Feature-Based Light Field Morphing

Zhunping Zhang Lifeng Wang Baining Guo Heung-Yeung Shum

Microsoft Research Asia*

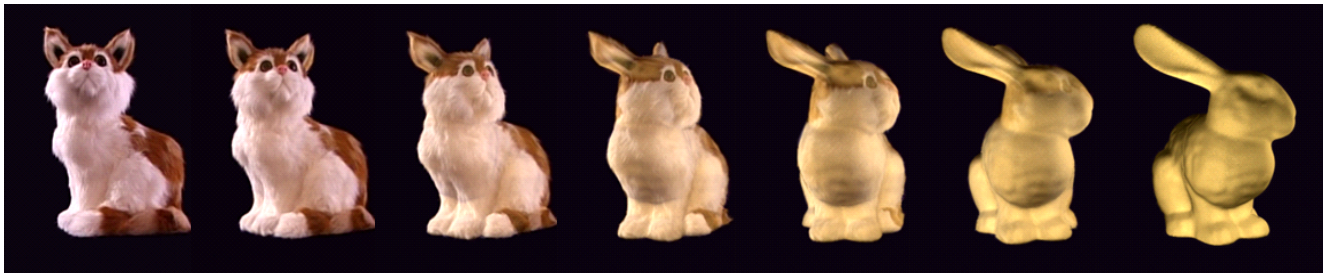

Figure 1: Light field morphing: A 3D morphing sequence from a furry toy cat (real object) to the Stanford bunny (synthetic object).

Abstract

We present a feature-based technique for morphing 3D objects represented by light fields. Our technique enables morphing of image-based objects whose geometry and surface properties are too difficult to model with traditional vision and graphics techniques. Light field morphing is not based on 3D reconstruction; instead it relies on *ray correspondence*, i.e., the correspondence between rays of the source and target light fields. We address two main issues in light field morphing: feature specification and visibility changes. For feature specification, we develop an intuitive and easy-to-use user interface (UI). The key to this UI is *feature polygons*, which are intuitively specified as 3D polygons and are used as a control mechanism for ray correspondence in the abstract 4D ray space. For handling visibility changes due to object shape changes, we introduce *ray-space warping*. Ray-space warping can fill arbitrarily large holes caused by object shape changes; these holes are usually too large to be properly handled by traditional image warping. Our method can deal with non-Lambertian surfaces, including specular surfaces (with dense light fields). We demonstrate that light field morphing is an effective and easy-to-use technqiue that can generate convincing 3D morphing effects.

Keywords: 3D morphing, light field, ray correspondence, feature polygons, global visibility map, ray-space warping

1 Introduction

Metamorphosis, or morphing, is a popular technique for visual effects. When used effectively, morphing can give a compelling illusion that an object is smoothly transforming into another. Following the success of image morphing [1, 26], graphics researchers have developed a variety of techniques for morphing 3D objects [15, 10]. These techniques are designed for geometry-based objects, i.e., objects whose geometry and surface properties are known, either explicitly as for boundary-based techniques (e.g., [14, 6, 16]) or implicitly as for volume-based techniques (e.g., [13, 17, 5]).

In this paper, we describe a feature-based morphing technique for 3D objects represented by light fields/lumigraphs [19, 11]. Unlike traditional graphics rendering, light field rendering generates novel views directly from images; no knowledge about object geometry or surface properties is assumed [19]. Light field morphing thus enables morphing between image-based objects, whose geometry and surface properties, including surface reflectance, hypertexture, and subsurface scattering [7], may be unknown or difficult to model with traditional graphics techniques.

The light field morphing problem can be stated as follows: Given the source and target light fields L_0 and L_1 representing objects O_0 and O_1, construct a set of intermediate light fields $\{L_\alpha \mid 0 < \alpha < 1\}$ that smoothly transforms L_0 into L_1, with each L_α representing a plausible object O_α having the essential features of O_0 and O_1. We call the intermediate light field L_α a light field morph, or simply a *morph*.

A naive approach to light field morphing is to apply image morphing to individual images in the source and target light fields and assemble the light field morphs from the intermediate images of image morphing. Unfortunately, this approach will fail for a fundamental reason: light field morphing is a 3D morphing and image morphing is not. This difference manifests itself when a hidden part of the morphing object becomes visible because of object shape change, as image morphing will produce "ghosting" that betrays a compelling 3D morphing.

The plenoptic editing proposed by Seitz and Kutulakos [22] represents another approach to image-based 3D morphing. They first recover a 3D voxel model from the image data and then apply traditional 3D warping to the recovered model. The visibility issues can be resolved with the recovered geometry, but there are problems, including the Lambertian surface assumption needed for voxel carving [22] and the difficulties with recovering detailed geometry. Most of the problems are related to the fundamental difficulties of recovering surface geometry from images [9].

*3F, Beijing Sigma Center, No. 49, Zhichun Road, Haidian District, Beijing 100080, P R China. email:bainguo@microsoft.com

Copyright © 2002 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1-212-869-0481 or e-mail permissions@acm.org.
© 2002 ACM 1-58113-521-1/02/0007 $5.00

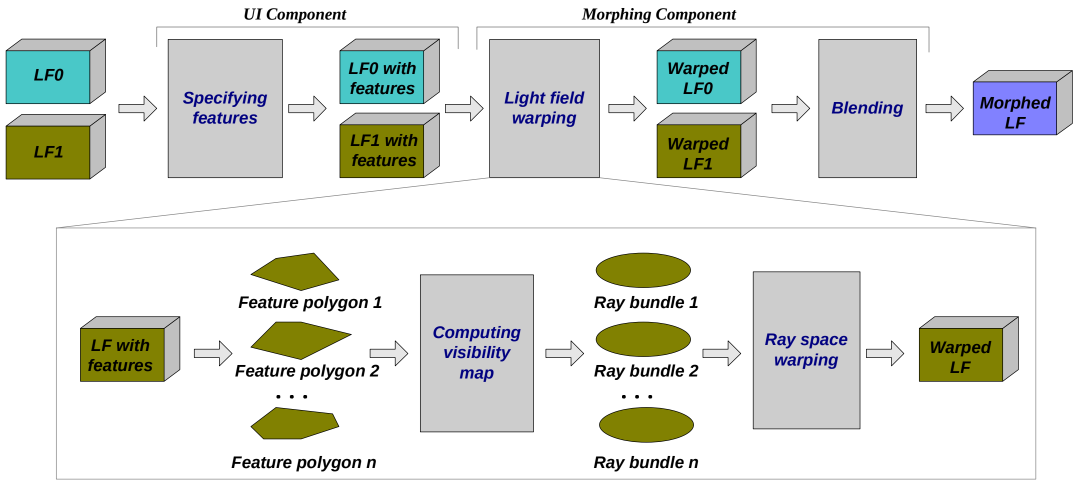

Figure 2: Overview of light field morphing. The overall pipeline is illustrated in the upper part, whereas the warping of a light field is detailed in the lower part.

