Reconstruction of B-spline skinning surface from generalized cylinder mesh

Ken Yano
Graduate School of Engineering
Hiroshima University
1-4-1 Kagamiyama
Higashi Hiroshima, Hiroshima, Japan
email: d064016@hiroshima-u.ac.jp

Koichi Harada
Graduate School of Engineering
Hiroshima University
1-4-1 Kagamiyama
Higashi Hiroshima, Hiroshima, Japan
email: harada@mis.hiroshima-u.ac.jp

Abstract

We propose a novel method to reconstruct B-spline surfaces from generalized cylindrical meshes by skinning. Skinning is a well known surface creation technique and has been used in CAD and CG modeling. However there are few papers which address the issue of automated creation and preparation of sectional curves for skinning. Although our method is only applicable to generalized cylindrical meshes, there are many real world objects which can be created or reconstructed by skinning. The proposed surface reconstruction method is fully automated with minimal user interventions. We have evaluated the validity of this method by reconstructing B-spline surfaces from various polygonal meshes varying in shapes and geometries. The final results show the effectiveness of our proposed method.

1 Introduction

Reconstruction of free-form spline surfaces from other formats such as point sets, polygonal meshes, etc has been studied by many researchers. Especially in CAD modeling, measuring point coordinates from existing objects then generating a smooth free-form surface which interpolates or approximates these points is called ”reverse engineering” and many papers have dealt with this issue [2] [3] [12] [13] [15] [22] .

[22] describes the two-step approach to construct a NURBS surface from a point set. This paper is one of a few papers which deal with the calculation of the weights of control points using symmetric eigenvalue decomposition. [2] deals with the important issue of how to avoid the explosion of control points by using knot-control method. [13] presents a technique to assign parameter values to randomly measured points. The parameterization is first realized by projecting the measured points to a base surface and later is refined by reducing fitting error from the base surface. [12] presents an approach to update a local area of a B-spline surface based on a set of locally distributed and unorganized points. The region of the original surface to be updated is first identified and then the control points affecting these area are updated by modifying part or all of the control points through a local fitting process. [8] presents a system for converting a dense irregular polygon mesh of arbitrary topology into tensor product B-spline surface patches with accompanying displacement maps. Skinning is simply a newer term for lofting and is a widely used technique in the shipbuilding, automotive, and aircraft industries as well as the sweep surface construction techniques. The details of such surface construction methods can be

found in [16]. Readers are referred to [3] about the general issues pertaining to skinning especially when the sectional curves are rational B-spline curves.

This paper tries to automate the construction of sectional curves on a triangle mesh in order to convert the mesh into a skinning surface. Defining or drawing a smooth curve on a mesh is not trivial and there are several papers addressing this issue [17]. For example, Dijkstra's algorithm is known to be a good algorithm to find a shortest path on a mesh, however the path will not be a smooth curve. [17] computes geodesic paths on a mesh by solving the Eikonal equation by Fast Marching Method. Geodesic path may not be used for the sectional curve since the path would skirt around humps on the mesh which are important surface features. Our method to generate sectional curves for skinning is motivated by the work [18]. They generate a smooth scalar field on mesh by solving Laplacian equation, then create a set of gradient flows by tracing the field from a selected point on the mesh. The gradient flows are used for quadrilateral remeshing.

This paper is organized as following. In section 2, the details of generation of gradient flows are descried. Although our method mostly follows the work [18], we restate it for clarity. In section 3, the gradient flows are sorted then converted to a set of B-spline sectional curves. In order to prevent "explosion of control points" by skinning, each sectional curve is parameterized by adopting the candidate knots which are fully described in [2]. In section 4, a skinning surface is made from the sectional curves. The total number of control points of the skinning surface is suppressed as much as possible. We also discuss some issues of known surface parameterization and show a solution for them. In section 6, we apply our method to various mesh objects and the resulting skinning surfaces are presented with the timing analysis. Section 6 concludes our work and states some future works.

2 Gradient flows on triangulated mesh

We assume that we are given a triangular polygonal mesh $M = (V, F)$, composed of a set of vertices V and a set of triangles F. Each vertex v is assigned a position p in $3D$ Euclidean space. We also restrict that the mesh M need to be a generalized cylinder. The mesh M might have a boundary at each end.

First we define a smooth scalar field s which assign a scalar value to each vertex. Within each triangle, the scalar values are linearly interpolated. The scalar field is defined as a solution of harmonic function satisfying the Laplace equation $\triangle s = 0 \quad (\triangle = \partial^2/\partial x^2 + \partial^2/\partial y^2 + \partial^2/\partial z^2)$ subject to the Dirichlet boundary conditions. Secondly gradient flows on the mesh are generated. We adopts the method proposed by [18]. The method is efficient in terms of computational time and robust since the generated scalar field is smooth everywhere we are less likely to have accumulated errors when tracing flow lines.

The details of each step are described in the following.

step0 : initialization At initialization a octree enclosing the mesh is defined by recursively subdividing a octant until one side length of the octant becomes user defined size. This octree is later used to speed up searching for neighboring points on the mesh.

step1 : setting of boundary conditions In this step, users define vertices with minimum and maximum scalar values. If the mesh has no boundary, select one vertex as minima where the scalar value is 0, and one vertex as maxima where the scalar value is 1. If the mesh has boundary at either end, a set of vertices belonging to the boundary to be either minima or maxima. Those vertices defined as minima or maxima become the boundary conditions when solving the Laplacian equation.

step2: generate smooth scalar field The smooth scalar field s is defined as the solution of

$$\triangle s = 0 \tag{1}$$

where $\triangle$ is the Laplacian operator subject to the Dirichlet boundary conditions described above.

