Feature-Based Surface Light Field Morphing

Eunhee Jeong\textsuperscript{1} Mincheol Yoon\textsuperscript{1} Yunjin Lee\textsuperscript{1} Minsu Ahn\textsuperscript{1} Seungyong Lee\textsuperscript{1} Baining Guo\textsuperscript{2}

\textsuperscript{1}POSTECH\textsuperscript{*} \textsuperscript{2}Microsoft Research Asia\textsuperscript{†}

Abstract

A surface light field is a function that gives the colors of each object point viewed from different directions. An object representation with a surface light field provides a nice structure for 3D photography. This paper presents a feature-based morphing technique for two objects equipped with surface light fields. The technique consists of geometry morphing and in-between light field mapping. Geometry morphing is accomplished by 3D mesh morphing, where we introduce a vertex merging technique to generate a simpler metamesh. In in-between light field mapping, an in-between object is rendered by extracting necessary fragments from input surface light fields. We also propose an acceleration technique for rendering an in-between object. Experimental results with real and synthetic data show natural and plausible morphing between objects with surface light fields. The proposed morphing technique can be used for an editing tool for 3D photography.

1. Introduction

Metamorphosis, or morphing, is a popular technique for visual effects, which generates a smooth transformation of an object into another. Several approaches have been developed for morphing of objects in different representations. 2D image morphing manipulates the images of objects to achieve compelling illusions of object transformation [3, 12, 15]. 3D object morphing deals with 3D geometry-based objects [10], where techniques were proposed for handling polygonal meshes (e.g., [9, 11]) and volumetric representations (e.g., [7, 13, 5]). Recently, a morphing technique for image-based objects was proposed [17], which generates an in-between light field from two given light fields.

In this paper, we consider morphing of two objects represented with surface light fields. A surface light field contains the colors of each object point seen from different view directions [14, 16, 4]. With a surface light field, we can represent surface properties, such as textures and specularity variations, and render accurate images of the object from arbitrary viewpoints. Recently, an object representation with a surface light field has attracted much attention because it provides a nice structure for 3D photography. Surface light field morphing can be used for an editing tool for 3D photography, which enables warping and morphing of 3D photographs.

Different representations were proposed for compression and rendering of surface light fields [14, 16, 4]. In this paper, we chose the representation proposed by Chen et al. [4] because it produces extremely compact data sets and allows hardware-supported rendering. In the representation, a surface light field is partitioned into vertex light fields, which are factorized into surface and view maps. In the rendering process, called light field mapping, the color of each triangle is determined by the fragments in surface and view maps corresponding to the triangle.

For morphing two objects with surface light fields, we should consider three problems: geometry transformation, surface light field transformation, and rendering acceleration. First, the geometry transformation can be achieved by a 3D mesh morphing technique but the resulting in-between mesh will have a more complicated structure than input. Since surface light field data should be equipped for each component of the in-between mesh, the complicated structure incurs much computational and memory overhead. Hence, it is important to reduce the complexity of an in-between mesh for efficient morphing of surface light fields. Second, the transformation of surface light fields cannot be handled by previous morphing techniques, including light field morphing [17], and will be the main problem of this paper. Finally, since an in-between object simultaneously contains the shape and surface characteristics of two input objects, it has a more complicated data set and takes more time to render than an input object. Thus, an acceleration technique will be useful for rendering an in-between object.
In this paper, we present effective solutions for the problems of surface light field morphing. The main contributions of this paper are:

- **simpler metamesh construction**: We adopt the technique proposed by Alexa [1] to generate a metamesh that merges the geometry of input meshes. However, to produce a simpler metamesh, this paper proposes a vertex merging technique, which drastically reduces the number of vertices in a metamesh. A simpler metamesh allows efficient generation and rendering of the surface light field of an in-between object.

- **in-between light field mapping**: To handle the transformation of surface light field, this paper presents a rendering technique for the triangles of a metamesh, which we call *in-between light field mapping*. We do not explicitly construct an in-between surface light field for a metamesh. Instead, a triangle of a metamesh is rendered by dynamically extracting the necessary fragments from the input surface and view maps. With this approach, we can avoid the computational and memory overhead for constructing an in-between surface light field, while properly interpolating the surface characteristics from the source to target objects.

- **rendering acceleration**: This paper presents an approximation technique for the in-between images from surface light field morphing. The interior of an in-between object is rendered by applying in-between light field mapping to the triangles of input objects which have been transformed to in-between positions. Only the parts around the silhouettes are rendered with metamesh triangles. With this approximation, we can accelerate the generation of an in-between image without much degrading the image quality.

The remainder of this paper is organized as follows. Section 2 gives preliminaries for surface light field morphing. In Section 3, we present the overview of the proposed techniques, which will be detailed in the following three sections. Section 7 shows several morphing examples and Section 8 concludes this paper.

