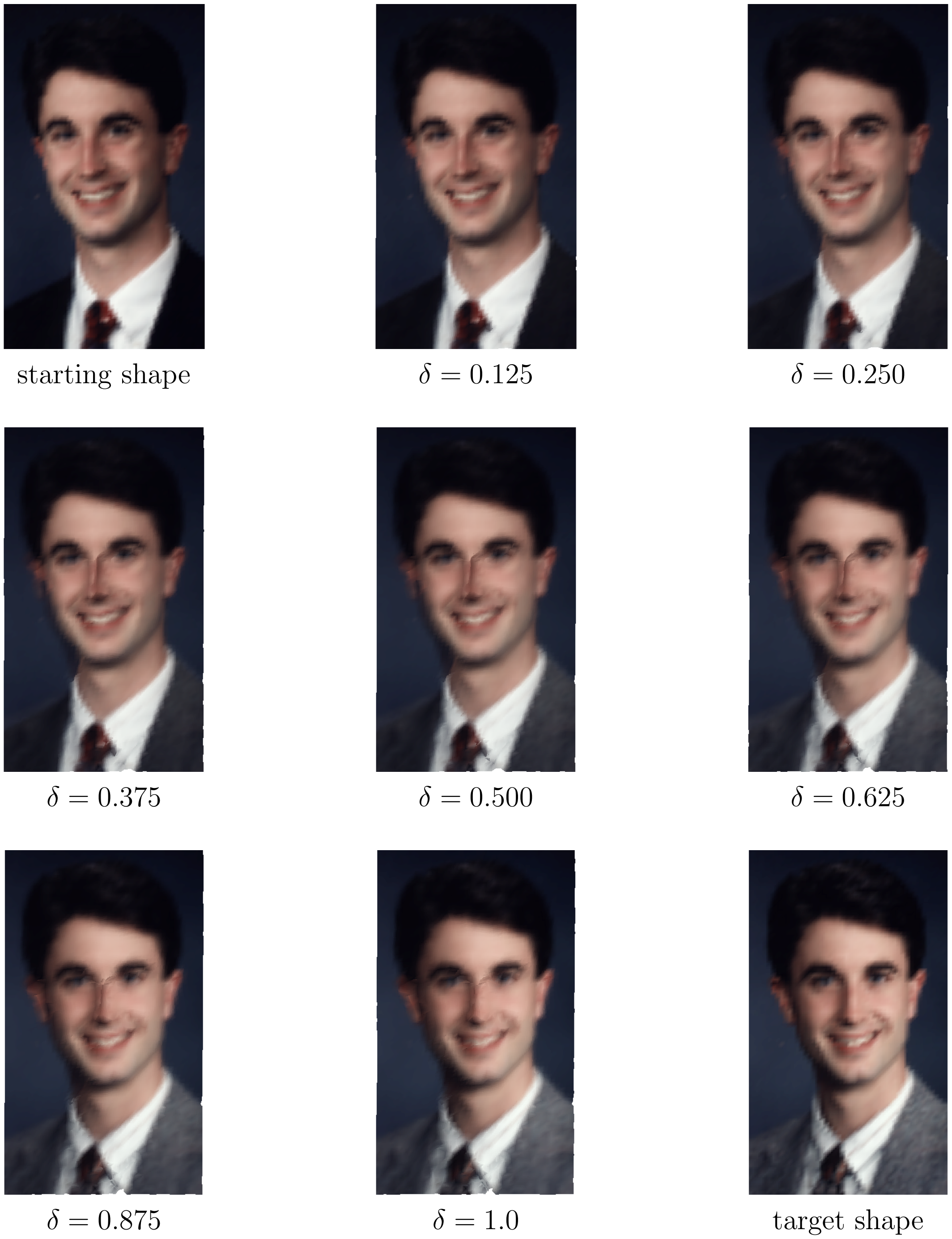


Figure 4-13: Morph showing the generated correspondence between the two images shown in figure 4-11. The shape in the upper left was specified as A and the shape in the lower right was specified as B. The frames in between represent $\delta\vec{D}(A)$. The outlines of the triangles (as shown in previous morph sequences) have been removed here for clarity.

