Technical Section

Three-dimensional shape blending: intrinsic solutions to spatial interpolation problems

Li-Gang Liu, Guo-Jin Wang*

State Key Laboratory of CAD & CG and Department of Applied Mathematics, Zhejiang University, Hangzhou 310027, People’s Republic of China

Abstract

This paper presents an efficient algorithm for shape blending between 3-D polylines and between 3-D polyhedral objects based on their intrinsic definitions. Rather than interpolating the vertex locations explicitly, our algorithms determine the intermediate shapes by interpolating the intrinsic variables of the initial and the final shapes. Our algorithms are easy and fast enough in fully interactive time. The generated intermediate shapes are independent of the locations and orientations of the two key frame shapes and all intrinsic variables (the edge lengths and the directed angles) change monotonically. An optimization algorithm is performed for closed requirement. We then generalize our method to shape blending between triangular meshes with arbitrary topology and between freeform curves/surfaces and our results are satisfactory. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Computer animation; Shape blending; Interpolation; Intrinsic variables; MSI algorithm

1. Introduction

This paper deals with shape blending of 3-D polylines and 3-D polyhedral objects. Shape blending determines the in-between shapes that provide a smooth transformation between two given shapes (called the key frame shapes). It is also known variously as shape averaging, shape interpolation, metamorphosis, and shape evolving. It has widespread application in illustration, computer animation, and industrial design [1–3].

Shape blending is usually considered a two-step problem. The first step is to establish a mapping from each point on the initial frame to some point on the final one. Once these correspondences have been already established, the second step is to consider the in-between interpolation method that is employed, bridging between the two key shapes. The first step will be referred to as the correspondence problem and the second step will be referred to as the interpolation problem. The two problems are interrelated since the method used to solve the interpolation problem is dependent upon the manner in which the correspondences are established.

There have been several approaches to determine vertex correspondence automatically [4,5]. Many approaches to the interpolation problem have been published [6–9].

In [8] Sederberg et al. presented an intrinsic solution to the interpolation problem of 2-D shape blending, in which the intrinsic geometric information, such as edge lengths and angles between edges, rather than vertex location, are interpolated. This solution avoids the shrinkage and kinks in the blend and yields better results. To fully realize the benefits of transformations in computer animation and industrial design, 3-D models of the objects must be transformed, instead of just 2-D images of these objets. As Sederberg et al. [3] pointed out: ‘extending this algorithm to polygonal surfaces in 3-D is a worthwhile goal’, it is necessary to find the intrinsic geometric information of 3-D polylines and polyhedra.

Sun et al. [9] consider a polyhedral model as two plane graphs, the vertex adjacency graph and the face adjacency graph, which represent the interrelations between
vertices and faces. The shape parameters, such as edge lengths, dihedral angles and interior angles that interrelate the vertices and faces in the two graphs, are used for interpolation. The results are more satisfactory than the linear or cubic curve paths would, and is translation and rotation invariant. But the method is not the direct generalization of Sederberg’s intrinsic solution in 3-D because it cannot deal with the shape blending between polylines. And it is much complicated and it is difficult to be carried out.

In this paper we take a step to directly generalize the Sederberg’s intrinsic solution to 3-D polylines and meshes and try to get better results. When the 3-D polylines become to the 2-D polylines, our method is the same as Sederberg’s method. Since the main contribution of this paper is the algorithm for spatial geometric interpolation, we suppose that the correspondences between the two key frames have been already established. And we will restrict our discussion to genus zero polyhedra, i.e., polyhedra with no handles.

A shape is the description of an object that we wish to create. It is often easier to model a shape by giving its geometric information rather than describing it explicitly. Thus, we are interested in methods that allow us to create shapes automatically by only giving a set of information. We present a good method to model the information using only intrinsic variables. This method enables us to simulate wide changes in shape transformation by a simple interpolation of intrinsic variables.

Our method of the shape blending for 3-D polylines makes use of a moving local frame associated with each vertex of the polylines. The position of the frame at the vertex depends on the three previous successive vertices. The spherical coordinates of the next vertex in the local frame are the intrinsic variables associated with the vertex. The intrinsic set, consisting of all the intrinsic variables of the 3-D polyline, expresses the intrinsic geometric information because the polylines having the same intrinsic set have the same shape, differing only by a rigid motion. Given the initial and the final polylines, we compute the initial and the final sets of the intrinsic variables. Then, we interpolate the initial and the final corresponding values to obtain the in-between values. Finally, the intermediate shapes with the desired values are reconstructed. In the following the moving spherical coordinate interpolation technique will be referred to as MSI, which is easily understood in three steps as stated above.