A function s satisfying the constrained Laplace equation is known to generate a discrete harmonic function. An important consequence of the harmonics of s is that it will have no local extrema other than at constrained vertices. This is important as it implies that the flows of gradient will converge precisely at the specified constrained points and flow smoothly everywhere else.

On a triangulated manifold, the usual discretization of Laplacian operator is defined as a umbrella operator with *cotangent* weight. We have used the method described in [5].

step3: generate gradient vector field Having computed the scalar field s, the gradient vector field $\boldsymbol{g}$ is defined such that,

$$\boldsymbol{g} = \nabla s \quad (\nabla = \partial/\partial x\boldsymbol{i} + \partial/\partial y\boldsymbol{j} + \partial/\partial z\boldsymbol{k}) \tag{2}$$

Given a triangle $T = (p_1, p_2, p_3 \in R^3)$ with scalar values $s1, s2, s3$ associated with each vertex, the gradient of scalar field is defined by interpolating scalar values s_i over a triangle T. The component is a linear combination of scalar values $(s1, s2, s3)$ and the coordinate (see [10]).

step4: tracing of flow line Given an arbitrary seed point on the mesh, we trace a flow line in both the positive and negative gradient direction separately until we encounter the maximum or minimum point, respectively. In order to trace the flow line, we need to consider three cases (see Figure 1). These three cases need to be carefully checked and be correctly handled otherwise we might end up with zigzag flow lines or two flows might cross over. Readers are also referred to [18] about the details of implementation.

Every time when the flow line intersects an edge or a vertex, we place two new seed points at both sides of the flow line (see Figure 1). The spacing of the seed and the incident flow point is controlled by a user defined spacing function $h = \theta \times (dx + dy + dz) \quad \theta \in R$, where dx, dy, and dz is the average distance between neighboring vertices for each coordinate. θ is adjusted to control the total number of generated flows. As the new seed is generated, it is registered into the octree. The seed is also registered into a priority queue with associated distance to the closest flow node by querying in the octree.

step4: selection of next seed point By getting a next element off the priority queue, the next seed is obtained. The seed is again checked whether there is no flow nodes within the user specified distance h around it. If it is false, another element is taken from the queue. This checking is necessary since newly generated flow nodes might invalidate the closest distance when the seed is first registered into the priority queue.

We continue the process from step3 and step4 until no valid seed can be taken from the queue.

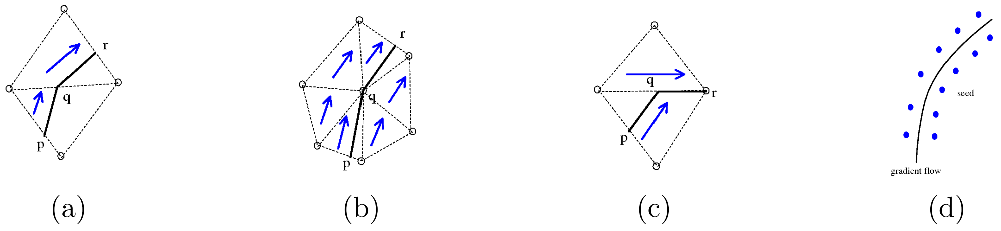


Figure 1: Tracing a gradient flow; (a) regular case (b) vertex case (c) edge case (d) generated seed points along a gradient flow

3 Sectional curves for skinning

In the preceding steps, we have obtained a set of gradient flows and every adjacent flows are mutually at least h, e.g., the user defined distance, away. Since each gradient flow is generated by taking a seed point from the priority queue described above, generated gradient flows may not be sequenced around the mesh. And also, the maximum distance between every two adjacent flows may not be exactly equal. In order to create sectional curves, first the gradient flows have to be sorted. Secondly a set of gradient flows have to be selected to generate sectional curves. The detail of the method is described as follows.

3.1 Sorting of gradient flows

We first sort the gradient flows around the mesh to make sectional curves. We can manually define the order of generated flows by selecting each flow one by one, however manual sorting is cumbersome and takes long time for complicated objects. We propose an automatic sorting. The method automatically sort gradient flows by projecting selected point of each flow to a plane and sort the flows by the order of projected points in that plane.

Figure 2 illustrates this algorithm for two cases (a) a generalized cylinder with no boundaries (b) a generalized cylinder with boundaries. The flows are sorted by the angle Θ defined on the projecting plane. In case (a), the normal of projecting plane is given by the normal of the extrema point p_0. In case (b), the projecting plane is defined as the plane approximating the boundary points.

Let c_0 be the center of the boundary points $\{p_0\}$, we define the covariance matrix C as follows

$$C = \Sigma_i (p_0^i - c_0)(p_0^i - c_0)^T \tag{3}$$

The eigen vector corresponding to the smallest eigen value gives the normal of the plane.