2. Preliminaries

2.1. Surface light field representation

Chen et al. proposed a compact representation and a hardware-supported rendering technique for a surface light field [4]. In the representation, a surface light field is partitioned into a set of vertex light fields. Each vertex light field \( f(r, s, \theta, \phi) \) is decomposed into several surface maps \( g_k[r, s] \) and view maps \( h_k[\theta, \phi] \) which satisfy

\[
f(r, s, \theta, \phi) \approx \sum_{k=1}^{K} g_k[r, s]h_k[\theta, \phi]. \tag{1}
\]

In Eq. (1), the first pair \((r, s)\) indicates a surface location, the second pair \((\theta, \phi)\) indicates a viewing direction, and \(k\) is the number of approximation terms.

With the representation, an object triangle is rendered by blending three vertex light fields at the vertices of the triangle. To render a vertex light field at a vertex, fragments are extracted from the surface and view maps, \( g_k[r, s] \) and \( h_k[\theta, \phi] \), and multiplied pixel-by-pixel by multitexturing. The results of fragments from \( g_k[r, s] \) and \( h_k[\theta, \phi] \), \( k = 1, \ldots, K \), are added to determine the final color provided by a vertex light field.

2.2. Mesh morphing

Mesh morphing techniques based on polygonal representations generally consist of two steps: correspondence establishment and geometry interpolation [10]. In the first step, the vertices and edges of the source and target meshes are embedded onto a common domain such as a sphere [9, 1], 2D polygon [8, 2, 6], and base mesh [11]. Then, a metamesh is created by merging the embedded vertex and edge sets on the common domain. For each vertex in the metamesh, the source and target positions are determined by mapping the vertex onto the surfaces of the source and target meshes, respectively. In the interpolation step, an in-between mesh is generated by interpolating the vertices in the metamesh between the source and target positions.

3. Overview

We first introduce the notations used in this paper. The source and target objects with surface light fields are denoted by \( SLF^2 \) and \( SLF^3 \), respectively. The source object \( SLF^2 \) consists of a light field map \( L^2 \) and a 3D mesh \( M^2 \). Similarly, the light field and the 3D mesh of \( SLF^3 \) are denoted by \( L^3 \) and \( M^3 \), respectively. In this paper, we assume that \( M^2 \) and \( M^3 \) are 2-manifold triangular meshes. An in-between object, its light field, and its 3D mesh are denoted by \( SLF^T \), \( L^T \), and \( M^T \), respectively. For the blending rate \( \alpha \), 0 and 1 imply the source and target objects, respectively. We represent \( SLF^T \) with the rate \( \alpha \) as \( SLF^T_\alpha \), which implies \( SLF^T_0 = SLF^2 \) and \( SLF^T_1 = SLF^3 \). When \( \alpha \) changes from 0 to 1, \( SLF^T_\alpha \) should smoothly transform from \( SLF^2 \) to \( SLF^3 \).

The overall process for morphing two objects with surface light fields consists of two main components, geometry morphing and in-between light field mapping, as illustrated in Figure 1.
4. Geometry Morphing

4.1. In-between mesh generation

To generate a plausible and desired shape of an in-between mesh \( M^t \), the correspondence of features should be established between the source and target meshes, \( M^s \) and \( M^t \). In this paper, feature correspondence is specified by pairs of vertices on \( M^s \) and \( M^t \). Figure 2 shows an example. Several points on input meshes are selected by the user, where the same colored points on \( M^s \) and \( M^t \) correspond to each other.

After the feature correspondence between \( M^s \) and \( M^t \) has been specified, we construct a metamesh by merging the vertices and edges of \( M^s \) and \( M^t \). In this paper, we adopt the metamesh construction technique proposed by Alexa [1], which is based on spherical embedding of meshes (see Figure 2).

Regardless of the blending rate \( \alpha \), \( M^t_\alpha \) has the same connectivity as the metamesh constructed from \( M^s \) and \( M^t \). Hence, to obtain \( M^t_\alpha \) for a given \( \alpha \), it is sufficient to interpolate the vertex positions of the metamesh from \( M^s \) to \( M^t \). In this paper, the position \( p^t_\alpha \) of a vertex in \( M^t_\alpha \) is determined by linearly interpolating the corresponding positions \( p^s \) in \( M^s \) and \( p^t \) in \( M^t \); that is,
\[
p^t_\alpha = (1 - \alpha) \cdot p^s + \alpha \cdot p^t.
\] (2)

4.2. Vertex merging

The vertex set of a metamesh contains the vertices of the source and target meshes, \( M^s \) and \( M^t \), and the intersection points of the edges in the embedding of \( M^s \) and \( M^t \) onto the common spherical domain. The edges and faces in \( M^s \) and \( M^t \) may be partitioned into several pieces on a metamesh. Hence, a metamesh usually has much more complicated structure than \( M^s \) and \( M^t \), which increases the
required memory and rendering time for an in-between object $SLF^i$.