We feel that the MSI technique yields good results naturally and simply be generated to the shape blending for 3-D polyhedral objects and 3-D free form curves/surfaces.

- Our MSI technique can be used for the shape blending for some more complex objects such as generalized cylinders or any objects defined by skeleton curves with a common structure.
- The algorithms are easy to be carried out and the computation is fast enough for animation in fully interactive time.
- The geometric shape of the intermediate polylines is independent of the locations and orientations of the two key frame polylines.
- All the intrinsic geometric variables, the edge lengths, and the directed angles, are changing monotonically.
- The algorithms can usually avoid the shrinkage that normally occurs when rotating rigid bodies are linearly blended, and avoid kinks in the blend when they are none in key frames.

The remainder of this paper is organized as follows. Section 2 is a brief mathematical description of the intrinsic definition of 2-D polylines and their blending. The representation of the intrinsic definition of the 3-D polylines is given in Section 3. This is followed by the intrinsic blending algorithm of the 3-D polylines in Section 4. In Section 5, we describe the intrinsic definition of the quadrilateral mesh polyhedral objects and obtain the intrinsic solution to their shape blending. Address our method to the general freeform curves/surfaces and to the arbitrary triangular meshes are introduced in Section 6. Finally, four examples are included in Section 7.

2. Intrinsic definition of 2-D polylines and their blending

A polyline definition that lists the Cartesian coordinates of its vertices can be called an explicit description. Each vertex is located independently of any other vertex and depends only on its coordinates, so there is no geometric information about the polyline. Geometric information denotes variables such as edge lengths and angles between edges, which are referred to as ‘intrinsic variables’. Here we give a brief account of the intrinsic definition of 2-D polyline.

**Definition 1.** Let $P_i, i = 0, 1, \ldots, n$ be the vertices of a 2-D polyline $P$. Denote $\rho_i = |P_i - P_{i+1}|, i = 1, 2, \ldots, n$ and $\theta_i$ the directed angle from the direction $P_{i-1}P_i$ to the
Fig. 1. The intrinsic variables of 2-D polygon.

Obviously, for a 2-D polyline whose vertices do not coincide, we have

**Proposition 1.** A polyline $\Gamma$ determines its intrinsic set $\Omega$ uniquely.

To reconstruct the shape we need to choose an initial point and an initial edge direction, which will give the position and orientation to locate the shape. Algorithm 1 discusses the reconstruction of the entire shape of the polyline from the given intrinsic variables.

**Algorithm 1.** Let $P_0$ be the initial vertex, $X_0$ be the initial unit edge vector and $\Omega = \{\rho_i, \theta_i\}^{n}_{i=1}$ be the intrinsic set. We can determine a 2-D polyline $\Gamma = P_0, P_1, \ldots, P_n$ uniquely from the set $\Omega \cup \{P_0, X_0\}$ as follows.

**Step 1:** Calculate the vertex $P_1 = P_0 + \rho_1 X_0$.

**Step 2:** For $i = 2, 3, \ldots, n$, recursively evaluate the vertex

$$P_i = P_{i-1} + \rho_i \frac{P_{i-2}P_{i-1}}{|P_{i-2}P_{i-1}|} \cdot M(\theta_i),$$

where

$$M(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}.$$

**Proposition 2.** The 2-D polylines that are generated by the same intrinsic set $\Omega$ and the different initial point $P_0$ and/or initial unit edge vector $X_0$ have the same shape, differing only by a rigid body motion.

It is known by Propositions 1 and 2 that we can recursively compute the vertices of the 2-D polyline by its intrinsic set. We can characterize the polyline by specifying that each vertex $P_i$ is deduced from the previous two vertices $P_{i-2}, P_{i-1}$ and the intrinsic variables: the edge length $\rho_i$ and the turning angle $\theta_i$. So we can deduce all the vertices from the initial vertex $P_0$ and the initial vector $X_0$. In fact, $(\rho_i, \theta_i)$ are the polar coordinates of the point $P_i$ in the local planar coordinate system, whose origin is point $P_{i-1}$ and X-axis is in the direction $P_{i-2}P_{i-1}$ (see Fig. 2).

**Fig. 2.** The moving frame $[P_{i-1}; X_i, Y_i]$ at vertex $P_{i-1}$ on 2-D polygon.

3. **Intrinsic definition of 3-D polyline**

We use a similar idea as in 2-D. Moving a local frame along the 3-D polyline to obtain a set of intrinsic variables will also decide the shape of the 3-D polyline uniquely.