Light field morphing is an image-based 3D morphing technique that is not based on 3D surface reconstruction. The basis of light field morphing is *ray correspondence*, i.e., the correspondence between rays of the source and target light fields [19]. The role of ray correspondence in light field morphing is the similar to that of vertex correspondence in geometry-based 3D morphing [15, 16]. Like vertex correspondence (e.g., see [6, 12]), ray correspondence is controlled by user-specified feature elements.

A key issue in light field morphing is thus the construction of a user interface (UI) for specifying feature elements. Since there is no intrinsic solution to a morphing problem, user interaction is essential to the success of any morphing system [17, 26, 15]. For light field morphing, the main challenge is the design of intuitive feature elements for an abstract 4D ray space [19]. To address this challenge, we introduce *feature polygons* as the central feature elements for light field morphing. As 3D polygons, feature polygons are intuitive to specify. More importantly, feature polygons partition a light field into groups of rays. The rays associated with a feature polygon P constitute a *ray bundle*, and the ray correspondence of this ray bundle is controlled by the control primitives of the feature polygon P. Note that feature polygons do not make a rough geometry of the underlying object; they are needed only at places where visibility changes (due to object shape change).

Another key issue in light field morphing, and more generally in image-based 3D morphing, is visibility change. Two types of visibility change exist. The first is due to viewpoint changes. In light field morphing, this type of visibility change is automatically taken care of by the input light fields. The second type of visibility change is that caused by object shape changes, and this is an issue we must handle. For a given view, a hole is created when a hidden surface patch in the source light field L_0 becomes visible in the target light field L_1 due to object shape change. This type of hole may be arbitrarily large and thus cannot be dealt with properly by traditional image warping methods (e.g., [4, 21]). We solve this problem using a novel technique called *ray-space warping*, which is inspired by Beier and Neely's image warping [1]. With ray-space warping we can fill a hole by approximating an occluded ray with the "nearest visible ray". Not surprisingly, ray-space warping requires visibility processing and the key to visibility processing is the *global visibility map*, which associates each light field ray with a feature polygon.

Ray-space warping produces accurate results under the popular Lambertian surface assumption [4, 21]. For non-Lambertian surfaces, ray-space warping tries to minimize the errors by using the "nearest visible rays". We demonstrate that, unlike plenoptic editing [22], our method can effectively handle non-Lambertian surfaces, including specular surfaces.

Light field morphing is easy to use and flexible. Feature specification usually takes about 20 $\sim$ 30 minutes and sparse light fields can be used as input to save storage and computation. When the input light fields are very sparse (e.g. 2 $\sim$ 3 images per light field), we call light field morphing key-frame morphing to emphasize its similarity with image morphing. Key-frame morphing may be regarded as a generalization of view morphing [21] because key-frame morphing allows the user to add more input images as needed to eliminate the holes caused by visibility changes. Note that although view morphing can generate morphing sequences that appear strikingly 3D, it is not a general scheme for image-based 3D morphing because the viewpoint is restricted to move along a prescribed line.

We will demonstrate results for a few applications of light field morphing, including generating 3D morphs for interactive viewing, creating animation sequences of a 3D morphing observed by a camera moving along an arbitrary path in 3D, key-frame morphing, and transferring textures from one 3D object to another. For computing an animation sequence of a 3D morphing, we present an efficient method that generates the sequence without fully evaluating all the morphs. The techniques we present can be used as visualization tools for illustration/education purposes [2], in the entertainment industry, and for warping/sculpting image-based objects [18, 22].

The rest of the paper is organized as follows. In Section 2, we give an overview of our system. Section 3 describes the specification of feature elements, in particular feature polygons, and visibility processing. Section 4 presents the warping algorithms for warping a light field and for generating 3D animation sequences. Experimental results are reported in Section 5, followed by conclusions in Section 6.

2 Overview

As shown in Fig. 2, our system has two main components. The first is a UI for specifying feature element pairs through side-by-side interactive displays of the source and target light fields. We use three types of feature elements: feature lines, feature polygons, and background edges. The second component is a morphing unit that

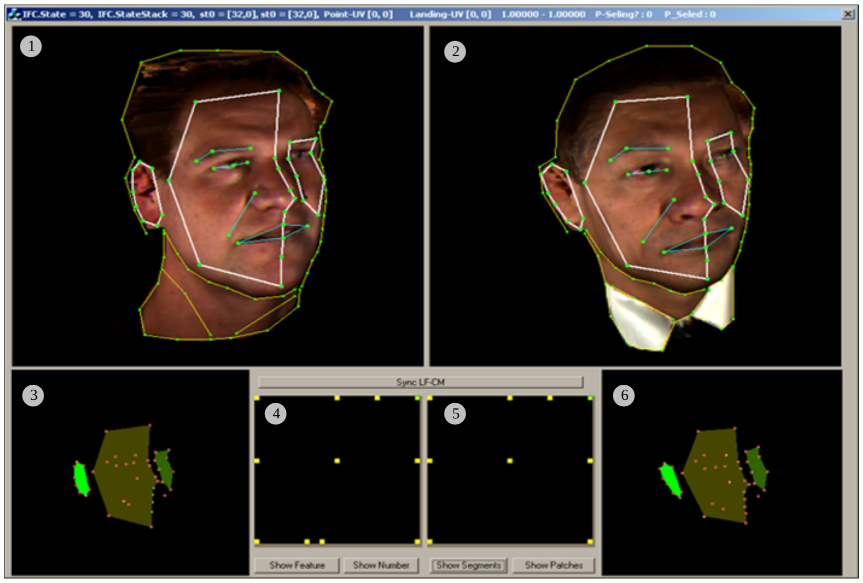

Figure 3: The user interface for feature specification. On the top, windows (1) and (2) are interactive renderings of the source and target light fields. Three pairs of feature polygons are drawn using wireframe rendering (white lines) on top of the source and target objects. The background edges are drawn as yellow polylines. On the bottom, windows (3) and (6) are interactive renderings of the global visibility maps, showing the visibility of the feature polygons using color-coded polygons. Windows (4) and (5) display the (s, t)-planes of the two light fields, with each yellow dot representing a key view used for specifying background edges.

