Parametric Curves on Meshes

Takashi Kanai
RIKEN

Abstract

In this paper, we propose a method for creating parametric curves on triangular meshes. A curve on a mesh is frequently used as a boundary curve of a specific region of a mesh in mesh modeling and applications such as texture mapping, remeshing or morphing. Although the curve defined in this paper is a piecewise linear approximation of a strict parametric curve, it is guaranteed that such a curve is just on a mesh. The basic idea is creating a curve on a spherical parameterization instead of direct definition on a mesh. The computation of this curve is done by using only the control points on a spherical parameterization which does not depend on the number of vertices in a mesh. This enables interactive creation/modification of curves even for dense meshes.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, Surface, Solid, and Object Representations; I.3.5 [Computer Graphics]: Graphics Utilities—Graphics Editors

Keywords: Geometric modeling, Triangular mesh, Parametric curve, Quaternion curve, Spherical parameterization.

1 Introduction

Unstructured triangular meshes (hereafter referred to as *meshes*) are the most standard representation of surface geometry in Computer Graphics (CG) applications due to their simplicity and flexibility. Recent progress of shape measurement machines such as range scanners facilitates the acquisition of dense range data. Mesh reconstruction from such range data is still an active research area. Furthermore, there is a much greater need for the re-use of such reconstructed meshes in the applications of three-dimensional CG, CAD/CAM, etc.

Now let us consider defining a curve on such an unstructured mesh. A curve on a mesh is frequently used as the boundary curve in applications such as texture mapping, remeshing and morphing. In these applications, a curve which is strictly represented by mathematical representation is not required. Instead, the *visual* smoothness of a curve and the interactivity which can be freely created or modified by users would really be needed.

In this paper, we propose a novel method of interactive curve creation and modification as a parametric curve on a mesh. In past researches on curve definition, the approach of Krishnamurthy and Levoy [Krishnamurthy and Levoy 1996] can be cited first. In [Krishnamurthy and Levoy 1996], a sampling method of dense point sets on a mesh based on B-spline curve interpolation is described. However, only point samples are generated on a mesh, and the defined curve is not strictly on a mesh. In the work of Kanai and Suzuki [Kanai and Suzuki 2001], they propose an approach to define an approximate straight line between two points on a mesh. This approach is based on using selective refinement strategy of Dijkstra's algorithm for the graph structure of a mesh. Unfortunately, a global search of mesh edges (some of them may be subdivided) is required. Furthermore, a dense subdivision of a graph to generate a higher accurate path is needed and, in this case, computation takes time.

On the other hand, a simple approach conceivable by anyone is the "projection"-based scheme. In this approach, a spatial curve (e.g. B-spline curve) is interpolated from point sets and is projected onto a mesh. However, there is a need to decide the direction on which a curve is projected. If points sampled from a curve project to the corresponding nearest points on a mesh, the nearest point search will be required. Such findings typically require an additional data structure (e.g. spatial data structure such as octree) which will however make computation costly. Since our objective here is to create a curve interactively, operations with high computational costs are to be avoided. If all point samples are projected in the same direction, projections could fail as in the case of mesh geometry (especially highly convex/concave regions).

The basic idea of our approach is to define a curve in the parametric space generated from a mesh, not on a mesh itself. A curve defined on a parametric space is just a parametric curve based on strict mathematical formula, thus its controllability can be maintained. Our approach uses only points on a mesh corresponding to control vertices to evaluate a curve. Therefore the computation is local and does not depend on the number of mesh vertices. These two characteristics enable an interactive creation and modification of curves on a mesh. To prove this advantage, we also describe two applications, curve editing and mesh decomposition in the later section.

This paper is organized as follows: Section 2 gives an overview of our curve generation procedure, and Section 3 explains the details of our algorithm with related work. Section 4 discusses applications of parametric curves on meshes and Section 5 provides the conclusions of this paper.