The projected flow points $\{p'\}$ can be computed as

$$p' = p_k - c_0 - dot(p_k - c_0, N) * N \tag{4}$$

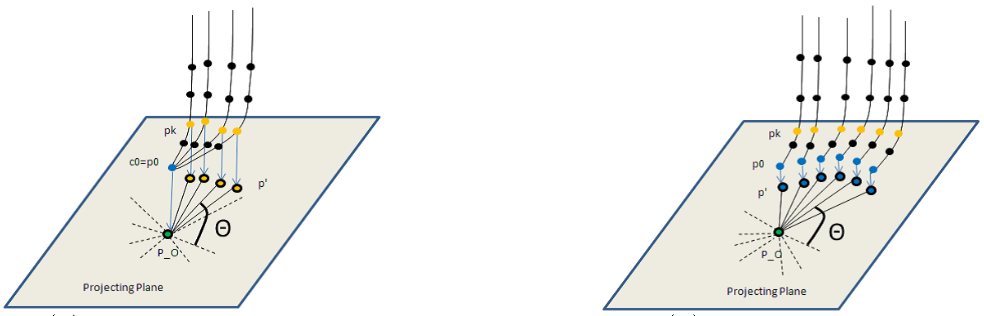
for case (a)

$$p' = p_0 - c_0 - dot(p_0 - c_0, N) * N \tag{5}$$

for case (b), where N is the normal of the projecting plane defined above.

The optimal index k can not be chosen in general, however for most meshes predefined index should be fine. In our test cases, $k = 5$ is chosen. Moreover in case (b), the sorting method does not work when the shape of boundary is not convex. In such cases, we need to resort to some other methods. An intuitive solution uses the fact that the points $\{p_0\}$

are on the boundary edges of the original mesh. As we trace the boundary edges of the original mesh, record the order of time whenever encountering boundary flow nodes. The flows are then sorted by the time. In order to accelerate this method, each flow node needs to record the information on which edge of the original mesh it resides.



(a) without boundaries (b) with boundaries

Figure 2: Sorting of gradient flows. In case (a), $c_0 (= p_0)$ is the common extrema point. In case (b), c_0 is the center of boundary points $\{p_0\}$. P_O is the projection of c_0 onto the projecting plane. Points in blue are extrema points $\{p_0\}$ and points in orange are the flow nodes $\{p_k\}$ with index k

3.2 Selection of gradient flows

Given the sorted gradient flows $\{f_i \ : i = 0..M\}$, we select a set of candidate flows to make sectional curves $\{C_j : j = 0..K\}(K >= 4)$ for skinning. Since the generated flows might be overly generated, only a subset of them is required for skinning. A simple method is to uniformly sample the flows without regarding the shape of the mesh. If K is too small, the skinning surface will just approximate the original object. If K is too large, the skinning surface will contains many unnecessary control points and might causes artifacts on the surface.

The following code snippet is to sample K flows uniformly from M flows. Since the spacing between each of original neighboring flows is not exactly equal, we call it *semi-uniform selection.*

```
int S[K]; // selected flows
// M;  total num of flows
// K;  num of flows for sectional curves
void Semiuniform_selection(int M, int K)
{


  float err;
  float r=(float)M/(float)K;
  err = 0;
  U = 0;
  S[0]=U;
  for (int i = 1; i < K; i++){
     u = U + r;
     U = (int)floor(u);
     err += u - U;
     if (err > 1.0){
       U++;
       err -= 1.0;
     }
     S[i] = U;
}
```

Semi-uniform selection does not consider the shape inherent in the mesh. The more optimal sampling is to adopt an adaptive method. The *adaptive selection* samples the flows with respect to local curvature of the boundary curve, which results in a more efficient approximation. The strategy of this method is described as

1. choose a criterion for refining samples.
2. evaluate the criterion at the middle of the interval flow points.
3. if the boundary curve of the flow points is almost flat in the interval, then the sample is given by its two extremes
4. otherwise, divide the interval into two parts and recursively sample two parts.

The following code snippet implements the algorithm. For the criterion above, we choose the angle $\angle amb$ for checking of flatness. Note that the method generates flow index in exact order they occur along the boundary, as the recursive implementation performs a depth-first search of the interval. After the *adaptive selection* is done, duplicated flow indexes in S need to be removed.

```
p0[M]; // sorted boundary points
int S[]; // selected flows
void adaptive_selection(int a,int b)
{
   int m;
    m = (a+b)/2;
    if (flat(p0[a],p0[m],p0[b])) {
       add_flow(a);
       add_flow(b);
    }else{
       adaptive_selection(a,m);
       adaptive_selection(m,b);
    }
}
```

Figure 3 illustrates the result of two different sampling methods for a simple object.

The above *adaptive selection* method only considers the boundary shape of the mesh so it may not be an optimal sampling for other parts if the mesh is more complicated. In those cases, semi-uniform sampling method can be used.

3.3 Least squares curve fitting

For each flow, a non-rational B-spline curve is fitted by least square approximation. Let $Q_k (k = 0..n)$ be flow points, the B-spline curve $C(u)$ is defined as

$$C(u) = \Sigma_{i=0}^{n} N_{i,p}(u) P_i \qquad u \in [0, 1] \tag{6}$$

where $N_{i,p}$ is the B-spline basis function and P_i is the control points [16].

- $Q_0 = C(0)$ and $Q_n = C(1)$
- the remaining Q_k are approximated in the least squares sense, i.e.