automatically computes the morph L_α for a given α through the following steps. First, the feature elements of L_α are obtained by linearly interpolating those of L_0 and L_1. Second, L_0 and L_1 are warped to $\hat{L}_0$ and $\hat{L}_1$ respectively for feature alignment. Finally, L_α is obtained by linearly interpolating the warped light fields $\hat{L}_0$ and $\hat{L}_1$. Of these steps, both the first and last steps are simple; the warping step is the main part of the second component.

The two most important operations in light field morphing are feature specification and visibility processing. The key to feature specification is the feature polygons, which are 3D (non-planar) polygons approximating surface patches of 3D objects. The key to visibility processing is the global visibility map, which associates each ray of a light field L with a feature polygon of L.

The critical roles of feature polygons and global visibility maps in the warping of a light field L are illustrated in the lower part of Fig. 2. From the user-specified feature polygons of L, we can compute the global visibility map of L. The global visibility map partitions L into ray bundles such that each feature polygon P is associated with a ray bundle $R(P)$. Light field warping is then performed by warping one ray bundle at a time using ray-space warping, with the ray correspondence of a ray bundle $R(P)$ determined by P's control primitives.

Feature polygons are only needed where visibility changes. Rays not in any ray bundle are called background rays, which can be easily treated by image warping because there is no visibility change involved.

Notation: Following the convention of [11], we call the image plane the (u, v)-plane and the camera plane the (s, t)-plane. For a given light field L, we can think of L either as a collection of images $\{L_{(s,t)}\}$ or as a set of rays $\{L(u, v, s, t)\}$. An image $L_{(s_0,t_0)}$ is also called a view of the light field L. In $L_{(s_0,t_0)}$, the pixel at position (u_0, v_0) is denoted as $L_{(s_0,t_0)}(u_0, v_0)$, which is equivalent to ray $L(u_0, v_0, s_0, t_0)$.

3 Features and Visibility

In feature-based morphing [10], the corresponding features of the source and target objects are identified by a pair of feature elements. In this section, we show how to specify such feature element pairs when the source and target objects are described by light fields. We also describe visibility processing using feature polygons.

3.1 Feature Specification

The user specifies feature element pairs using the UI shown in Fig. 3. We use three types of feature elements: feature lines, feature polygons, and background edges.

Feature Lines: A feature line is a 3D line segment connecting two points called its vertices, which are also called *feature points*. The purpose of a feature line is to approximate a curve on the surface of a 3D object. The user specifies a feature line E by identifying the pixel locations of its vertices. Once E is specified, our system displays E on top of the interactive rendering of the light field.

To determine the 3D position of a vertex $\mathbf{v}$, we use *geometry-guided manual correspondence*: the user manually identifies projections $p_1(\mathbf{v})$ and $p_2(\mathbf{v})$ of $\mathbf{v}$ in two different views $L_{(s_1,t_1)}$ and $L_{(s_2,t_2)}$ under the guidance of epipolar geometry [9]. After the user specifies $p_1(\mathbf{v})$ in view $L_{(s_1,t_1)}$, the epipolar line of $p_1(\mathbf{v})$ is drawn in view $L_{(s_2,t_2)}$ as a guide for specifying $p_2(\mathbf{v})$ since $p_2(\mathbf{v})$ must be on the epipolar line of $p_1(\mathbf{v})$. Because the camera parameters of both views are known, calculating $\mathbf{v}$ from $p_1(\mathbf{v})$ and $p_2(\mathbf{v})$ is straightforward.

Feature Polygons: A feature polygon P is a 3D polygon defined by n feature lines $\{E^1, ..., E^n\}$, which are called the edges of P. P has *control primitives* $\{E^1, ..., E^{n+k}\}$ which includes both the edges of P and *supplementary feature lines* $\{E^{n+1}, ..., E^{n+k}\}$ for additional control inside the feature polygon. The purpose of a feature polygon is to approximate a surface patch of a 3D object. In general, P is allowed to be non-planar so that it can approximate a large surface patch as long as the surface patch is relatively flat.

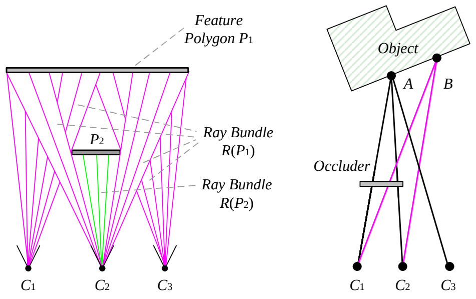

Figure 4: Left: Example of ray bundles in a light field with three cameras. The rays associated with a feature polygon P_1 across all views of the light field constitute ray bundle $R(P_1)$. In this example, $R(P_1)$ includes all pink rays but not the green rays. Right: Example of nearest visible rays in a light field with three cameras. Occluded ray C_1B (pink) is replaced by C_2B (pink), while C_1A is replaced by C_3A.

To specify a feature polygon, the user draws a series of connected feature lines (two consecutive lines sharing a vertex) in counter-clockwise order in the interactive display of a light field. A technical difficulty in the specification process is that, because light field rendering does not perform visible surface computation, all vertices are visible in every view. Fortunately, our experience indicates that the user can easily distinguish vertices on visible surfaces from those on hidden surfaces, for two reasons. First, there are relatively few vertices and a vertex on the visible surface can be identified by the landmark it labels. More importantly, the interactive display gives different motion parallax to visible vertices in the front and invisible ones in the back.