2 Overview of Our Basic Procedure

Figure 1 shows an overview of our basic procedure for curve generation. The input is a genus zero mesh with arbitrary connectivity and the output is a curve drawn on a mesh. The procedure is described as follows:

1. For a mesh M, we apply *spherical parameterization* $\phi : M \mapsto P$ [Kanai 2004] and store each parameter $p \in P$ to a vertex of a mesh $v \in M$. This computation is done only once as a pre-process.
2. After specifying several points on a mesh, we compute a *quaternion curve* $L(t)$ using corresponding points on a sphere.
3. $L(t)$ is projected onto a spherical embedding to compute a *piecewise linear approximation* $p_i \in P (i = 1 \dots n)$. Each ver-

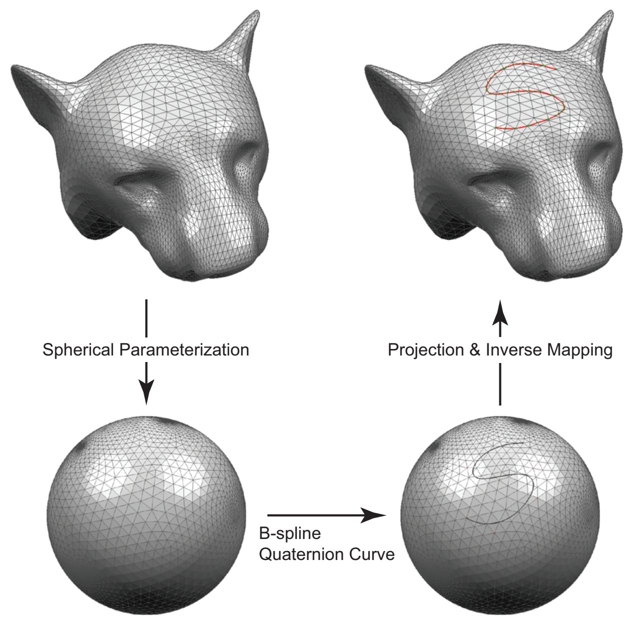

Figure 1: Overview of our basic procedure for curve generation.

tex of a linear approximation on a mesh $v_i \in M$ is computed by an inverse mapping ϕ^{-1}.

We apply the spherical parameterization method, which unfortunately could limit objects which can be handled to genus zero meshes.

A curve on a mesh is just a piecewise linear approximation because a mesh itself is defined as a piecewise linear surface. It is indeed impossible to define a curve as mathematical expression. However, a line segment $\overline{p_i p_{i+1}}$, which is converted from a part of a curve, is just on M.

In the case of multiple curves, the above steps 2 and 3 are repeated.

3 Algorithm Details

In this section, we will describe the details of two specific techniques; quaternion curve and conversion from a curve to linear approximation, used in our basic curve generation procedure.

3.1 Quaternion Curve

Quaternion curve is a classical tool for the control of orientations in the fields of computer animation and robotics. In past years, various quaternion curve methods have been proposed ([Shoemake 1985; Kim et al. 1995b; Ramamoorthi and Barr 1997; Miura 2000; Buss and Fillmore 2001] and so on). We offer an alternative usage of these quaternion curves.

We use here a *unit B-spline quaternion curve* proposed by Kim et al. [Kim et al. 1995b]. There are several reasons that we adopt Kim et al.'s approach; a smoothly-connected composite curve is easy to create, the computation only depends on the number of control vertices, and the algebraic and the differential property of a planar B-spline curve can be inherited. Moreover, a unit B-spline quaternion curve is formulated with arbitrary number of control vertices as a planar B-spline curve; therefore there is a lot of flexibility for designing a curve. Our purpose here is to create a visually-smooth and controllable curve. Consequently, Kim et al.'s approach is much suitable in our case.

A point on a quaternion curve $q(t)$ is calculated using the following formula:

$$q(t) = q_0 \prod_{i=1}^{n} \exp(w_i \tilde{B}_{i,k}(t)), \quad w_i = \log(q_{i-1}^{-1} q_i), \tag{1}$$

where q_i denotes a control point as the form of quaternion, exp and log denote an *exponential map* and a *logarithmic map* of quaternion respectively. $\tilde{B}_{i,k}(t)$ means the *cumulative form* of the B-spline basis function $B_{j,k}(t)$:

$$\tilde{B}_{i,k}(t) \quad = \quad \sum_{j=i}^{n} B_{j,k}(t).$$

One issue should be considered when creating unit B-spline quaternion curves on the sphere. A unit quaternion $q = (\cos\theta, \sin\theta \cdot (a,b,c)) \in S^3$ is by nature composed of a *direction* (a,b,c) and a *rotation* θ. However, as pointed out in [Miura 2000], a curve generation problem on the sphere is nothing less than to define an *orientation* (= direction + rotation) from a given parameter. Consequently, a point on the sphere corresponds to an infinite number of unit quaternion. When applying the above issue to our case, we have to determine the rotation term when converting a unit quaternion from a point on the sphere. Our simple solution for this issue is to set $\cos\theta = 0$, $\sin\theta = 1$ for all conversions to unit quaternion.