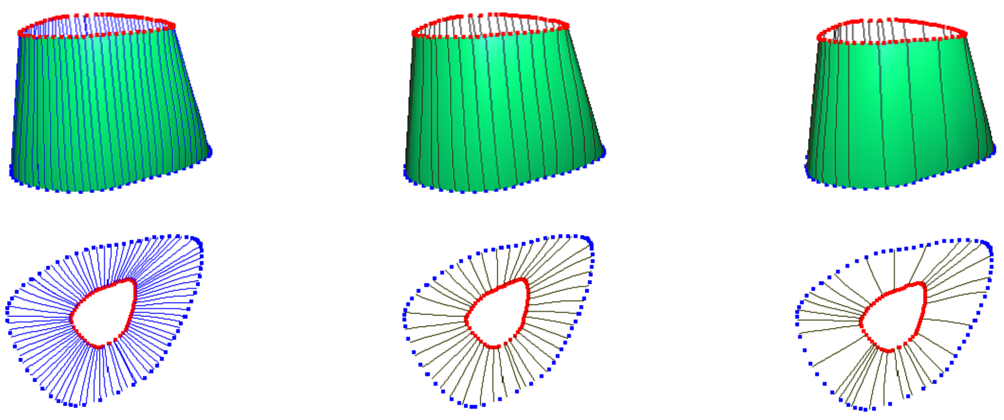

(a) gradient flows (b) semi-uniform sampling (c) adaptive sampling

Figure 3: Selection of gradient flows

$$\Sigma_{k=1}^{n-1}|Q_k - C(u_k)|^2 \tag{7}$$

is a minimum with respect to the control points P_i. The u_k are the precomputed parameter values. We have computed u_k using chord length parameterization with candidate knot [2]. Knot vector for each sectional curve have to be defined with special care, since these curves are later to be changed to be compatible by knot merging. When each sectional curve is parameterized disregarding to other curves, the number of control points will explode by knot merging. We avoid this issue by adopting the method described in [2]. To briefly explain their method, although the choice of the knot vector is crucial in curve fitting, each knot has some flexibility locally. That is, the position of the knots can be perturbed without causing either numerical problems or creating curves of unacceptable shape.

The idea is to define an interval in which each internal knot can freely be moved. Denoting the knot vector for degree p approximation by u^p, these intervals are defined as

$$u_{k-1}^{p-1} \leq u_k^p \leq u_k^{p-1} \tag{8}$$

That is, the knot for a degree p approximation is bracketed into an interval defined by knots used for a degree $p-1$ approximation. Introducing a percentage parameter per, one can control the width of each interval. In other words, $per = 0\%$ gives no flexibility, whereas for $per = 100\%$, any value within $(u_{k-1}^{p-1}, u_k^{p-1})$ may be chosen [2]. The algorithm accepts the data points and a knot vector as input. It then sees if the given knots can be used to approximate the data. To do that, it sets up an "ideal" knot vector for degree p and another one for degree $p-1$. If there are input knots in a flexibility interval, the algorithm selects the one closest to the ideal knot. If no input knot is present in the interval, the ideal knot is used. After curve approximation is achieved, the input knot vector is updated by adding the ideal knots whose flexibility intervals did not contain input knots.

Following is the algorithm to obtain B-spline sectional curve fitted to a gradient flow.

```
Input:
  N : number of flow points;
  P[N] : flow points;
  P_s[N] : scalar values;
  nU : number of control points in U;
  K_in[] : candidate knot vector;
  eps : fitting error tolerance;
  per : knot control (percentage of interval use)
Output:
  C_s(u) : sectional curve;
  K_in[] : updated candidate knot vector;
Algorithm:
  U[N] : parameter of flow points;
  K[] : knot vector;
  deg : degree of curve (=3)
  f_err : fitting error;

  // Chord length parameterization
  for (i=0;i<N;i++) {
    U[i] := Chord_Length_Param(P);
  }
  while(1) {
    // compute knot vector using candidate
    K = compute_knot_with_candidate(U,nU,deg,K_in,per);
    C_s(u) = LeastSquaresFitting(P,U,deg,nU,K,f_err);
    if (f_err > eps) {
       nU := nU + 1;
       continue;
    } else {
      K_in = K;
      done;
    }
  }
```

4 Surface skinning

Let $C_j(u), j = 0, \dots, K$ be a set of section curves defined as

$$C_j(u) = \Sigma_{i=0}^{n} N_{i,p} P_{i,j} \quad j = 0, \dots, K \tag{9}$$

In order to make a skinning surface from these sectional curves, $C_j(u)$ need to meet the three conditions,

- All $C_j(u)$ need to be defined on the same knot vector U.
- The degree of each $C_j(u)$ is the same (3).
- All $C_j(u)$ are defined on the same parameter range.

The second and the third condition are met in the above procedures. In order to make $C_j(u)$ defined on the same knot vector, knot merging method is used. In general, when knots are merged for all $C_j(u)$, the number of control points grows rapidly. If the average number of control points of the cross sections is n, and there are K section curves, then after knot merging the total number of control points can be as high as $O(K^2 n)$, instead of $O(Kn)$ [2]. By adopting the method described in previous section, we can minimize the number of control points since the most of the knot vector of each sectional curves have many common knots across them. Figure 4 and Table 1 show the effect of knot control by varying the parameter *per* from 0.0 to 1.0.

When the $C_j(u)$ are converted such that three conditions are met, we interpolate each control points of $C_j(u)$ along the V parameter direction. The degree in the V direction are set as 3 as default and the parameter $\bar{v}_l, l = 0, \dots, K$ are computed by chord length parameterization and knot vector V are computed by averaging method [16].