To ensure that the patches are well approximated by feature polygons, we restrict the geometry of the patches. Specifically, for a surface patch S approximated by a feature polygon P, we require that S has no self-occlusion and is relatively flat. We split S if either requirement is not met. By requiring S to have no self-occlusion, we can avoid self-occlusion in P if it is a sufficiently close approximation of S. For such a P, we only have to check occlusion caused by other feature polygons during visibility processing. Note that whether S satisfies the two conditions is solely judged within the viewing range of L. For example, consider any one of the faces in Fig. 3. The surface patch approximated by a feature polygon has no self-occlusion for the viewing range of the light field shown. However, when the viewpoint moves beyond the viewing range of this light field, e.g., to the side of the face, the nose will cause self-occlusion within the surface patch.

Background Edges: We introduce background edges to control rays that do not belong to any feature polygons. These rays exist for two reasons. First, feature polygons only roughly approximate surface patches of a 3D object. In each light field view, rays near the object silhouette may not be covered by the projection of any feature polygons. Second, parts of the object surface may not be affected by the visibility change caused by object shape change. There is no need to specify feature polygons for the corresponding rays.

For rays that do not belong to any feature polygons, we control them with background edges, which are 2D image edges specified by the user. Background edges play the same role as the feature edges in image morphing [1]. A series of connected background edges form a background polyline. As shown in Fig. 3, a background polyline is manually specified in a few key views and interpolated into other views by linear interpolation.

3.2 Global Visibility Map

After specifying all feature elements of a light field L, we can define the global visibility map (or visibility map for short) of L as follows.

Definition 1 *The global visibility map of a light field L with feature polygons $\{P_1, ..., P_{n_p}\}$ is a mapping $V : L \to N$ from the ray space L to the set of integers N such that*

$$V(u, v, s, t) = \begin{cases} i & \text{if ray } L(u, v, s, t) \text{ belongs to } P_i \\ -1 & \text{otherwise} \end{cases}$$

Intuitively, V may be regarded as a light field of false colors, with $V(u, v, s, t)$ indicating the id of the feature polygon visible at ray $L(u, v, s, t)$. Fig. 3 shows examples of visibility maps.
Visibility Computation: The visibility map V is computed based on the vertex geometry of feature polygons[1] as well as the fact that feature polygons have no self-occlusion by construction. The main calculation is that of the visibility of a set of relatively flat but non-planar polygons. This is a calculation that can be done efficiently using OpenGL.

Consider rendering a non-planar polygon P_i into a view $L_{(s,t)}$. A problem with this rendering is that the projection of P_i into the view $L_{(s,t)}$ may be a concave polygon, which OpenGL cannot display correctly. One solution to this problem is a two-pass rendering method using the stencil buffer. This method works for feature polygons since they have no self-occlusion as we mentioned earlier. Alternatively, we can simplify visibility map computation by restricting feature polygons to be triangles without supplementary feature lines, but then the user has to draw many feature polygons, which makes feature specification unnecessarily tedious.
Ray Bundles: Based on the visibility map V, we can group the rays of L according to their associated feature polygons. A group so obtained is called a ray bundle, denoted as $R(P_i)$ where P_i is the associated feature polygon. As we shall see in Section 4, $R(P_i)$ can be warped using ray-space warping with the control primitives of P_i (see the ray-space warping equation (1) in Section 4). The ray correspondence of $R(P_i)$ is thus completely determined by the control primitives of P_i. Rays that do not belong to any ray bundle are called background rays. Background rays are controlled by the background edges.

Ray bundles have been used by Szirmay-Kalos in the context of global illumination [25].

4 Warping

As mentioned, for each $0 < \alpha < 1$, the light field morph L_α is obtained by blending two light fields $\hat{L}_0$ and $\hat{L}_1$, which are warped from L_0 and L_1 for feature alignment. In this section, we discuss the warping from L_0 to $\hat{L}_0$ since the warping from L_1 to $\hat{L}_1$ is essentially the same. We also describe an efficient warping algorithm for animation sequences of 3D morphing.

The warping from L_0 to $\hat{L}_0$ takes the following steps: (a) calculate feature polygons and background edges of $\hat{L}_0$, (b) build the visibility map of $\hat{L}_0$, (c) compute ray bundles of the warped light field $\hat{L}_0$, and (d) treat background rays.

4.1 Basic Ray-Space Warping

Because the rays of a light field L are grouped ray bundles, the basic operation of light field warping is to warp a ray bundle $R(P_i)$. For

[1]Using feature polygons to handle occlusion is related to layered representations in image-based rendering (e.g. [23]).

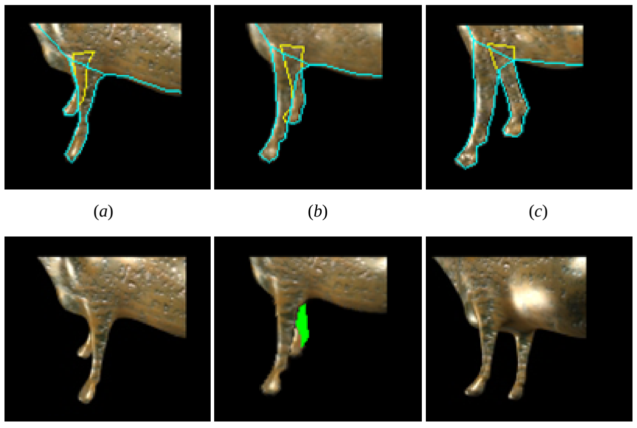

Figure 5: Issues in light field warping. Top row shows interpolation of feature polygons, viewed from $(s,t) = (3,0)$. (a) Light field L_0 with feature lines. (b) Light field $L_{0.5}$ with feature lines. (c) Light field L_1 with feature lines. Bottom row shows a hole caused by object shape change. (d) Light field L_0 viewed from $(s,t) = (3,0)$. (e) Warped light field $\hat{L}_0$ viewed from $(s,t) = (3,0)$. The area highlighted in green is a hole corresponding to the occluded part of a feature polygon in (d). (f) Light field L_0 viewed from $(s,t) = (32,32)$. The feature polygon occluded at view $(s,t) = (3,0)$ is now fully visible.

simplicity, let us assume that L has only an n-sided feature polygon P_i, whose feature lines are $\{E^1, ..., E^{n+k}\}$ before warping and $\{\hat{E}^1, ..., \hat{E}^{n+k}\}$ afterwards.