Chapter 5

Conclusion

5.1 Related Work

Blanz and Vetter have also done work on surface correspondences [3]. In their work, the surface is first projected onto a surrounding shape which is then unrolled and optical flow techniques are used to find correspondence across these unrolled sheets. This has the advantage that the resulting objects (these sheets of color and distance values) are dense and optical flow techniques work well. Unfortunately, this also means that only a limited class of objects can be represented (namely it places limits on the curvature of the surfaces).

The energy minimization portion of the algorithm was inspired by the formulation of elastic nets [5]. It is also similar to snakes [10]. [11] provides an overview of such deformable models and their uses in matching. Our technique differs from other methods in its easy extension to higher dimensions, handling of colored surfaces, and multi-resolution approach.

The pyramid structure of the algorithm was originally inspired by Gaussian pyramid schemes used in optical flow such as the hierarchical method in [2]. Yet, this is the first time that a data-dependent method of reduction has been used for hierarchical correspondence generation. In hierarchical methods previously used in correspondence, the pyramid was created in the same way regardless of the data, whereas in this work, the shapes are reduced in a data-specific manner.

We feel that this is the most important contribution of this work. By using a compression technique which depends on the data (instead of Gaussian pyramids or other fixed methods), we perform implicit feature extraction. We do not work at a generic coarse level first, but rather with the most fundamental features (or points) on the shape where such features depend on the shapes given. This allows the algorithm to match gross features first. Otherwise, until the general outline of the two objects are set in correspondence, it is impossible to try to match the more detailed features. While a data-independent method of compression may pull out some of the important coarse features, a data-dependent method, like the mesh reduction used here, has a much better chance of finding coarse descriptions which have the important high-level features of the original objects.

5.2 Future Work

The algorithm developed in this thesis is a general automatic correspondence algorithm. As such, it does not produce exact correspondences for specific types of surfaces, but rather tries to produce acceptable results automatically for a wide range of surfaces.

There are a few additions to the algorithm which may help its performance. Since the energy function is locally quadratic, there may be faster methods for finding the minimum. It may also be helpful to consider matching curvature and not just position and orientation for the points on the surface. Adding the ability to refine the mesh of the base object further (i.e. to cut existing triangles) might give more flexibility and allow more general correspondence.

But, probably the most important improvement would be to modify the algorithm to be symmetric. That is, the correspondence between A and B should be the same as the correspondence between B and A. At the moment that requirement does not exist. However, adding such a qualification to the problem statement may aid the algorithm in finding the "correct" correspondence. Some points on surface A may not have an obvious match on surface B (there may be many points which look plausible or they may be no points which look reasonable); yet, by considering points on B, it may become obvious which point on B maps to the problematic point on A. Similar techniques have been used for optical flow and there is good reason to believe they would also work well here.

But, clearly the most important extension to this work will be to use the correspondences generated by this algorithm in an application. That can be the only true method of evaluation, for ultimately its usefulness depends on how well the correspondence allow an application to perform. With any automatic technique there will be flaws in the correspondences but the real question is whether those flaws are fatal for the desired application and that can only be answered through implementation and tests.

Appendix A

Mathematical Derivations

A.1 Derivative of the distance from a point to a triangle

Consider figure A-1. We have the triangle $\triangle abc$ and the point p where p is not in the same plane as $\triangle abc$. p' is the point on the triangle closest to p. For the moment we will assume that p' is the same as the projection of p into the plane of $\triangle abc$ (or phrased differently, the projection of p onto the plane of $\triangle abc$ lies inside of $\triangle abc$). We wish to find $\frac{d(D^2)}{d\mathbf{a}}$ (where $\mathbf{a}$ is the vector of the position of a by an abuse of notation and D is the distance from p to p'). That is, we wish to find the derivative of the square of the distance from p to $\triangle abc$ with respect to the point a.

First we construct the line segment $\overline{aq}$ which is a segment running from a through p' and terminating on the line segment $\overline{bc}$. We then note that the derivative of D^2 with respect to a must be parallel to the vector $\mathbf{p} - \mathbf{p}'$ since movement of a in the plane of $\triangle abc$ will not produce a change in D (it can similarly be shown that in higher dimensions, of all of the directions normal to the plane of $\triangle abc$, only in the direction of $\mathbf{p} - \mathbf{p}'$ will D change instantaneously).

Now that we must only determine the magnitude of the derivative, we can limit ourselves to look at the slice of figure A-1 containing the points p, q, and a. This is shown in figure A-2. We set a coordinate system with q at the origin and the line segment $\overline{aq}$ along the x axis. If we let L be the distance from q to p' and K be the distance from q to a, then p has coordinates (L, D) in this coordinate system.