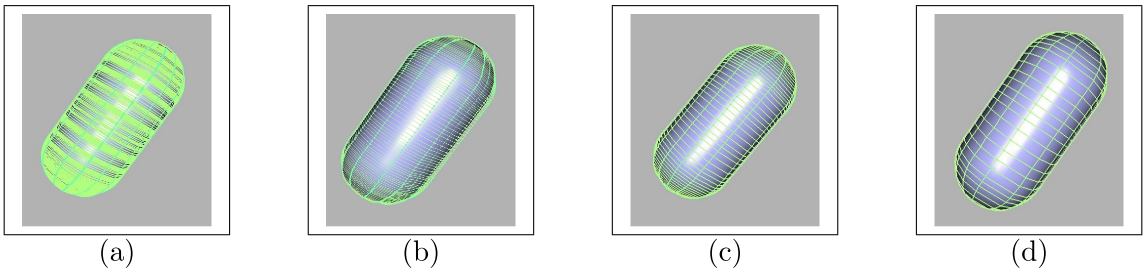


Figure 4: Knot control; *per*: percentage of interval described in [2] (a) $per = 0$ (b) $per = 0.2$ (c) $per = 0.5$ (d) $per = 1.0$

per	0	0.2	0.5	1.0
U	244	89	55	30
V	16	16	16	16
Min fitting err	$4.4e-6$	$3.9e-6$	$3.6e-6$	$1.5e-6$
Max fitting err	$9.9e-4$	$9.6e-4$	$9.7e-4$	$9.7e-4$

Table 1: The number of control points after skinning in V parameter direction by setting per as $0, 0.2, 0.5, 1.0$. When the per is equal 1.0. The number of control points in U parameter is reduced by 1/8 compared with the skinning surface when no candidate knots are applied.

4.1 Some issues of parameterization

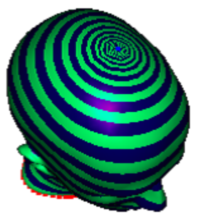
For a generalized clynderical mesh which is smooth and almost symmetric about the axis of the clynder such as the capsule in Figure 4, chord length parameterization usually generate a smooth skinning surface. However, our results show that the parameterization may not produce a good result when the mesh is complicated and not symmetric about the axis. We propose a solution for this issue.

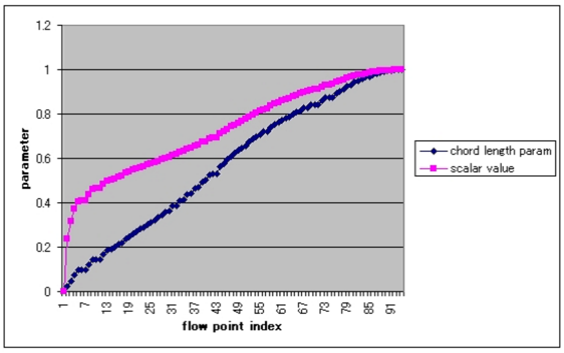
Since the scalar field defined on the mesh is smooth, the parameter value along the sectional curve should reflect the scalar value along the curve. In other words, the parameter values u of different sectional curves should be the same if the scalar values at the corresponding flow points are the same.

The scalar value can not be directly used for parameterization since its characteristic curve is not linear near the extreme points for some meshes as shown in Figure 5 (b), which results in a singular system of equation 6. Figure 5 (a) shows the scalar value near the minimum point.

The proposed method chooses one arbitrary gradient flow as a reference and obtain its chord length parameterization. Then a mapping table Φ from the scalar values to its chord length parameters is created. The other flows obtain its parameter at each flow node by querying into the parameter table Φ. If the scalar value is not exactly on the table Φ, the parameter u is obtained by linearly interpolating nearest two parameters on the table. Let we call this method *guided chord length* parameterization. The figure 6 shows the effect of this guided parameterization for an asymmetric object.

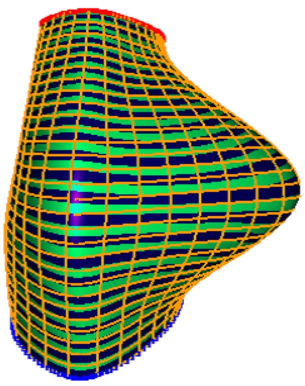
The Figure 7 shows the difference of the skinning surfaces of a human head model for the two different parameterization methods. In chord length parameterization severe wrinkles appear around the nose and ears. In those area, iso-parameter curves are bended

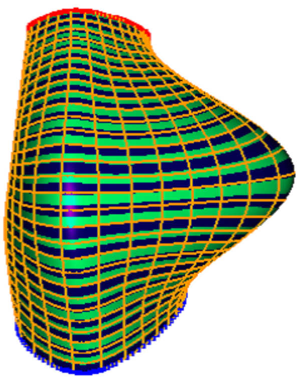




(a) scalar field near the extreme (b) scalar value and chord length parameter

Figure 5: Scalar field and chord length parameterization





(a) chord length parameterization (b) guided chord length parameterization

Figure 6: Two different parameterization for an asymmetric object. (a) Iso-parameter curves in orange are not aligned with the scalar field represented by the texture color. (b) Iso-parameter curves are aligned with the scalar field

in undesired way. These unwanted wrinkles are remedied by the *guided chord length* parameterization. Note that there are still wrinkles near the extrema point. This is the limitation of the proposed method near the extrema without boundaries. Because all the sectional curves meet at one extrema point, some overlapping phenomenon of those curves can not be avoided unless removing too close sectional curves.