The basic ray-space warping regards the warped light field $\hat{L}$ as a 4D ray space and directly computes color values of individual rays:

$$\hat{L}(u,v,s,t) = L(u',v',s',t'), \text{ where}$$

$$(u',v')^T = \mathbf{f}(u,v,E^1_{(s',t')}, ..., E^{n+k}_{(s',t')}, \hat{E}^1_{(s,t)}, ..., \hat{E}^{n+k}_{(s,t)}) \quad (1)$$

and (s',t') are free variables in the (s,t)-plane. The vector function $\mathbf{f}()$ is the Beier-Neely field warping function [1]. For a given point (u,v) in view $\hat{L}_{(s,t)}$, $\mathbf{f}()$ finds the preimage (u',v') in view $L_{(s',t')}$ based on the correspondence between the feature lines $E^1_{(s',t')}$, ..., $E^{n+k}_{(s',t')}$ in $L_{(s',t')}$ and $\hat{E}^1_{(s,t)}$, ..., $\hat{E}^{n+k}_{(s,t)}$ in $\hat{L}_{(s,t)}$.

For each ray $\hat{L}(u,v,s,t)$, the basic ray-space warping provides a set of rays $\{L(u',v',s',t')\}$ whose colors can be assigned to $\hat{L}(u,v,s,t)$. Possible values of (s',t') include (s,t), in which case ray-space warping yields the same result as image warping [1].

4.2 Light Field Warping

To warp the light field L_0 to $\hat{L}_0$, we apply the basic warping methods described above to feature polygons of L_0. The warping takes four steps. First, we calculate feature polygons and background edges of $\hat{L}_0$. The vertices of feature lines in $\hat{L}_0$ are linearly interpolated from their counterparts of L_0 and L_1. Fig. 5 (top row) shows an example of feature interpolation. For $i = 0, 1$, let $\{\mathbf{v}^i_1, ..., \mathbf{v}^i_n\}$ be the vertices of feature lines in L_i. The vertices of feature lines in $\hat{L}_0$ are $\{\hat{\mathbf{v}}_1, ..., \hat{\mathbf{v}}_n\}$, where

$$\hat{\mathbf{v}}_k = (1-\alpha)\mathbf{v}^0_k + \alpha\mathbf{v}^1_k, \ k = 1, ..., n.$$

The connections between the vertices are the same in $\hat{L}_0$ and L_0. Thus we can easily obtain the feature polygons of $\hat{L}_0$ as well as their control primitives.

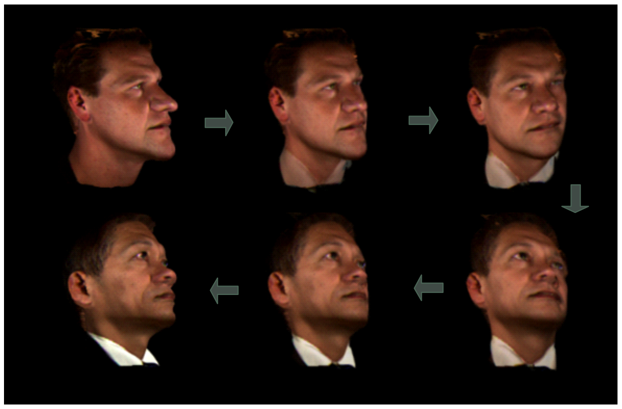

Figure 6: 3D facial morphing.

Second, we build the visibility map of $\hat{L}_0$ and that gives us information about the visibility changes caused by object shape change. Using the edge geometry of feature polygons of $\hat{L}_0$, we can perform the visibility calculation of these polygons, with non-planar polygons rendered by the view-dependent triangulation as before. The result of this visibility calculation is the visibility map of $\hat{L}_0$.

Third, we compute the warped ray bundles of light field $\hat{L}_0 = \{\hat{L}_{0\ (s,t)}\}$ view-by-view. Consider processing ray bundle $R(\hat{P}_0)$ in view $\hat{L}_{0\ (s,t)}$ for feature polygon $\hat{P}_0$ that corresponds to feature polygon P_0 in L_0. We evaluate $\hat{L}_0(u,v,s,t)$ in three steps:

visibility testing We check the visibility map of L_0 to see whether P_0 is visible at ray $L_0(u',v',s,t)$ determined by the ray-space warping equation (1) with $(s',t') = (s,t)$.

pixel mapping If P_0 is visible at ray $L_0(u',v',s,t)$, we let

$$\hat{L}_0(u,v,s,t) = L_0(u',v',s,t).$$

ray-space warping Otherwise, $\hat{L}_{0\ (s,t)}(u,v)$ is in a hole and we invoke ray-space warping to fill the hole. Fig. 5 (top row) shows an example of a hole. The basic ray-space warping described earlier provides a set of values $\{L_0(u',v',s',t')\}$ parameterized by free variable (s',t'). Using the visibility map of L_0, we search for the “nearest visible ray” $L_0(u',v',s',t')$ such that P_0 is visible at ray $L_0(u',v',s',t')$ determined by the ray-space warping equation (1) and (s',t') is as close to (s,t) as possible. This search starts from the immediate neighbors of (s,t) in the (s,t)-plane and propagates outwards, accepting the first valid (s',t'). Note that the search will never fail because P_0 by construction is fully visible in at least one view of L_0. Once (s',t') is found, we set

$$\hat{L}(u,v,s,t) = L_0(u',v',s',t')$$

according to the ray-space warping equation (1).

Fig. 4 illustrates the “nearest visible ray”.

In the last step, we treat background rays, which correspond to pixels not covered by the projection of any feature polygon. We apply image warping to these pixels, using the background edges and (projected) feature polygon edges as control primitives.