**Definition 2.** Let $\Gamma$ be a 3-D polyline with vertices $P_0, P_1, \ldots, P_n$. Let $X_i$ be the unit vector in the direction $P_{i-1}P_{i-1}$, let $Z_i$ be the unit vector in the direction $P_{i-2}P_{i-1} \times P_{i-2}P_{i-1}$, and denote $Y_i = Z_i \times X_i$, $i = 3, 4, \ldots, n$ (see Fig. 3). The set $T_i = [P_{i-1}; X_i, Y_i, Z_i]$, consisting of the point vector $P_{i-1}$ and the mutually orthogonal unit vectors $\{X_i, Y_i, Z_i\}$, is called the moving coordinate system at the vertex $P_{i-1}$. Denote $\rho_i = |P_{i-1}|, i = 1, 2$, the angle between vector $P_0P_1$ and $P_0P_2$, and $(\rho_i, \theta_i, \phi_i) - \pi < \theta_i < \pi, -\pi/2 < \phi_i < \pi/2$ the spherical coordinates of the vertex $P_i$ in the frame $T_i$, $i = 3, 4, \ldots, n$. The values $\rho_i, \theta_i$ and $\phi_i$ are called the intrinsic variables and the set $\Omega = \{\rho_i, \theta_i, \phi_i\}^{n}_{i=1}$ is called the intrinsic set of the polygon $\Gamma$.

**Proposition 3.** A 3-D polyline $\Gamma$ determines its intrinsic set $\Omega$ uniquely.

Similarly, we can characterize the polyline by specifying that each vertex $P_i$ is deduced from the three previous vertices $P_{i-3}, P_{i-2}, P_{i-1}$ and the intrinsic variables $(\rho_i, \theta_i, \phi_i)$ by using vector products and trigonometric functions. Proceeding this way, and supplying a vertex, a direction and a plane to start with, we can deduce, step by step, all the vertices of the polyline. To summarize, our approach will consist of two steps as Algorithm 2.
Algorithm 2. Let \( P_0 \) be the initial point, \( X_0 \) be the initial unit edge vector and \( Z_0 \) be the initial unit planar normal vector \( (Z_0, X_0) = 0 \). Let
\[
\Omega = \{ (\rho_i)^n, (\theta_i)^n, (\phi_i)^n | \rho_i > 0, -\pi < \theta_i \leq \pi, -\pi/2 \leq \phi_i \leq \pi/2 \} \text{ be the intrinsic set. We can obtain a unique 3-D polyline by the set } \Omega \cup \{ P_0, X_0, Z_0 \} \text{ as follows.}
\]

Step 1: Denote \( \pi_0 \) be the plane passing through \( P_0 \) and with normal vector \( Z_0 \). In the plane \( \pi_0 \) we calculate the polyline \( P_0 \) where \( P_2 \) from the set \( (\rho_1, \rho_2, \theta_2) \cup \{ P_0, X_0, Z_0 \} \) by Algorithm 1.

Step 2: For \( i = 3, 4, \ldots, n \), recursively
1. evaluate the local frame \( T_i \) at the vertex \( P_{i-1} \);
2. compute the vertex \( P_i \) by its spherical coordinates \((\rho_i, \theta_i, \phi_i)\) in the frame \( T_i \).

By the process above we assert

**Proposition 4.** The 3-D polylines that are generated by the same intrinsic set and by the different initial point, initial unit edge vector and/or initial unit planar normal vector have the same shape, differing only by a rigid body motion.

By Propositions 3 and 4 we have the truth that the intrinsic set describes the basic geometric nature of a 3-D polylines because it defines the shape independent of translation and orientation. In Algorithm 2, translation is specified using the initial vertex \( P_0 \) and rotation is constrained by designating the initial unit edge vector \( X_0 \) and the initial unit plane normal vector \( Z_0 \). Therefore, a complete description of a 3-D polyline can be obtained by specifying the initial point, the initial unit edge vector, the initial unit plane normal vector and the intrinsic set.

Remark 1. If \( \phi_i \equiv 0 \) for all \( i = 3, 4, \ldots, n \), it is the case for 2-D polyline.

Remark 2. When coincident vertices occur, we have a filtering preprocess as in the appendix.

Remark 3. It is also possible to handle polylines that consist of collinear vertices. If the three successive vertices \( P_{i-3}, P_{i-2} \) and \( P_{i-1} \) are collinear, the local frame at the vertex \( P_{i-1} \) is undefined. In this case we set \( T_i \equiv T_{i-1} \).