We now consider allowing the y coordinate of the point a to change so that we can take the derivative of the square of the distance with respect to this new variable. To do this, we let a' be a "new" version of a restrained to have the same x coordinate. Then, p'' becomes the point closest to p along the line from q to a' and D' becomes the distance from p to p''. We now want to find the derivative of D'^2 with respect to y (the y-axis coordinate of a').

Noting that the two angles labeled S in figure A-2 are the same, we can quickly find that

$$D' = (D - L \tan S) \cos S = \frac{DK - Ly}{\sqrt{K^2 + y^2}} \tag{A.1}$$

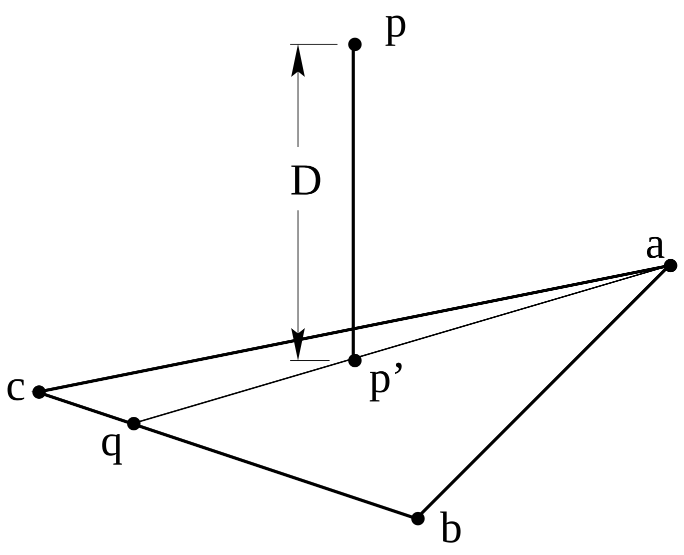


Figure A-1: Distance from a point to a triangle

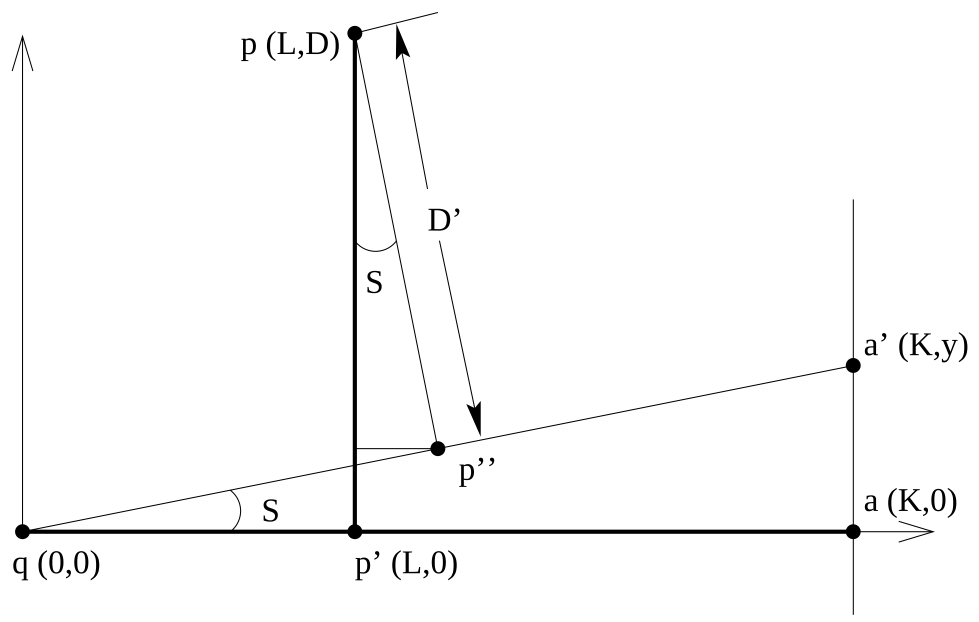


Figure A-2: Cross section of figure A-1 along the plane containing p, q, and a.