The idea behind choosing the “nearest visible ray” is the following. For $\hat{L}_0(u,v,s,t)$, the basic ray-space warping provides a set of values $\{L_0(u',v',s',t')\}$. Under the Lambertian surface assumption, all rays are equally valid. However, the Lambertian surface

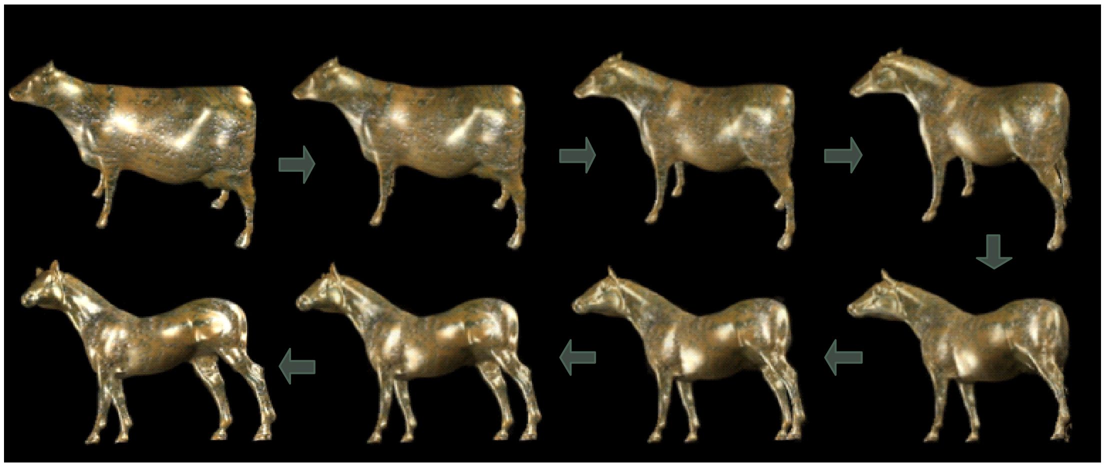

Figure 7: A morphing example with large occlusions and specular surfaces.

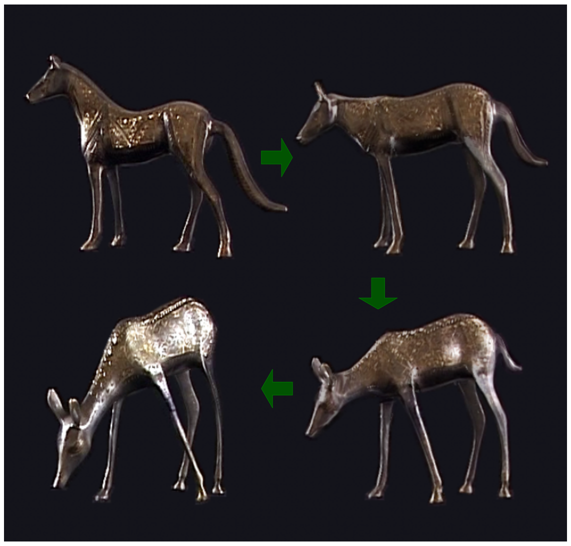

Figure 8: Morphing between two real objects.

assumption only approximately holds despite its widespread use in image-based rendering [4, 21]. By choosing the visible ray nearest to ray $L_0(u', v', s, t)$ when P_0 is occluded at the latter, we are trying to minimize the error caused by the Lambertian surface assumption.

Note that for the "nearest visible ray", we choose a visible ray $L_0(u', v', s', t')$ with (s', t') as close to (s, t) as possible. This is the measure of "closeness" used in [19]. A more natural measure is the angle deviation in [3]. Unfortunately, calculation of angle deviation requires a good estimation of the depth at pixel $L_{0\ (s,t)}(u', v')$. The depth estimation we can get from the associated feature polygon does not necessarily have enough accuracy.

4.3 Warping for Animation Sequences

Our system can produce animation sequences that allow the user to observe a morphing process from a camera moving along an arbitrary path in 3D. In particular, the camera does not have to be inside the (s, t)-plane. One way to compute such a 3D morphing sequence is to first compute a sequence of light field morphs $M = \{L_0, L_{1/n}, ..., L_{(n-1)/n}, L_1\}$ and then create the 3D morphing sequence by rendering the light field morphs in M. Unfortunately, the CPU/storage costs for computing M can be very high. We describe a method for generating a 3D morphing sequence without fully evaluating the sequence M.

Suppose we are given α and we want to compute the image I_α in the morphing sequence. From the known camera path and α, we can find the camera position $\mathbf{v}_\alpha$. The image I_α is a blend of two images $\hat{I}_0$ and $\hat{I}_1$, where $\hat{I}_0$ is warped from L_0 and $\hat{I}_1$ is warped from L_1. The image $\hat{I}_0$ is warped from L_0 by first calculating, for each pixel (x_α, y_α) in the image $\hat{I}_0$, its corresponding ray $(u_\alpha, v_\alpha, s_\alpha, t_\alpha)$ and then applying ray-space warping. The image $\hat{I}_1$ is warped from L_1 the same way.

5 Results and Discussion

We have implemented the light field morphing algorithm on a Pentium III 667 MHz PC. In this section we report some results and a few applications.

3D Morphs and Animations: A 3D morph L_α represents a plausible object having the essential features of both the source and target objects. We can interactively display L_α by light field rendering [19]. We can also generate an animation sequence of the 3D morphing process from a camera moving along an arbitrary path in 3D.

Fig. 6 shows a 3D facial morphing between an Asian male and a Caucasian male. Both light fields are 33×33 in the (s, t)-plane and 256×256 in the (u, v)-plane. These light fields are rendered in 3D Studio Max from two Cyberscan models, each having about 90k triangles; the models are not used for morphing. Feature specification took about 20 minutes. We specified 12 pairs of feature polygons and 9 pairs of supplementary feature lines with 41 pairs of feature points. We also specified 8 background polylines made of 53 background edges. On the average, a background polyline was specified in 8 out of the 1089 views of each light field and interpolated into other views. With our unoptimized implementation, the two global visibility maps took 37 seconds each, whereas light field warping and blending took 15 seconds and 0.5 seconds respectively per image.