### 4. Intrinsic solution to 3-D polyline shape blending

#### 4.1. Shape blending for 3-D polylines

A shape can be determined by its intrinsic variables. Interpolation between the intrinsic variables of two 3-D polylines leads to the transformation of their shapes. If we have the intrinsic variables, we have the corresponding shape.

Given the initial and the final 3-D polylines of the transformation, we assume that both key polylines have the same number of vertices, as will be the case after vertex correspondence is established [4,5]. We first compute the initial and the final values of the intrinsic variables. Then, we interpolate the corresponding values to obtain the in-between values. Finally, the intermediate polyline is reconstructed which has the calculated values of the intrinsic variables. Here we will summarize our algorithm steps.

Algorithm 3. Denote the vertices of the two key polylines \( \Gamma^1 \) and \( \Gamma^2 \), which present at the time 0 and 1 respectively, by \( P_i^1 \) and \( P_i^2 \), \( i = 0, 1, \ldots, n \). The vertices \( P_i^j (i = 0, 1, \ldots, n) \) of the intermediate polylines \( \Gamma^j \), presenting at the time \( t (0 < t < 1) \), are computed as follows.

Step 1: When coincident vertices occur, a special treatment is added as in the appendix.

Step 2: Calculate the intrinsic sets \( \Omega^1 \) and \( \Omega^2 \) of the two key polylines \( \Gamma^1 \) and \( \Gamma^2 \):

\[
\Omega^j = \{ (\rho_i)^n, (\theta_i)^n, (\phi_i)^n | \rho_i > 0, -\pi < \theta_i \leq \pi, -\pi/2 \leq \phi_i \leq \pi/2 \}, \quad j = 1, 2.
\]

Step 3: The intrinsic set \( \Omega^t = \{ (\rho_i)^n, (\theta_i)^n, (\phi_i)^n | \quad j = 1, 2 \}

\[
\rho_i^t = (1 - f(t))\rho_i^1 + f(t)\rho_i^2, \quad i = 1, 2, \ldots, n,
\]

\[
(\theta_i^t = (1 - f(t))\theta_i^1 + f(t)\theta_i^2, \quad i = 2, 3, \ldots, n,
\]

\[
\phi_i^t = (1 - f(t))\phi_i^1 + f(t)\phi_i^2, \quad i = 3, 4, \ldots, n,
\]

where \( f(t) \) is a blending function that simulates the transformation satisfying \( f(0) = 0 \) and \( f(1) = 1 \). For simplicity, we choose the linear blending function \( f(t) = t, 0 \leq t \leq 1 \).

Step 4: Determine the initial point \( P_0^t \), the initial unit edge vector \( X_0^t \) and the initial plane normal vector \( Z_0^t \).
Step 5: By Algorithm 2 evaluate the vertices $\mathbf{P}_i^*$, $i = 0, 1, \ldots, n$, of the intermediate polyline $\Gamma'$ from the set $\Omega' \cup \{\mathbf{P}_0^*, \mathbf{X}_0^*, \mathbf{Z}_0^*\}$.

Remark 4. Note that Step 4 only influences the relative position of the interpolated polyline and not its shape. It is reasonable and natural that $\mathbf{P}_0^*, \mathbf{X}_0^*$ and $\mathbf{Z}_0^*$ can be obtained by interpolating the corresponding values of $\Gamma'$ and $\Gamma'$.

Since the intrinsic definition of 3-D polyline is invariant to rigid body motion, a shape blend must specify translations and rotations for intermediate shapes. Translation is specified using the initial vertex and rotation is constrained by designating the initial unit edge vector and the initial unit plane normal vector. We can use these initial anchor positions to control the motion of the intermediate polylines.

As stated in [12], many techniques exist for designing an object with an axis, most of which consist of using the axis defined while creating the entity, for instance the axis of a sweep surface or the tree-like skeleton of an implicit surface. So the method MSI can also be used for the shape blending between generalized cylinders or implicit surfaces defined by skeleton curves because they may be animated through the use of their skeleton. The generalization of the shape blending using the star-skeleton representation of the 3-D polyhedra and the MSI method will be discussed in our second paper.

4.2. Closed constrained condition

If the two key frame polylines $\Gamma'$ and $\Gamma'$ are both closed, i.e., the initial vertex and the final vertex are coincident, it is required that the intermediate polylines be closed. Unfortunately, the resulting polylines will not generally close at the stage of our Algorithm 3. The problem remains as to how to best adjust the edge lengths and the angles so that the polylines do close.