Thus the derivation of $\frac{d(D'^2)}{dy}$ goes as follows:

$$\begin{aligned}
\frac{d(D'^2)}{dy} &= \frac{d}{dy}\frac{(DK - Ly)^2}{K^2 + y^2} \\
&= \frac{-2(DK - Ly)L(K^2 + y^2) - 2(DK - Ly)^2 y}{(K^2 + y^2)^2} \\
\left.\frac{d(D'^2)}{dy}\right|_{y=0} &= \frac{-2(DK)LK^2}{K^4} \\
&= -2D\frac{L}{K}
\end{aligned}$$

The derivative of D^2 with respect to the point a is a vector in the direction from point p to point p' with distance proportional to both D and the ratio of L to K. This result makes good sense: the closer p' is to q the smaller the derivative since moving a will change D less since a is much farther from the point of rotation (q) than p' is.

If we now consider the more general case where the projection of p onto the plane of $\triangle abc$ does not lie in $\triangle abc$, p' will lie on the edge of $\triangle abc$. Fortunately, the results above extend to this case as well. If p' lies on on $\overline{bc}$, $\frac{L}{K}$ is 0, which is correct since small changes to a will have no effect on p'. If p' lies on $\overline{ac}$ or $\overline{ac}$, then, provided we are careful to take the direction of the derivative to be along the vector $p - p'$ and not perpendicular to $\triangle abc$, we can form the same cross section as shown in figure A-2 and get the same result. The three special cases of p' coinciding with a vertex of $\triangle abc$ are limiting cases of the above result. If p' and either b or c coincide, then clearly $\frac{L}{K}$ is zero as desired. If p' and a coincide, then $\frac{L}{K}$ is 1 despite the fact that q is not unique.

A.2 Warping a single triangle

The fraction $\frac{L}{K}$ played an important role in the derivative developed in appendix A.1. The same ratio will be key in our formulation of correspondence between triangles. Consider the two triangles ($\triangle abc$ and $\triangle a'b'c'$) in figure A-3. Given that we know the correspondences between the vertexes of the two triangles (i.e. a corresponds to a', b to b' and c to c') and given an arbitrary point inside of $\triangle abc$, p, we would like to find the corresponding point, p', in $\triangle a'b'c'$.

The two triangles need not be similar so finding p' is not trivial. We would like p' to have the same "relationship" to a', b', and c' that p has to a, b, and c. One example would be that if p lies on $\overline{ab}$, then p' should lie on $\overline{a'b'}$, preferably such that the ratio of the lengths of $\overline{ap}$ and $\overline{pb}$ is the same as the ratio of the lengths of $\overline{a'p'}$ and $\overline{p'b'}$. Similarly, if p rests in the middle of $\triangle abc$, p' should also sit in the middle of $\triangle a'b'c'$. To make this more concrete, we shall say that the ratios of the areas of the three subtriangles $\triangle abp$, $\triangle bcp$, and $\triangle acp$ to the total area of the triangle $\triangle abc$ should be the same as the corresponding ratios in the "prime" triangle (namely the ratios of $\triangle a'b'p'$, $\triangle b'c'p'$, and $\triangle a'c'p'$ to $\triangle a'b'c'$).