3.2 Projection and Inverse Mapping

A unit B-spline quaternion curve described in the previous subsection is not just on a spherical embedding. Here we project a curve on a spherical embedding to create a piecewise linear approximation. A piecewise linear approximation is represented as a poly-line on a mesh: $l = \{p_1, p_2, \ldots, p_n\}$. A vertex of such a poly-line is defined by a vertex of a mesh or a point on an edge of a mesh.

The projection algorithm is as follows:

Algorithm 1 *Curve projection algorithm*

```
**Input** Spherical embedding $P$ and a quaternion curve $q(t)$
**Output** A poly-line $l = \{p_1, p_2, \ldots, p_n\}$ on $P$

$t = 0$; $p_i = q(t)$;
while ( $p_i \neq q(t = 1)$ )
    $x_{i-1} = p_i$;
    while ( $f_{i-1} == f_i$ )
        $t = t + \Delta t$;
        Compute a nearest point $x_i \in P$ to $q_i = q(t)$ and its
        corresponding face $f_i$;    ... (1)
        if ( $x_i == nil$ ) $\Delta t = \Delta t * 0.1$;    // Make $\Delta t$ smaller
    end
    Compute an intersection point $p_i$ between a neighboring
    edge $e$ of $f_i$, $f_{i-1}$ and line segment $\overline{x_i x_{i-1}}$;
    ... (2)
end

end
```

The algorithm starts at $t = 0$. t is incrementally updated to $t + \Delta t$ by an interval Δt in each step. We then compute a point on the

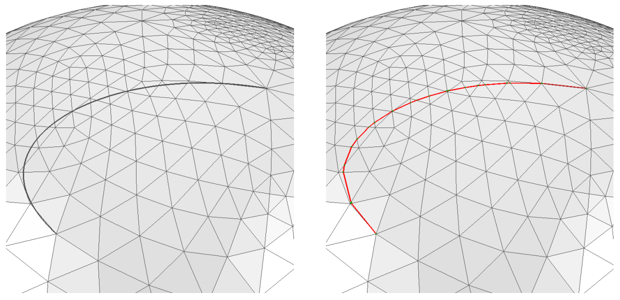
(a) (b)

Figure 2: (a) B-spline quaternion curve on sphere. (b) Piecewise linear approximation projected on spherical embedding.

curve $q(t)$ and project it to a face to compute the nearest point x_i and its corresponding face f_i ((1) in Algorithm 1). In the nearest point search, x_i is calculated by an intersection between a face and its perpendicular line extending to q_i. We only have to search a face f_i and its three neighbor faces which can be quickly accessed by preparing the appropriate data structure such as half-edge.

If corresponding faces f_{i-1}, f_i of two consecutive nearest points x_{i-i} and x_i are different, an intersection point p_i between a common edge e of f_{i-1}, f_i and a line segment $\overline{x_i x_{i-1}}$ is computed ((2) in Algorithm 1). In the intersection point calculation, either face of f_i, f_{i-1} is rotated so that both faces are co-planar to define a line segment $\overline{x_i x_{i-1}}$. The above two processes are repeated until $t = 1$.

In the curve projection algorithm, we should take care how to set an interval Δt. If we set Δt to an excessively large value, we cannot find the nearest point described in (1) in Algorithm 1. Also, if we set Δt to an excessively small value, it takes longer time to compute a piecewise approximation. Here we adopt the following strategy for setting Δt from several experiments: We determine an initial value of Δt by 10% for an average length of all the edges of a mesh. In the algorithm, we make Δt smaller by tithe unless a nearest point can be found.

Figure 2 illustrates a unit B-spline quaternion curve on a sphere and a piecewise linear approximation projected on a spherical embedding. The calculation of the inverse mapping is quite simple because of the correspondence between a mesh M and its spherical embedding P: When computing an intersection point on P ((1) in Algorithm 1), a corresponding point on M can also be computed.

4 Applications

In this section, we will discuss the following two applications of curve generation, curve editing and mesh decomposition.

4.1 Curve Editing

Our curve editing tool can not only create curves but also translate or rotate them freely on a mesh with interactive speed. Figure 3 shows the examples of interactively rotating and translating character-shaped curves on a mesh.

There are different area ratios of faces of a mesh for faces of a spherical embedding depending on regions of a mesh. In curve editing, this issue causes a *scaling problem*. For example, when a curve passes a region in which the area of faces on a spherical embedding is relatively small (e.g. a region of the arm), a curve becomes large as shown in Figure 3(c).