Fig. 7 provides an example with large occlusion and specular surfaces. The light fields are acquired the same way at the same resolution as in the 3D facial morphing example. Feature specification took about 30 minutes. We specified 50 pairs of feature polygons and 1 pair of supplementary feature lines with 126 pairs of feature

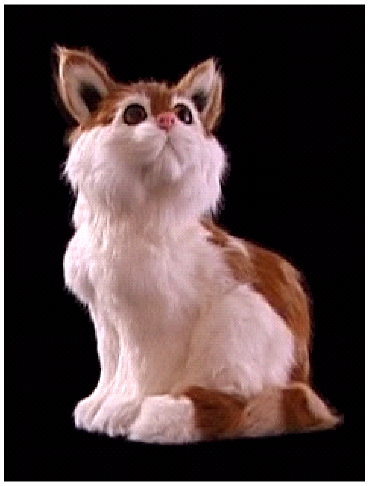

Figure 9: A frame (second from the left) enlarged from the 3D morphing sequence shown in Fig. 1.

points. We also specified 4 background polylines made of 50 background edges. On the average, a background polyline was specified in 8 out of the 1089 views of each light field and interpolated into other views.

Fig. 11 shows the morphing between a real bronze statue and the famous statue of Egyptian queen Nefertiti. The surface of the antique bronze statue shows complicated material property. This is very difficult to model with a textured geometry model, although some nice progress has been made in this area [7].

To acquire the light fields of the bronze statue, we used an outside-looking-in concentric mosaic (CM) [24], which is a slightly different parameterization of the light field. We acquired 300 images of the bronze statue at a resolution of 360×360. The CM of the Nefertiti statue was rendered at the same resolution in 3D Studio Max from a textured model (model not used for morphing). Feature specification took about 20 minutes. As for morphing time, the global visibility map took 15 seconds per map. Warping and blending took 7 seconds per image.

Fig. 8 shows the morphing between two real bronze statues: a deer and a horse. The light fields were acquired at the same resolution as in the Nefertiti example. Feature specification took about 25 minutes. The global visibility map took 25 seconds per map. Warping and blending took 5 seconds per image.

Key-Frame Morphing: As mentioned, key-frame morphing is light field morphing with very sparse light fields. Fig. 1 and Fig. 9 show the result of key-frame morphing between a real furry toy cat and the Stanford bunny. Note that the fur on the cat surface is very difficult to model with textured geometry models.

We took three photographs of the cat using a calibrated camera. Three images of the bunny were rendered using the same camera parameters as the real photographs. Feature specification took about 7 minutes. The global visibility map took 4 seconds to compute for each object. Warping and blending each image of the 3D morphing sequence took 24 seconds.

The number of key frames needed depends on both the visibility complexity of the source and target objects and the presence of non-lambertian surfaces. As expected, the quality of key-frame morphing improves as more input images are used. In this regard, key-frame morphing is more flexible than view morphing [21]. This flexibility is particularly important when, e.g., there are a lot of visibility changes due to object shape change, in which case the nearest visible ray will be frequently needed to fill the holes, and we want the nearest visible rays to be actually nearer for highly non-Lambertian surfaces.

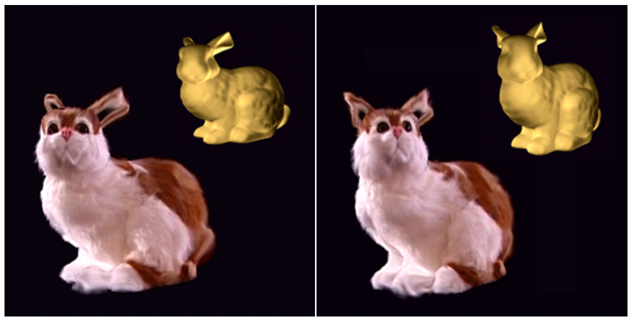

Figure 10: Plenoptic texture transfer. The fur of the furry cat toy in Fig. 1 is transferred onto the surface of the Stanford bunny. The images show the bunny before and after the texture transfer from different view points.

Plenoptic Texture Transfer: Given the source and target objects O_0 and O_1 represented by light fields L_0 and L_1, we transfer the texture of O_0 onto O_1 by constructing a light field L_{01} as follows. First, the feature elements of L_{01} are obtained as those of L_1. Second, L_0 is warped to $\hat{L}_0$ for feature alignment. Finally, L_{01} is obtained as the warped light fields $\hat{L}_0$. Intuitively, we create a morph using the feature elements of L_1 and the radiance of L_0. Unlike 2D texture transfer (e.g., [8]), plenoptic texture transfer is a 3D effect. Fig. 10 shows the result of plenoptic texture transfer from the furry cat toy in Fig. 1 onto the Stanford bunny. Note that for plenoptic texture transfer to work well, the two objects should be similar to avoid texture distortions.

Discussion: In light field morphing, it is easy to handle complex surface properties. Geometry-based 3D morphing, for example, will have difficulties with the furry cat example in Fig. 1. On the other hand, the lack of geometry causes problems in light field morphing. An example is the view point restriction imposed by the input light fields. We can incorporate geometry-based 3D morphing into light field morphing by using image-based visual hulls [20] as rough geometry to morph two light fields. However, the visual hull geometry cannot replace feature polygons because visual hull geometry is obtained from the silhouette and thus cannot handle visibility changes not on object silhouette.

Light field morphing can be regarded as a generalization of image morphing (an image is a 1×1 light field) and as such can suffer the ghost problem in image morphing for poorly-specified feature lines [1]. Fortunately the usual fixes in image morphing also work for light field morphing [1].