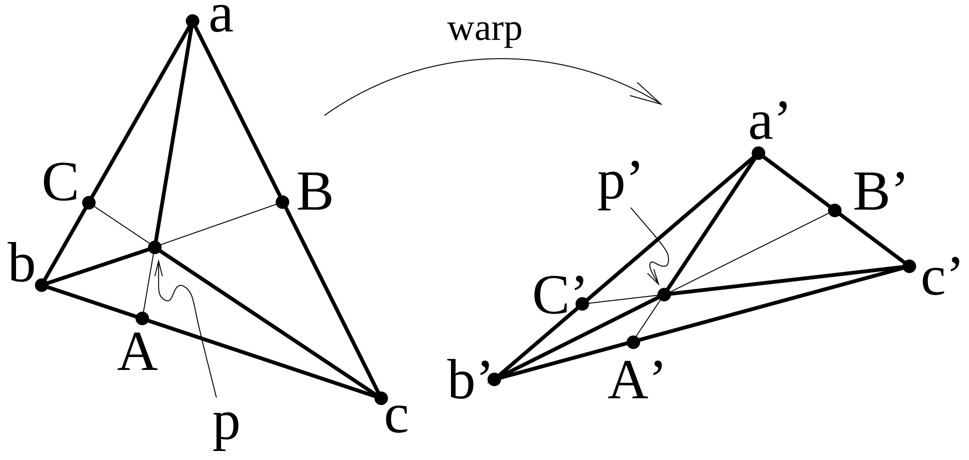


Figure A-3: Correspondence between two triangles

This constraint on p' is interestingly related to the $\frac{L}{K}$ in appendix A.1. If one constructs the points A, B, and C as shown in figure A-3, then for example, the ratio of the area of $\triangle abp$ to the area of $\triangle abc$ is the same as the ratio of the length of $\overline{pB}$ to the length of $\overline{bB}$. This ratio is the same length ratio as $\frac{L}{K}$ in appendix A.1. $\frac{L}{K}$ determined the degree to which a vertex of the triangle effected the distance from the triangle to another point. Here, this ratio will determine how much moving a vertex of the triangle will pull or push another point inside of the triangle.

We will define $\alpha(p)$ to be the ratio of the length of $\overline{aA}$ to the length of $\overline{pA}$, $\beta(p)$ to be the ratio of the length of $\overline{bB}$ to the length of $\overline{pB}$, and $\gamma(p)$ to be the ratio of the length of $\overline{cC}$ to the length of $\overline{pC}$. α', β', and γ' are similarly defined for p' and $\triangle a'b'c'$. We can then state our constraint on the position of p' as the following system of equations.

$$\begin{aligned} \alpha(p) &= \alpha'(p') \\ \beta(p) &= \beta'(p') \\ \gamma(p) &= \gamma'(p') \end{aligned} \tag{A.2}$$

Since α, β, and γ are invariant to warps of the triangle and uniquely define a point on the triangle, we may view these three variables as a warp invariant coordinate system on the triangle. This may seem like three constraints for only two unknowns (since p and p' each lie in a plane). However, returning to our initial formulation of α, β, and γ as ratios of areas, we can easily see that $\alpha(p) + \beta(p) + \gamma(p) = 1$ for all p inside $\triangle abc$ and thus one of the three coordinates is redundant. Thus, to warp a point from one triangle to another, we first convert the point from the Cartesian coordinates to the vertex relative coordinates (α, β, γ) and then perform an inverse mapping back to Cartesian coordinates, but using the new “prime” vertexes instead of the original vertexes.

The inverse mapping from (α, β, γ) back to Cartesian coordinates can be accomplished by calculating the intersection of two lines. Note that, given the positions of a', b', and c', knowing α' fixes p' to be on a line parallel to $\overline{b'c'}$ between a' and $\overline{b'c'}$. β defines a similar line parallel to $\overline{a'c'}$ and γ defines a third line parallel to $\overline{a'b'}$. Provided that these three coefficients sum to 1, the three lines will meet at the single point p'. In practice, it is best to take each pair of lines and find their intersection and then average the three resulting points to get p'.

A.3 A new operator: the scalar product of two planes

We would like to be able to define a similarity measure for the orientation of two linear subspaces. First consider two lines in space that pass through the origin. These lines can be parameterized by their tangent normal vectors, $\mathbf{v}$ and $\mathbf{w}$. The angle between the two lines is $\arccos(\mathbf{v}^T\mathbf{w})$ and $(\mathbf{v}^T\mathbf{w})^2$ provides a reasonable measure for the similarity in orientation between two one-dimensional subspaces (we wish to square the inner product since our subspaces are not oriented surfaces and thus angles, Θ, greater than $\pi/2$ should actually be measured as $\pi - \Theta$).

Now consider the case of two planes of the same dimension passing through the origin. We will describe each subspace as the column space of a matrix (V and W). If V and W have dimensions d by $d-1$ (e.g. a two dimensional plane in three dimensions), then the common definition of the angle between the two planes is $\arccos(\mathbf{v'}^T\mathbf{w'})$ where $\mathbf{v'}$ is the normal vector perpendicular to the column space of V and $\mathbf{w'}$ is similarly defined relative to W. From this we may conclude that $(\mathbf{v'}^T\mathbf{w'})^2$ is a reasonable rotational similarity measure for subspaces of dimension one less than their embedding space.

However, neither of these work for cases in which the dimensionality of the subspace is neither 1 nor $d-1$. There are still definitions for the angle between two planes, but these unfortunately do not capture the desired properties. For instance, consider the planes defined as the column spaces of X, Y, and Z:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \; Y = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \; Z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.3}$$

The angle between the planes defined by X and Y is $\pi/2$ and the angle between the planes defined by X and Z is also $\pi/2$. However intuitively, it seems that the rotational distance between the planes of X and Z should be greater than the rotational distance between the planes of X and Y: the column spaces of X and Y intersect in a line whereas the column spaces of X and Z intersect only at the origin.