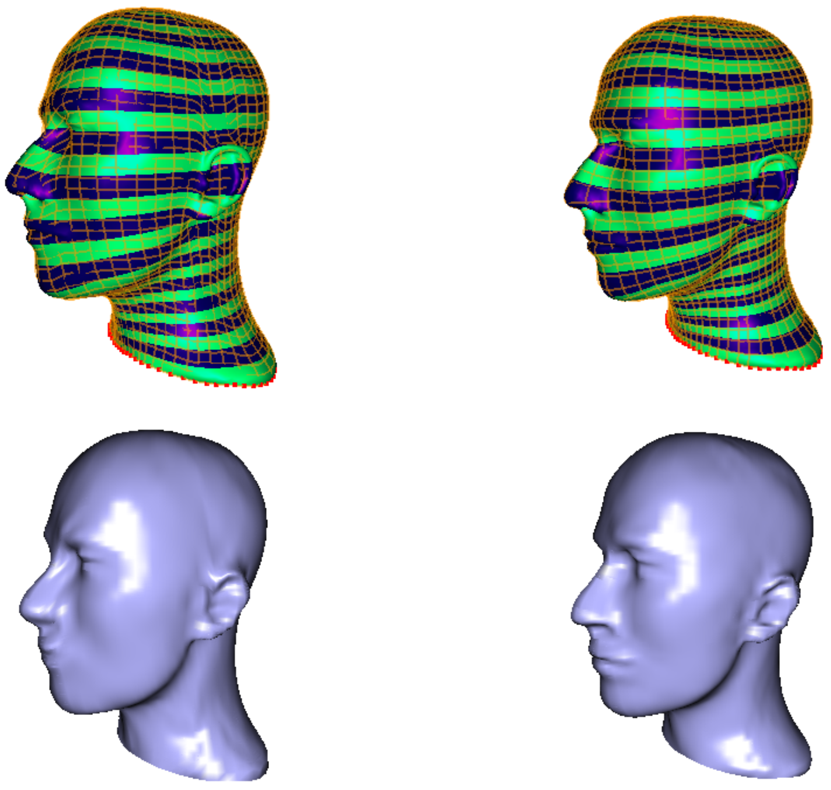

(a) chord length parameterization (b) guided chord length parameterization

Figure 7: Two different parameterizations of a human head. (a) undesired wrinkles appear around the nose and ears. (b) undesired wrinkles are alleviated

5 Results and Discussion

The proposed method is tested with various mesh objects. Maximum and minimum extrema points are selected manually by the user. All the reconstruction steps are done automatically once that the surface fitting criteria, such as the desired number of sectional curves and the fitting error tolerances of the sectional curves and that of the skinning surface are fixed.

Figure 8 shows the results when the generalized cylinder have boundary at both ends and in Figure 9 shows the case with no boundaries or one boundary at either end. In both figures, first row of the figure shows the original polygonal mesh. Maximum extrema points are colored in red and minimum extrema points are colored in blue. Second row of the figure shows the smooth scalar field defined on the mesh. Third row shows the iso-parameter curves of the skinning surface and in fourth row the shading results of the skinning surface are shown.

For moai and head examples in Figure 9, the *guided chord length* parameterization is used, for the other examples chord length parameterization is used. We use the uniform sampling method to select gradient flows for skinning for all examples except for the object 1 which is generated by adaptive sampling.

In Table 2, the geometry of each object and the results of generated B-spline surfaces are described. The minimum and maximum fitting errors are obtained by calculating the distance between the skinning surface and each vertex of the original mesh by using Newton iteration method (see [16]).

Table 3 shows timing results for the examples. Time is measured for each step of the procedures. We use a PC based on Pentium 4(1.60 GHz) with 1.0GB RAM. The generation of gradient flows takes most of the time especially for large complicated meshes. So the spacing between the gradient flows affect the total time quite directly.

object name	glass	object 1	object 2	tube	moai	head
vertex	1004	1500	360	1082	5000	6743
face	1932	2872	672	2160	9996	13424
g-flows	62	72	31	15	98	196
U	113	119	88	74	132	156
V	30	30	15	9	48	60
Min fitting err	$4.9e-6$	$5.7e-6$	$1.3e-5$	$4.6e-7$	$8.8e-9$	$7.3e-8$
Max fitting err	$2.3e-1$	$3.2e-3$	$2.4e-3$	$1.7e-2$	$1.7e-3$	$4.8e-3$

Table 2: The geometry of each test objects and the results of surface fitting. The g-flows is the number of gradient flows generated. U is the number of control points in U direction. V is the number of control points in V direction. Skinning is done in V direction.

object name	glass	object 1	object 2	tube	moai	head
Step 1	2.3s	2.6s	1.1s	1.3s	32.1s	73.9s
Step 2	0.4s	0.8s	0.3s	0.4s	0.9s	11.1s
Step 3	0.7s	0.8s	0.3s	0.1s	1.7s	2.9s

Table 3: Time took for each steps. Step1: the generation of the smooth scalar field then tracing of all gradient flows. Step2: the sorting of gradient flows and sectional curve fitting to the selected flows. Step3: the generation of skinning surface from the sectional curves

To stabilize the proposed method, the source mesh may need to undergo remeshing steps depending on the quality of the mesh before generating gradient flows. Existing remeshing techniques such as decimation [7], smoothing [19] [20], and hole filling [11] [14] can be used. If only the point set is available such as the digitized coordinates from 3D scanners, they need to be converted to a polygonal mesh using existing techniques [1] [4] [21] before being fed into the proposed method.