6 Conclusions

We have presented an algorithm for morphing 3D objects represented by light fields. The principal advantage of light field morphing is the ability to morph between image-based objects whose geometry and surface properties may be too difficult to model with traditional vision and graphics techniques. Light field morphing is based on ray correspondence, not surface reconstruction. The morphing algorithm we present is feature-based. We built an intuitive and easy-to-use UI for specifying feature polygons for controlling the ray correspondence between two light fields. We also show that the visibility changes due to object shape changes can be effectively handled by ray-space warping. Finally, it is worth to note that light field morphing is a flexible morphing scheme. The user can perform 3D morphing by starting with a few input images and adding more

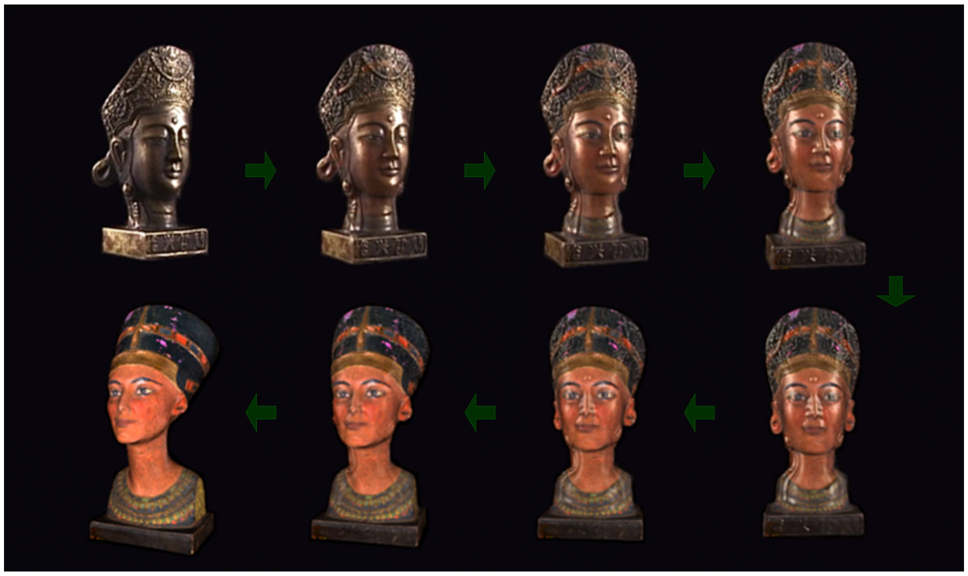

Figure 11: A morphing example involving a surface of complicated material property (the antique bronze statue).

input images as necessary to improve the quality of 3D morphing sequences.

A number of topics remains to be explored. It is desirable to extend our system so that it supports a change of topology [6] and multiple light slabs representing the same object [19]. Another promising area is to use computer vision techniques to automate the feature/visibility specification tasks as much as possible. Finally, we are interested in other operations that can be performed on light fields [18].

Acknowledgments: First of all, we want to thank Hua Zhong, who developed an earlier version of the light field morphing system at Microsoft Research Asia. Since then he moved onto more exciting things and did not have time to work on the current system. Many thanks to Sing Bing Kang for useful discussions, to Yin Li and Steve Lin for their help in video production, to Steve Lin for proofreading this paper, and to anonymous reviewers for their constructive critique.

References

[1] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):35–42, July 1992.

[2] Bryan P. Bergeron. Morphing as a means of generating variability in visual medical teaching materials. *Computers in Biology and Medicine*, 24:11–18, January 1994.

[3] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. *Proceedings of SIGGRAPH 2001*, pages 425–432, August 2001.

[4] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. *Proceedings of SIGGRAPH 93*, pages 279–288, August 1993.

[5] Daniel Cohen-Or, Amira Solomovici, and David Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, April 1998. ISSN 0730-0301.

[6] Douglas DeCarlo and Jean Gallier. Topological evolution of surfaces. *Graphics Interface '96*, pages 194–203, May 1996.

[7] Julie Dorsey and Pat Hanrahan. Modeling and rendering of metallic patinas. *Proceedings of SIGGRAPH 96*, pages 387–396, August 1996.

[8] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.

[9] Oliver Faugeras. *3D Computer Vision*. The MIT Press, Cambridge, MA, 1993.

[10] Jonas Gomes, Bruno Costa, Lucia Darsa, and Luiz Velho. *Warping and Morphing of Graphics Objects*. Morgan Kaufmann, 1998.

[11] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Proceedings of SIGGRAPH 96*, pages 43–54, August 1996.

[12] Arthur Gregory, Andrei State, Ming C. Lin, Dinesh Manocha, and Mark A. Livingston. Interactive surface decomposition for polyhedral morphing. *The Visual Computer*, 15(9):453–470, 1999.

[13] John F. Hughes. Scheduled fourier volume morphing. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):43–46, July 1992.

[14] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):47–54, July 1992.

[15] Francis Lazarus and Anne Verroust. Three-dimensional metamorphosis: a survey. *The Visual Computer*, 14(8-9):373–389, 1998.

[16] Aaron Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. *Proceedings of SIGGRAPH 99*, pages 343–350, August 1999.

[17] Apostolos Lerios, Chase D. Garfinkle, and Marc Levoy. Feature-based volume metamorphosis. *Proceedings of SIGGRAPH 95*, pages 449–456, August 1995.

[18] Marc Levoy. Expanding the horizons of image-based modeling and rendering. In *SIGGRAPH 97 Panel:Image-Based Rendering:Really New or Deja Vu*, 1997.

[19] Marc Levoy and Pat Hanrahan. Light field rendering. *Proceedings of SIGGRAPH 96*, pages 31–42, August 1996.

[20] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 369–374, July 2000.

[21] Steven M. Seitz and Charles R. Dyer. View morphing: Synthesizing 3d metamorphoses using image transforms. *Proceedings of SIGGRAPH 96*, pages 21–30, August 1996.

[22] Steven M. Seitz and Kiriakos N. Kutulakos. Plenoptic image editing. In *ICCV98*, pages 17–24, 1998.

[23] Jonathan Shade, Steven J. Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 231–242, Orlando, Florida, July 1998.

[24] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. *Proceedings of SIGGRAPH 99*, pages 299–306, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[25] László Szirmay-Kalos and Werner Purgathofer. Global ray-bundle tracing with hardware acceleration. *Eurographics Rendering Workshop 1998*, pages 247–258, June 1998.

[26] George Wolberg. Image morphing: a survey. *The Visual Computer*, 14(8-9):360–372, 1998.