Thus we will construct a new measure for the rotational similarity between two subspaces. We will, as before, define the two subspaces as the column spaces of the matrices V and W. We will further restrict V and W to have same dimensions (d by n) and to each have orthonormal columns. The subspaces (or column spaces of V and W) will be denoted by $\mathcal{V}$ and $\mathcal{W}$. Furthermore, we will denote the left nullspace of V and W by $\mathcal{V}'$ and $\mathcal{W}'$ respectively. We will also let V' and W' be two matrices of orthonormal columns whose column spaces are $\mathcal{V}'$ and $\mathcal{W}'$ respectively. All of these definitions are analogous to the definitions in the first few paragraphs of this appendix. $\mathcal{V}$ is the column space of V. The column space of V' is the space of all vectors perpendicular to $\mathcal{V}$ and we have denoted this space as $\mathcal{V}'$. Thus, if $\mathcal{V}$ is a plane, V has column vectors which lie in the plane, and V' has column vectors which are normal to the plane.

Let us denote this new measure with the symbol $\diamond$, a binary operator on two matrices both of dimension d by n (i.e. $V \diamond W$). There are a number of properties it should observe:

1. if $n = 1$, $V \diamond W = (V^T W)^2$
2. if $n = d - 1$, $V \diamond W = (V'^T W')^2$
3. $V \diamond W = W \diamond V$
4. $V \diamond W = V' \diamond W'$
5. Given an orthogonal matrix R, $(VR) \diamond W = V \diamond W$.
6. if $\mathcal{V} = \mathcal{W}$, $V \diamond W = 1$

The first two properties simply state that it should reduce to the two examples listed in the beginning of this section. The third property states that the operator should be commutative. The fourth property says that the operator applied to the left nullspaces of two matrices should be the same as it applied to the column spaces of those matrices. This is an extension of the observation that for planes of dimension one less than the dimension of the space, the angle can be computed using the inner product of the normals to the planes. The left nullspace is, in general, all of the directions which are perpendicular to the column space. Thus, properties one and four together imply property two. The fifth property maintains that the measure should be invariant to the basis used to describe the subspace and the final property says that the value of 1 will indicate parallel planes. There is one final property that is not listed and that is we want it to handle cases like the case of X, Y, and Z given in equation A.3 properly (e.g. it should assign a smaller value to $X \diamond Z$ than to $X \diamond Y$).

In order to construct this similarity measure, let us consider the matrix P which will be equal to $V^T W$. This matrix has all possible inner products of the columns of V with the columns of W. $\|P\|_F^2$ is almost the operator we desire.[1] We can view

[1] $\|\cdot\|_F$ represents the Frobenius Norm, or the square root of the sum of the squares of all of the elements of a matrix

$\|P\|_F^2$ as the sum of the squares of the lengths of the projections of the bases of $\mathcal{V}$ onto $\mathcal{W}$ (or vice-versa). Consider that each row of P contains the inner products of one basis vector of $\mathcal{V}$ with each of the basis vectors of $\mathcal{W}$. Thus, the sum of the squares of a row is the square of the length of the a basis vector of $\mathcal{V}$ projected onto $\mathcal{W}$. Clearly, this is invariant to choice of the basis vectors for $\mathcal{W}$ (provided they are orthonormal). Since this operator is invariant to the particular choice of W to represent $\mathcal{W}$ and is obviously commutative, it must also be invariant to the choice of V to represent $\mathcal{V}$.