Usually the intermediate polylines come much close to ending where they started when the key polylines are closed. We can leave the angles unchanged and adjust the edge lengths.

Let $\Gamma = \mathbf{P}_0^* \mathbf{P}_1^*, \ldots, \mathbf{P}_n^*$ be the generated intermediate polyline. Let $(l_i, x_i, \beta_i), i = 0, 1, \ldots, n - 1$, be the spherical coordinates of edge vector $\mathbf{P}_i^* \mathbf{P}_{i+1}^*$ in the world coordinate system $T = \{\mathbf{O}; \mathbf{X}, \mathbf{Y}, \mathbf{Z}\}$. To close the polyline we adjust the edge lengths as

$L_i = l_i + s_i, \quad i = 0, 1, \ldots, n - 1.$

The closed conditions are

\[
\phi_1(s_0, s_1, \ldots, s_{n-1}) = \sum_{i=0}^{n-1} (l_i + s_i) \cos x_i \cos \beta_i = 0,
\]

\[
\phi_2(s_0, s_1, \ldots, s_{n-1}) = \sum_{i=0}^{n-1} (l_i + s_i) \sin x_i \sin \beta_i = 0,
\]

\[
\phi_3(s_0, s_1, \ldots, s_{n-1}) = \sum_{i=0}^{n-1} (l_i + s_i) \sin x_i = 0.
\]

It seems smart that the magnitudes of $s_i$ should roughly be proportion to $l_i$. Our goal is to find $s_0, s_1, \ldots, s_{n-1}$ so that the objective function

\[
f(s_0, s_1, \ldots, s_{n-1}) = \sum_{i=0}^{n-1} \frac{s_i^2}{l_i^2}
\]

is minimized subject to the three above constraints. We can now use the method of Lagrange multipliers to solve the desired tweak values $s_i$ as follows. Set

\[
\Phi(\lambda_1, \lambda_2, \lambda_3, s_0, s_1, \ldots, s_{n-1}) = f + \lambda_1 \phi_1 + \lambda_2 \phi_2 + \lambda_3 \phi_3,
\]

where $\lambda_1, \lambda_2$ and $\lambda_3$ are the multipliers. From

\[
\frac{\partial \Phi}{\partial \lambda_1} = \sum_{i=0}^{n-1} (l_i + s_i) \cos x_i \cos \beta_i = 0,
\]

\[
\frac{\partial \Phi}{\partial \lambda_2} = \sum_{i=0}^{n-1} (l_i + s_i) \cos x_i \sin \beta_i = 0,
\]

\[
\frac{\partial \Phi}{\partial \lambda_3} = \sum_{i=0}^{n-1} (l_i + s_i) \sin x_i = 0,
\]

we obtain

\[
A\lambda_1 + B\lambda_2 + C\lambda_3 = U,
\]

\[
B\lambda_1 + E\lambda_2 + F\lambda_3 = V,
\]

\[
C\lambda_1 + F\lambda_2 + G\lambda_3 = W,
\]

where

\[
(A, B, C, E, F, G, U, V, W) = \left(\sum_{i=0}^{n-1} l_i^2 \cos x_i \cos \beta_i, \sum_{i=0}^{n-1} l_i^2 \cos^2 x_i \sin \beta_i \cos \beta_i, \sum_{i=0}^{n-1} l_i^2 \sin x_i \cos x_i \cos \beta_i, \sum_{i=0}^{n-1} l_i^2 (\cos x_i \sin \beta_i)^2, \right)
\]
\[
\sum_{i=0}^{n-1} l_i^2 \sin z_i \cos z_i \sin \beta_i \sum_{i=0}^{n-1} l_i^2 \sin^2 z_i,
\]
\[
2 \sum_{i=0}^{n-1} l_i^2 \cos z_i \cos \beta_i, 2 \sum_{i=0}^{n-1} l_i^2 \cos z_i \sin \beta_i,
\]
\[
2 \sum_{i=0}^{n-1} l_i^2 \sin z_i.
\]

Thus under the normality condition we can get \( (\lambda_1, \lambda_2, \lambda_3) \)
\[
\begin{bmatrix}
A & B & C \\
B & E & F \\
C & F & G
\end{bmatrix}
^{-1}
\begin{bmatrix}
U & B & C \\
V & E & F \\
W & F & G
\end{bmatrix}
= \begin{bmatrix}
A & U & C \\
B & V & F \\
C & W & G
\end{bmatrix}
\begin{bmatrix}
A & B & U \\
B & V & E \\
C & W & F
\end{bmatrix}
\]

and

\[ s_i = -\frac{1}{2} l_i^2 (\lambda_1 \cos z_i \cos \beta_i + \lambda_2 \cos z_i \sin \beta_i + \lambda_3 \sin z_i), \]