Let $v^s$ and $v^t$ denote the vertices of $M^s$ and $M^t$ embedded onto the common spherical domain, respectively. To reduce the complexity of a metamesh, we merge the pairs of vertices $(v^s, v^t)$ which are close to each other before we compute the intersections of edges on the common spherical domain. The detailed steps of the vertex merging are as follows.

1. For each target vertex $v^t$, we find the source triangle $f^s$ to which $v^t$ belongs on the common spherical domain (see Figure 3). We make three vertex pairs with $v^t$ and three vertices of $f^s$. Each vertex pair is inserted into a priority queue with the distance between the two vertices in the pair.

2. From the priority queue, we extract the vertex pair one by one in the increasing order of the distance. Let $(v^s, v^t)$ be such a vertex pair. If $v^s$ has already been merged with another vertex $v^t'$, we discard the vertex pair because $v^t'$ is closer to $v^s$ than $v^t$. Otherwise, we merge $v^t$ with $v^t$. The vertex merging is not allowed when it causes a triangle flipping.

3. After processing all the vertex pairs in the queue, we apply the relaxation to the vertex positions on the spherical domain in the same way used for sphere embedding [1]. The relaxation step resolves the distortions introduced by the vertex merging.

**Figure 3. Vertex pairs in vertex merging**

The vertex merging process considerably decreases the number of vertices in a metamesh, as demonstrated in Section 7. The proposed vertex merging technique can be applied to other metamesh construction techniques without much modification.

5. In-between Light Field Mapping

To explicitly construct the light field map $L^i$ for an in-between object $SLF^i$, we have to compute the vertex light field for each vertex $v^t$ of the mesh $M^t$. The vertex light field of $v^t$ can be obtained by transforming and blending those of the vertices in $M^s$ and $M^t$ which are adjacent to $v^t$ in the spherical embedding. However, $M^t$ has a larger number of vertices than $M^s$ and $M^t$, and the vertex light field of $v^t$ should be changed with the blending rate $\alpha$. Hence, explicit construction of $L^i$ for $SLF^i$ is inefficient in terms of the required memory and computation.

In this paper, instead of constructing $L^i$, we render the triangles in $M^t$ by extracting and blending the necessary fragments from the source and target light field maps, $L^s$ and $L^t$. Consider a triangle $f^s$ in $M^s$ which is partitioned into several triangles in $M^t$ (see Figure 4). To render a triangle $f^s_m$ in $M^t$, we first extract the fragments from the surface and view maps on $f^s$ which correspond to $f^s_m$ in the spherical embedding. We can then obtain the color of $f^s_m$, given by $L^s$ with Eq. (1). Similarly, we can derive the color of $f^t_m$, given by $L^t$ from the surface and view maps on the face of $M^t$ which contains $f^t_m$. The final color of $f^s_m$ is determined by blending those given by $L^s$ and $L^t$.

**Figure 4. Polygons in input and in-between meshes**

In the remainder of this section, we explain the details of the fragment extraction from the surface and view maps on $f^s$. The same technique can be applied to the extraction from a target face.

5.1. Surface map fragment extraction

Let $g_k[r, s]$ be a surface map on the triangle $f^s$ in $M^s$ (see Figure 5). The fragment of $g_k[r, s]$ corresponding to the face $f^s_m$ in $M^t$ is determined by the position of $f^t_m$ in $f^s$ on the spherical embedding. We can easily obtain the fragment by using the barycentric coordinates of the vertices of $f^t_m$ computed with respect to $f^s$. In Figure 5(b), $p^t_{m}$, $p^t_{m+1}$, and $p^t_{m+2}$ are the positions on $g_k[r, s]$ determined by the barycentric coordinates, and the green region is the fragment extracted for $f^t_m$. Note that the extracted fragment does not change regardless of the blending rate $\alpha$ and the rendering viewpoint.

5.2. View map fragment extraction

Contrary to the case of a surface map, the fragment extracted from a view map changes with the viewpoint. In
addition, the blending rate α has influence on the extracted fragment. For different values of α, the vertex normals of an in-between mesh M\text{t} vary due to the shape change. Then, the color of a vertex in M\text{t} viewed from a viewpoint also changes with the vertex normal. Hence, the view map fragment extracted for a triangle f's in M\text{t} is determined by the vertex normals of f's and the viewpoint.

Let h_k[\theta, \phi] be a view map on the triangle f's in M\text{s} which contains the triangle f't in M\text{t} on the spherical embedding. For a view direction \vec{v}' = (\theta_m, \phi_m), h_k[\theta_m, \phi_m] contains the incoming radiance reflected through the normal vector \vec{n}' (see Figure 6(a)). With the assumption of a reflective BRDF, the incoming radiance information can be specified by the reflection vector \vec{v}'_R [16]. When the normal vector changes to \vec{n}'', the radiance is reflected to another view direction \vec{v}''.