Why $\|P\|_F^2$? It satisfies properties one, three, and five and property six is simple to fix by scaling or translation. It also produces different values for $X \diamond Y$ and $X \diamond Z$. There remains only one problem: $\|V^TW\|_F^2 \neq \|V'^TW'\|_F^2$. Yet, this is the precise reason that we chose the square of the Frobenius Norm. Consider the two orthogonal matrices $\hat{V}$ and $\hat{W}$ created by concatenating V and V' and W and W' respectively. If we now consider $\hat{V}^T\hat{W}$ pictorially we can view the importance of the Frobenius Norm:

$$\hat{V}^T\hat{W} = \begin{bmatrix} V^T \\ \\ V'^T \end{bmatrix} \begin{bmatrix} W & W' \end{bmatrix} = \begin{bmatrix} V^TW & V^TW' \\ \\ V'^TW & V'^TW'^T \end{bmatrix} = \hat{P} \qquad \text{(A.4)}$$

Note that $\hat{P}$ is orthogonal since both $\hat{V}$ and $\hat{W}$ are. This means that the sum of squares of the elements of each column or row of $\hat{P}$ is 1. Thus, the square of the Frobenius Norm of the first n rows of $\hat{P}$ is equal to n. Furthermore, it can be broken up into the sum of the squares of the Frobenius Norms of the upper left and upper right quadrants of $\hat{P}$. Similarly, the square of the Frobenius Norm of the right $d-n$ rows of $\hat{P}$ is equal to $d-n$ and can be broken into the sum of the squares of the Frobenius Norms of the upper right and lower right quadrants. Thus we have that

$$\begin{array}{rcl} \|V^TW\|_F^2 + \|V^TW'\|_F^2 & = & n \\ \|V^TW'\|_F^2 + \|V'^TW'\|_F^2 & = & d-n \\ & \Longrightarrow & \\ \|V^TW\|_F^2 - n & = & \|V'^TW'\|_F^2 - (d-n) \end{array}$$

which is exactly what we want. The constant n on the left side is the size of the column spaces of V and W and the constant $(d-n)$ on the right side is the size of the column spaces of V' and W'. We need only translate the operator so that property six is held and we have the following definition of the operator $\diamond$:

$$A_{d\times n} \diamond B_{d\times n} \triangleq \|A^TB\|_F^2 - d + 1 \qquad \text{(A.5)}$$

Since $\|A\|_F^2 = tr(A^TA)$, we can rewrite this definition as

$$A \diamond B \triangleq tr(B^TAA^TB - I) + 1 \qquad \text{(A.6)}$$

and remove an explicit mention of the dimensions of A or B.

What are the properties of this operator? For two subspaces of dimension d of a space of dimension n, the range of the operator is $[1 - min(d, n - d), 1]$ which means that for hyperplanes with one fewer dimensions than their embedded space or for one dimensional lines the operator measures the square of the cosine of the angle between either the normals or the tangents respectively. If the two hyperplanes are orthogonal and intersect along a $d - 1$ dimensional subspace, then the operator evaluates to 0. If they are orthogonal, but intersect in a smaller dimensional subspace, the operator evaluates to a negative number indicating "hyper-orthogonality." It should finally be noted that this operator is robust: small changes in the matrices do not produce large changes in the result.

Bibliography

[1] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 236–242, Champaign, IL, 1992.

[2] J. R. Bergen and R. Hingorani. Hierarchical motion-based frame rate conversion. Technical report, David Sarnoff Research Center, Princeton, NJ, April 1990.

[3] Volker Blanz and Thomas Vetter. Internal publication and personal communication.

[4] D. Louis Collins, Peter Neelin, Terrence M. Peters, and Alan C. Evans. Automatic 3D intersubject registration of MR volumetric data in standardized talairach space. *Journal of Computer Assisted Tomography*, 18(2):192–205, 1994.

[5] Richard Durbin and David Willshaw. An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691, April 1987.

[6] James C. Gee, Lionel Le Briquer, Christian Barillot, David R. Haynor, and Ruzena K. Bajcsy. Bayesian approach to the brain image matching problem. In *Proc. SPIE*, volume 2434, pages 145–156, May 1995.

[7] Hugues Hoppe. Progressive meshes. *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 99–108, 1996.

[8] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 19–26, 1993.

[9] Michael J. Jones and Tomaso Poggio. Model-based matching by linear combinations of prototypes. AI Memo 1583, AI Laboratory, MIT, 1996.

[10] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[11] Tim McInerney and Demetri Terzopoluos. Deformable models in medical image analysis: A survey. *Medical Image Analysis*, 1(2):91–108, 1996.

[12] Tomaso Poggio and Thomas Vetter. Recognition and structure from one 2d model view: Observations on prototypes, object classes and symmetries. AI Memo 1347, AI Laboratory, MIT, 1992.

[13] Thomas Vetter and Tomaso Poggio. Linear object classes and image synthesis from a single example image. AI Memo 1531, AI Laboratory, MIT, 1995.