\( i = 0, 1, \ldots, n - 1. \)

We can now calculate the coordinates \((x_i, y_i, z_i)\) of the vertices \(P_i\):

\[ x_i = x_{i-1} + l_{i-1} \cos z_{i-1} \cos \beta_{i-1}, \]
\[ y_i = y_{i-1} + l_{i-1} \cos z_{i-1} \sin \beta_{i-1}, \]
\[ z_i = z_{i-1} + l_{i-1} \sin z_{i-1}, \]

\( i = 1, 2, \ldots, n. \)

This method generally gives good results and is relatively fast. Furthermore, in most cases very little edge length adjustment is needed according to our experience. This can be detected by checking the values of \(s_i\).

An alternative approach is to treat the open polyline as a piece of wire which can stretch and bend at polyline vertices because some examples can be found where the edge lengths may change more than is desirable using the edge-tweaking-only method. Adjustments to angles and/or edges can then be computed by forcing the two unclosed joints to coincide, and determining the unique equilibrium shape of the wire as in [3].

5. Shape blending for quadrilateral polyhedra

Our MSI technique for shape blending as described so far allows only 3-D polylines. In order to blend the 3-D objects, we must generalize our technique to the surfaces. Surfaces in computer graphics are often represented using polygonal meshes. In order to minimize the number of special cases, we first restrict ourselves to the shape blending between regular quadrilateral meshes, which by topological regularity we mean a tensor product structure with four edges meeting at every vertex, since almost any surface representation (spline, implicit surface, volumetric) can be converted with arbitrary accuracy to a quadrilateral mesh by some polygonization process. Also our aim is to set up local frames that lead to the definition of polyhedron through their intrinsic variables and not through an explicit construction.

**Definition 3.** Denote \( A \) a mesh of 3-D \((m + 1) \times (n + 1)\) quadrilaterals with nodes \( P_{ij}, i = 0, 1, \ldots, m, j = 0, 1, \ldots, n \) (see Fig. 4). Denote \( \rho_{01} = |P_{00}P_{01}|, \rho_{10} = |P_{00}P_{10}| \) and \( \gamma = \angle P_{01}P_{00}P_{10} \). According to Definition 2, denote \((\rho_{ij}, \theta_{ij}, \phi_{ij})\) the spherical coordinates of \( P_{ij} \) in the frame determined by \( P_{i-1,j-1}P_{i,j-1} \) and \( P_{i-1,j}P_{i-1,j-1} \), where \( P_{-1,0} = P_{00} \); and denote \((\rho_{0j}, \theta_{0j}, \phi_{0j})\) the spherical coordinates of vertex \( P_{0j} \) in the frame determined by \( P_{0,j-1}P_{0,j-2} \) and \( P_{0,j-1} \), \( j = 2, 3, \ldots, n \), where \( P_{0,j-1} = P_{0,j} \) for \( j = 1, 2, \ldots, m \). The values \( \rho_{ij}, \theta_{ij}, \phi_{ij} \) are called the intrinsic variables and the set \( \Theta = \{ \rho_{ij}, \theta_{ij}, \phi_{ij} \mid 0 \leq i \leq m, 0 \leq j \leq n, i + j \geq 2 \} \cup \{ \rho_{01}, \rho_{10}, \gamma \} \) is called the intrinsic set of the 3-D mesh \( A \).

Given the intrinsic set \( \Theta \), supplying \( P_{00}, P_{01} \) and \( P_{10} \) to start with \((\angle P_{01}P_{00}P_{10} = \gamma)\), we can deduce all the nodes of the mesh step by step. First, the nodes along the two boundary directions of the mesh are determined by Algorithm 2. Then, each node \( P_{ij} \) \((i = 1, 2, \ldots, m, j = 1, 2, \ldots, n)\) is deduced from the three adjacent nodes \( P_{i-1,j}, P_{i,j-1}, P_{i,j} \) and the intrinsic variables \((\rho_{ij}, \theta_{ij}, \phi_{ij})\) by using vector operation and trigonometric functions. The following algorithm describes the process of reconstruction.

![Fig. 4. Configuration of a quadrilateral mesh of \((m + 1) \times (n + 1)\) grid.](image)
Algorithm 4.

Step 1: Determine the three initial vertices \( P_{00}, P_{01} \) and \( P_{10} \) by satisfying \( \angle P_{01} P_{00} P_{10} = \gamma \).