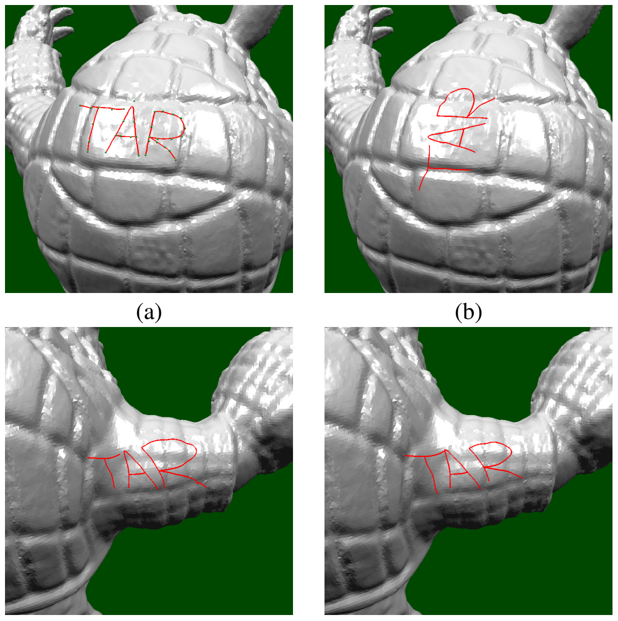
(a) (b) (c) (d)

Figure 3: Curve editing examples on “Armadillo” mesh (172,974 vertices, courtesy of Stanford University). (a) Characters “TAR” on back. (b) 90°-Rotation. (c) Translation to arm without scaling. (d)Translation with scaling.

To overcome such a scaling problem, we adjust the size of a curve depending on its position: Firstly, an *area ratio coefficient* $\gamma = \sqrt{area(f_P)/area(f_M)}$ is computed for each face in advance. An average of such coefficients for faces through which a curve (or a group of curves) passes is computed to adjust its scale. Figure 3(d) shows the results of adjusting such a scale. In this case, a curve on a spherical embedding also becomes relatively smaller. The characters “TAR” as shown in Figure 3 is composed of seven curves. Each curve has from four to nine control points. The computation time for drawing such curves mainly depends on the curve projection algorithm as described in Section 3.2. However, the computation time for drawing all seven curves is less than 1.0×10^{-4} seconds (measured on Pentium 4 2.4GHz PC), which may realize the establishment of an interactive operation. On the other hand, the computation time for spherical parameterization is roughly ten minutes for the model in Figure 3. This computation is done only once as a preprocess, and thus does not affect the interactive editing of curves. With our editing tool, an interactive displacement mapping could also be possible by selective refinement in a region centering on a curve.

4.2 Mesh Decomposition

Mesh decomposition into patches is an important technique, especially in the application of 3D morphing [Gregory et al. 1999; Lee et al. 1999; Kanai et al. 2000; Michikawa et al. 2001; Praun et al. 2001]. In applications to 3D morphing, a smooth boundary curve between patches is usually required for a visually-acceptable interpolation of two meshes. On the one hand, several automatic decomposition methods have been proposed [Katz and Tal 2003; Boier-Martin 2003]. In contrast, our decomposition approach is

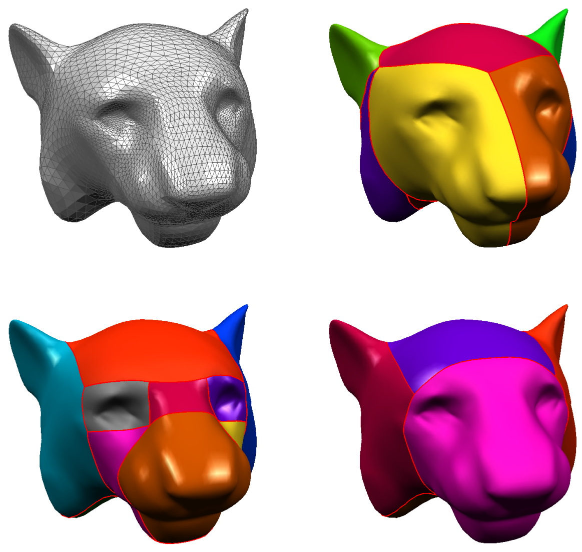

Figure 4: Various mesh decomposition results for "tiger's head" mesh (4,034 vertices). The number of parametric curves for decomposition is 22, 26, and 12 respectively (from left to right).