For a given viewpoint, the fragment extracted for f't is designated by three points in h_k[\theta, \phi] which are obtained from the three vertices of f't. Let v' be a vertex of f's and \vec{v}' be the view direction to v'. In the view map h_k[\theta, \phi], the radiance information is stored at the view direction \vec{v}'', which is the reflection vector of \vec{v}'_R through the normal vector \vec{n}' on f's. Hence, we use the direction \vec{v}'' to select the point in h_k[\theta, \phi] which corresponds to the vertex v'. The points in h_k[\theta, \phi] for the other vertices of f's can be obtained in the same way. If the direction \vec{v}'' exceeds the range of h_k[\theta, \phi], we select the nearest direction in h_k[\theta, \phi].

6. Rendering Acceleration

To accelerate the rendering of an in-between object SLF\text{t}, we can blend two images generated by rendering the source and target objects, SLF\text{s} and SLF\text{t}. In that case, before rendering SLF\text{s} and SLF\text{t}, we transform the vertex positions of M\text{s} and M\text{t} to the corresponding positions in M\text{t} (see Figure 7). Remind that M\text{s} and M\text{t} are much simpler than M\text{t} and so SLF\text{s} and SLF\text{t} can be rendered much faster than SLF\text{t}. Figure 8 shows an example with images from SLF\text{s} and SLF\text{t} and their blending.
7. Experimental Results

We have implemented the proposed surface light field morphing technique on a Pentium IV 2.0GHz PC. In this section, we show several morphing examples for the objects with real or synthetic surface light fields. All synthetic data used in this paper were rendered by MaYa™.

Figures 10 through 14 show morphing examples, where the horse and turtle models are equipped with real surface light fields and all the others have synthetic data. In the examples, we can see that the surface characteristics of objects, as well as the geometry, are smoothly transformed in the morphing sequence.

Figures 15 shows an example of texture transfer [17]. In the example, the surface light field of a turtle in Figure 11 is transferred to the 3D mesh of bunny in Figure 11. The texture transfer generates a plausible object with realistic surface characteristics.

For all morphing examples in this paper, feature specification could be completed in 10 to 20 minutes. The metamesh construction takes few minutes on a Pentium IV 2.0GHz PC. Table 1 shows other statistics of the examples. The number of specified feature points increases when the shapes of input objects are much different. With the vertex merging technique, the number of triangles in a simpler metamesh is reduced to about a half of the original metamesh. We could interactively render an in-between object for all morphing examples. The rendering acceleration technique increases the frame rates about twice while it generates approximate images of an in-between object.

8. Conclusion

In this paper, we presented a morphing technique for two objects represented with surface light fields. The technique consists of geometry morphing and in-between light field mapping. Geometry morphing is accomplished by 3D mesh morphing using a metamesh. We introduced the vertex merging technique that generates a simpler metamesh. In in-between light field mapping, an in-between object is rendered by extracting necessary fragments from source and target surface light fields. We also proposed a rendering acceleration technique to speed up the image generation for an in-between object. Experimental results with real and synthetic data show that the proposed technique produces natural and plausible morphing between objects with surface light fields.

Acknowledgments

The authors would like to thank Radek Grzeszczuk for the help on light field map data and the implementation of light field mapping.

References

Table 1. Statistics: The rendering speed is measured on a Pentium IV 2.0GHz PC.

<table>
<thead>
<tr>
<th>models</th>
<th>feature points</th>
<th># triangle</th>
<th>rendering speed (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$M^*$</td>
<td>$M^+$ metamesh</td>
</tr>
<tr>
<td>horse and cow</td>
<td>34</td>
<td>7,002</td>
<td>5,150</td>
</tr>
<tr>
<td>turtle and bunny</td>
<td>45</td>
<td>3,704</td>
<td>5,026</td>
</tr>
<tr>
<td>moai and rabbit</td>
<td>15</td>
<td>4,976</td>
<td>5,004</td>
</tr>
<tr>
<td>rabbit and bunny</td>
<td>28</td>
<td>4,004</td>
<td>5,026</td>
</tr>
<tr>
<td>pawn and rook</td>
<td>18</td>
<td>2,256</td>
<td>2,400</td>
</tr>
</tbody>
</table>

Figure 10. Morphing from a horse to a cow

Figure 11. Morphing from a turtle to bunny
Figure 12. Morphing from a moai to rabbit

Figure 13. Morphing from a rabbit to bunny

Figure 14. Morphing from a pawn to a rook

Figure 15. Texture transfer from a turtle to bunny