Step 2: Denote \( P_{-1,0} = P_{01} \) and \( P_{0,-1} = P_{10} \). Using Algorithm 2,

2.1. compute \( P_{i0}, i = 2, 3, \ldots, m \), from the set \( \{\rho_{i0}, \theta_{i0}, \varphi_{i0}\} \) starting with \( P_{-1,0}, P_{00} \) and \( P_{10} \);

2.2. compute \( P_{0,j}, j = 2, 3, \ldots, n \), from the set \( \{\rho_{0,j}, \theta_{0,j}, \varphi_{0,j}\} \) starting with \( P_{0,-1}, P_{00} \) and \( P_{01} \);

Step 3: For \( j = 1, 2, \ldots, n \), and for \( i = 1, 2, \ldots, n \), recursively

3.1. evaluate the local frame \( T_{i,j} \) at \( P_{i-1,j-1} \) determined by \( P_{i-1,j}, P_{i-1,j-1} \) and \( P_{i,j} \) as in Definition 3;

3.2. compute \( P_{ij} \) by its spherical coordinates \( (\rho_{ij}, \theta_{ij}, \varphi_{ij}) \) in the frame \( T_{i,j} \).

Remark 5. Step 1 only influences the relative position of the mesh, not its shape.

The intrinsic set \( \Theta \) describes the basic geometric nature of the polyhedra mesh \( A \) and defines the shape uniquely. Interpolation between the intrinsic variables of the quadrilateral polyhedra leads to the transformation of their shapes. Given two quadrilateral polyhedra, we first compute the initial and the final values of the intrinsic variables. Then we interpolate the corresponding values to obtain the in-between values. Finally, the intermediate quadrilateral polyhedron is reconstructed by Algorithm 4.

Fig. 5. The marching definition of triangulation meshes.

Fig. 6. The transformation from a helix curve into a circle. The intermediate polylines are interpolated at time \( t = 0.0, 0.25, 0.5, 0.75, 1.0 \). All the polylines are visualized by sweeping a circle along them.
6. Shape blending for free form curves and surfaces

6.1. Shape blending for general free form curves and surfaces

We consider the shape blending between two general free form curves by our MSI technique. We first make a piecewise linear approximation to the curve. This introduces the possibility of sampling problems. For simplicity, the \( n + 1 \) vertices are sampled uniformly in the parametric space of the curve. Then we use our technique to blend the two discrete approximation polylines. It is obvious that this approximation can be made more precise by increasing the number of segments in practice. Furthermore, it is clear that better sampling can result in a better blending, some sort of curve subdivision techniques must be employed, but it may hinder the efficiency of the algorithm and eliminate its interactive capabilities. Either arc length sampling or curvature-based adaptive sampling can be suggested but both necessitate significant overhead analysis of the free form curves. It is questionable and probably application dependent whether such a preprocessing analysis is desired.

For free form surface defined in a rectangle parametric space, simple uniform samples lead to a discrete approximation quadrilateral polyhedron. We can blend two surfaces by blending their discrete approximation quadrilateral polyhedra.

6.2. Shape blending for triangular meshes of arbitrary topology

In this section the topological type of a mesh refers to the connectivity between its vertices rather than its genus. Since the surfaces are generally represented by triangular meshes in computer graphics, it is of great importance to generalize our MSI techniques to the shape blending between the triangular meshes.

Suppose that the three vertices \( A, B \) and \( C \) of a triangle are known (see Fig. 5), by Definition 2 we define the spherical coordinates of vertex \( D \) in the frame determined by \( A, B \) and \( C \) as the intrinsic variables of \( D \). Similarly, the spherical coordinates of vertex \( E \) in the frame determined by \( A, C \) and \( D \) are defined as the intrinsic variables of \( E \), and so on. It is required that the triangular meshes

Fig. 7. The transformation from a bottle gourd surface into a torus surface. The intermediate frames are interpolated at time \( t = 0.0, 0.25, 0.5, 0.75, 1.0 \).
satisfy the manifold criteria [11]. There are numerous choices of the order of defining the vertices, so we consider the triangular meshes with their vertices as a plane graph structure, and define each vertex by a breadth-first traversal algorithm [11] for a graph which is described easily using an iterative procedure. Thus, after specifying a given triangle, we can obtain the intrinsic variables of other vertices in some sequence. We call it a marching definition. Here the main issue is the choice of a root. After that, any graph traversal (in our examples, breadth first) can be used to extract a tree.