manual-based. We think that our manual decomposition approach has both advantages and disadvantages. The advantage is that we can edit boundary curves which are really needed. Our approach can be implemented with high interactive speed even when accompanied with try-and-error operations. The disadvantage is that considerable work is needed for users because it is a completely manual operation. However, a *semi-automatic* approach which combines our approach with automatic decompositions should be possible. For example, an automatic decomposition approach can be first used, followed by the detailed manual modification of only a part of boundary curves. Figure 4 shows the results of demonstrating several different decompositions for a mesh. It is difficult to modify boundary curves freely as shown in these figures by the automatic approach. Our approach can also extract some regions by specifying several curves.

5 Conclusions and Future Work

In this paper, we have proposed a curve on a mesh utilizing spherical parameterization. Our approach can create a visually-smooth and controllable curves on a mesh with interactive speed. We have also shown from two practical applications that our system has the potential to be a powerful curve generation/editing tool on a mesh.

Two directions should be considered in future work. The first is the improvement of our approach. Currently, there are two sub-themes for this. One is the process for meshes with genus greater than zero. For this sub-theme, the use of texture atlas may be possible as noted in Section 2. However, it brings about the question of how to maintain continuity when a curve crosses over a boundary of two patches.

The other sub-theme is the generation of an interpolation curve which passes point sets. This seems to be possible by using an interpolative quaternion curve on the sphere [Kim et al. 1995a].

The other direction is developing a user interface for curve editing. We are especially interested in utilizing a tablet interface.

References

BOIER-MARTIN, I. M. 2003. Domain decomposition for multiresolution analysis. In *Proc. 1st Eurographics Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, 29–40.

BUSS, S. R., AND FILLMORE, J. P. 2001. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics 20*, 2, 95–126.

GREGORY, A., STATE, A., LIN, M., MANOCHA, D., AND LIVINGSTON, M. 1999. Interactive surface decomposition for polyhedral morphing. *The Visual Computer 15*, 9, 453–470.

KANAI, T., AND SUZUKI, H. 2001. Approximate shortest path on a polyhedral surface and its applications. *Computer Aided Design 33*, 11 (September), 801–811.

KANAI, T., SUZUKI, H., AND KIMURA, F. 2000. Metamorphosis of arbitrary triangular meshes. *IEEE Computer Graphics and Applications 20*, 2 (April), 62–75.

KANAI, T. 2004. Hierarchical computation of conformal spherical embeddings. In *6th International Conference on Mathematical Methods for Curves and Surfaces*.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (Proc. SIGGRAPH 2003) 22*, 3 (July), 954–961.

KIM, M.-J., KIM, M.-S., AND SHIN, S. Y. 1995. A C^2-continous B-spline quaternion curve interpolating a given sequence of solid orientations. In *Proc. Computer Animation '95*, IEEE CS Press, Los Alamitos CA, 72–81.

KIM, M.-J., KIM, M.-S., AND SHIN, S. Y. 1995. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Computer Graphics (Proc. SIGGRAPH 95)*, ACM Press, New York, 369–376.

KRISHNAMURTHY, V., AND LEVOY, M. 1996. Fitting smooth surfaces to dense polygon meshes. In *Computer Graphics (Proc. SIGGRAPH 96)*, ACM Press, New York, 313–324.

LEE, A. W. F., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution mesh morphing. In *Computer Graphics (Proc. SIGGRAPH 99)*, ACM Press, New York, 343–350.

MICHIKAWA, T., KANAI, T., FUJITA, M., AND CHIYOKURA, H. 2001. Multiresolution interpolation meshes. In *Proc. 9th Pacific Conference on Computer Graphics and Applications (Pacific Graphics)*, IEEE CS Press, Los Alamitos CA, 60–69.

MIURA, K. T. 2000. Unit quaternion integral curve: A new type of fair free-form curves. *Computer Aided Geometric Design 17*, 1, 39–58.

PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent mesh parameterizations. In *Computer Graphics (Proc. SIGGRAPH 2001)*, ACM Press, New York, 179–184.

RAMAMOORTHI, R., AND BARR, A. H. 1997. Fast construction of accurate quaternion splines. In *Computer Graphics (Proc. ACM SIGGRAPH '97)*, ACM Press, New York, 287–292.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Computer Graphics (Proc. SIGGRAPH 85)*, ACM Press, New York, 245–254.