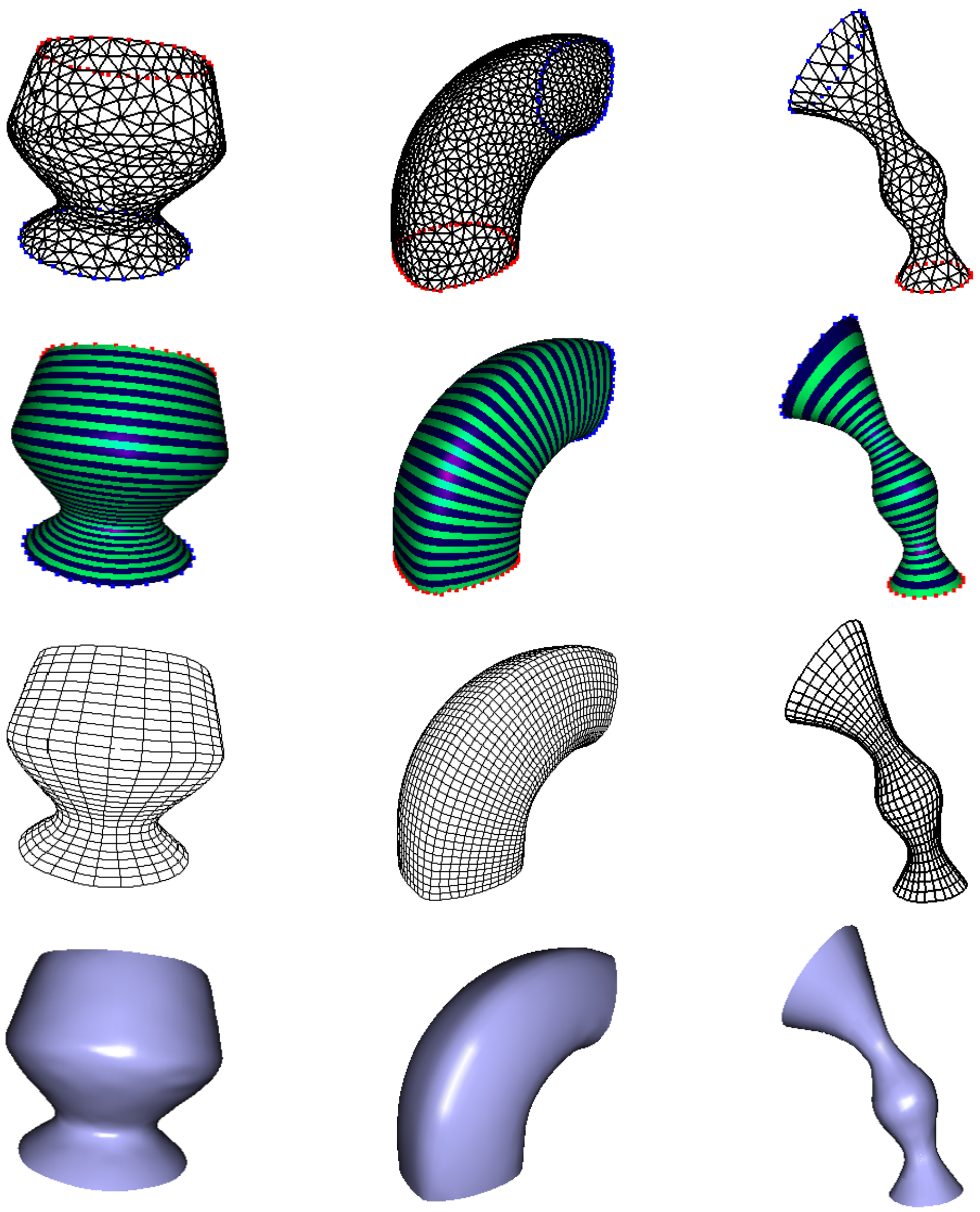


Figure 8: Generalized cylinder objects with boundary at both ends

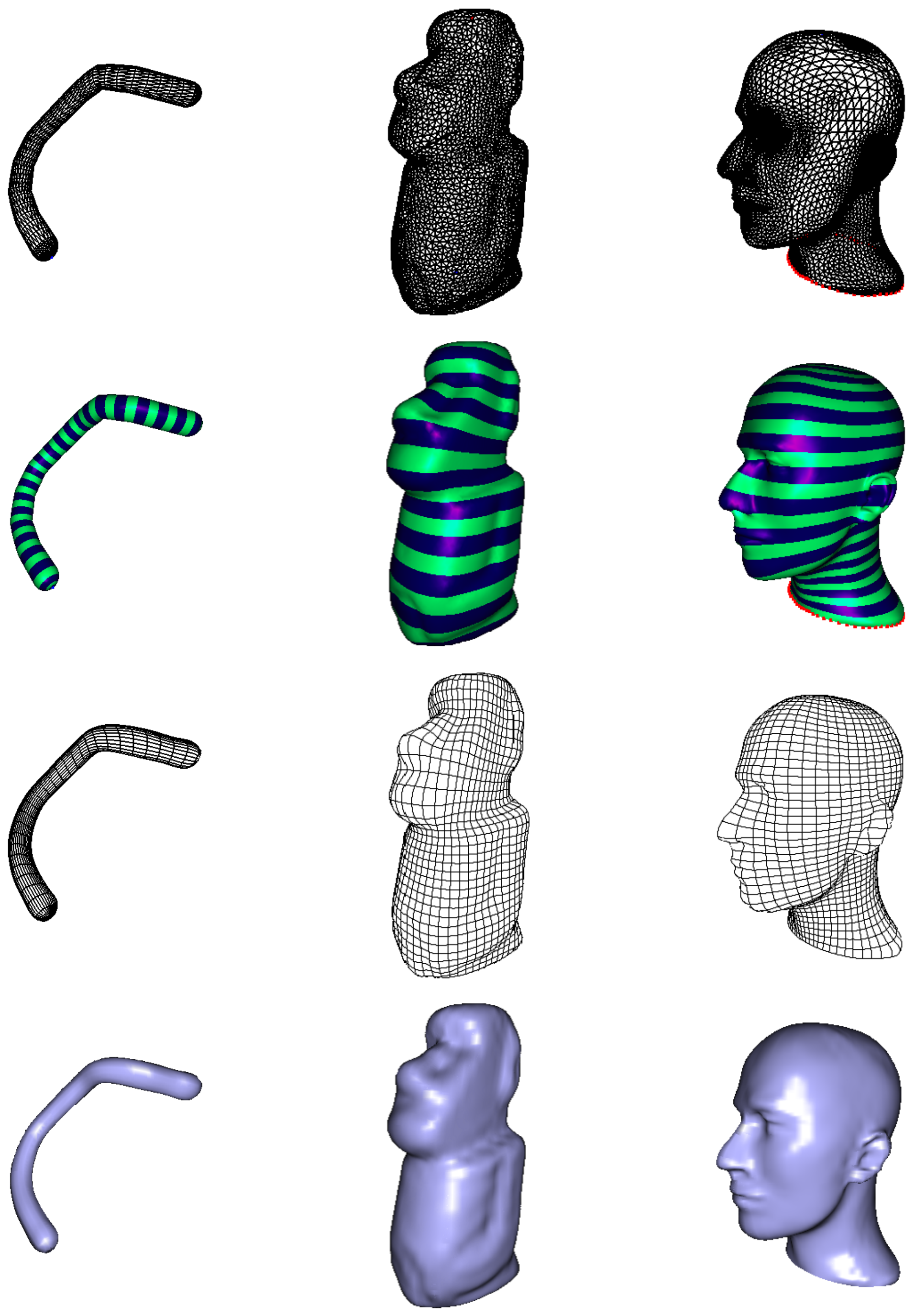


Figure 9: Generalized cylinder objects with no boundaries or with one boundary at either end

6 Conclusion and future works

Surface modeling by skinning is an often used modeling technique. In CAD, modeling from existing objects is called reverse engineering and the skinning is one of the techniques. There are many papers which deal with this topic but there are few discussing how the desired sectional curves can be defined and created [3] . Although the proposed skinning method is limited to the generalized cylinder mesh object, many real-world objects can be reconstructed using this techniques from very simple to more complicated objects as shown in our test cases. The effectiveness of the proposed method is demonstrated by converting various generalized cylinder meshes with or without boundaries to smooth B-spline surfaces. Object format conversion especially from a polygonal mesh to a spline surface is an important modeling methodology. As the results indicate, the proposed method is efficient in time and requires minimum user interventions. The main contribution of this paper is a novel method of skinning from a polygonal mesh especially the data preparation of sectional curves.

There are some possible future works. The first is the optimal generation of the sectional curves for skinning. Although the adaptive sampling is one technique, it might miss important features such as the ridge lines of the surface. The other direction is to adopt a hierarchical method proposed by [9] [6] to our skinning method. As the test results indicate, if the original mesh is large and complicated, the time to generate the gradient flows can not be ignored, so more optimized algorithms to speed up the process are required.