Given two key frames of 3-D triangular meshes with common topology, the correspondence between the vertices of the two meshes is then established. Then we calculate the intrinsic variables of the two key frame meshes by their identical marching definition. The intrinsic variables of the immediate frame are obtained by interpolating the corresponding intrinsic variables of the key frames. Thus the immediate triangular meshes are reconstructed by the calculated intrinsic variables in accordance with the sequence in the marching definition.

Obviously the choice of a root influences the resulting blend.

We can also blend two surfaces by blending their discrete approximation triangulation meshes that obtained by some approaches for triangulating surfaces.

7. Examples and conclusion

Figs. 6–9 present some examples of our transformation algorithms. The examples are rendered using faceted shading to better illustrate the intermediate shapes.

Fig. 6 shows a helix curve transforming into a circle, where we use one polyline segment sampled with 50 points. All the polylines in the example are visualized by sweeping a circle along them. Fig. 7 shows the transformation between two free form surfaces, one of which is a bottle gourd surface and the other is a torus surface. Fig. 8 shows the transformation from a banana into a pear. Our last example, Fig. 9, illustrates the shape

Fig. 8. The transformation from a banana into a pear. The intermediate frames are interpolated at time $t = 0.0, 0.25, 0.5, 0.75, 1.0$. 
Fig. 9. The transformation from a 3-D digitized sculpture into an "S" shaped object. The intermediate frames are interpolated at time $t = 0.0, 0.25, 0.5, 0.75, 1.0$.

```
P_1 = P_2 = P_3

Fig. 10. Treatment of coincident vertices.
```

blending between a 3-D digitized sculpture and an "S" shaped object.

We summarize the examples with a hint on the computational costs. Animating a metamorphosis sequence between two curves can be accomplished in an interactive speed of several frames a second. An intermediate polyhedron can be obtained in about half a second. Resorting to a larger sampling will obviously have some impact on the computation time. The examples are accomplished on an SGI IRIX 6.2 UNIX Workstation.

All the examples have demonstrated the usefulness of our MSI techniques to smoothly transforming between key frames. Our MSI techniques interpolate the intrinsic variables of the two key frames in natural and intuitive ways. They are adapted to the control of local geometry property (angles) as well as global geometric property (lengths). And the intermediate shapes are invariant to the relative position of the key frames. Moreover, the simplicity of the computation allows us to carry out the transformations in fully interactive time.

Several aspects of the interpolation problem are still under investigation. First, extensions of the MSI method to handle wider classes of polyhedra will be investigated. For complicated meshes or non-genus 0 objects, decomposing the objects into some simple ones connected with a tree-like skeleton structure as in [10] is possible. Second, we are interested in examining the problem of self-intersections during the interpolation. A good solution to this problem has applicability for many other problems
that involve interpolation, not just shape blending. The third problem currently under investigation is the design of a correspondence algorithm that can work well together with our MSI technique.

Acknowledgements

We are thankful to Dr. Lin-Jun Chen, Dr. Hu-Jun Bao and Wei Chen for their help for the head in Fig. 9. This work was supported by the National Natural Science Foundation of China under grant number 69473040 and by the Zhejiang Provincial Natural Science Foundation of China. It is also supported by the State Key Basic Research and Development Fund of 973 Project.

Appendix A

Coincident vertices are a common occurrence which invite special attention. When \( m \) adjacent vertices on one polygon lie at the same point, \( m - 1 \) edges collapse to that point. Since \( \angle P_{i-1}, P_i, P_{i+1} \) is undefined if \( P_i \) is coincident with either of its neighbors, the angle interpolation when such a case is involved in a shape blend is also undefined. It is expected that in our method a prefiltering process might be beneficial. In this case, a low pass filter can alleviate these difficulties.

As in [3], we imagine that coincident vertices actually lie evenly spaced along the base of an infinitesimal isosceles triangle as shown in Fig. 10. In this case \( P'_i = P_i + \epsilon (P_0 - P_1) \), \( P'_3 = P_3 + \epsilon (P_4 - P_3) \) and \( P'_2 = (P'_i + P'_3)/2 \) where \( 0 < \epsilon << 1 \). In general, if vertices \( P_0, P_{i+1}, \ldots, P_{i+k} \) are coincident, \( P'_i = P_i + \epsilon (P_{i-1} - P_i) \), \( P'_{i+k} = P_{i+k} + \epsilon (P_{i+k+1} - P_{i+k}) \) and \( P'_{i+j} = (1-j/k)P'_i + (j/k)P'_{i+k}, j = 1, 2, \ldots, k \).

References