References

[1] A.Hilton and A.J. Stoddartand J.Illingworth. Reliable surface reconstruction from multiple range images. *Computer Vision ECCV*, 1064:117–126, 1996.

[2] Les A.Piegl and Wayne Tiller. Surface approximation to scanned data. *Visual Computer*, 16(4):386–395, 2000.

[3] Les A.Piegl and Wayne Tiller. Surface skinning revisited. *Visual Computer*, 18(4):273–283, 2002.

[4] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of 23rd conference on Computer graphics and interactive techniques*, 1996.

[5] Mathieu Desbrun, Mark Meyer, Peter Schrder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Proceedings of Visualization and Mathematics*, pages 35–57. Springer-Verlag, 2002.

[6] D.Forsey and R.Bartles. Surface fitting with hierarchical splines. *ACM Transaction on Graphics*, 14(2):134–161, 1995.

[7] C Gotsman, S Gumhold, and L Kobbelt. Simplification and compression of 3d-meshes. In *Proceedings of the European Summer School on Principles of Multiresolution in Geometric Modelling*, pages 319–361. Springer, 2002.

[8] Venkat Krishnamurthy and Mark Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM Press, 1996.

[9] S Lee, G Wolberg, and S Y Shin. Scattered data interpolation with multilevel b-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):228–244, 1997.

[10] Bruno Levy. Constrained texture mapping for polygonal meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, volume 16, pages 12–17, 2001.

[11] Peter Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, 2003.

[12] Weiyin Ma and Periren He. B-spline surface local updating with unorganized points. *Computer-Aided Design*, 30(11):853–862, 1998.

[13] Weiyin Ma and J P Kruth. Parameterization of randomly measured points for least squares fitting of b-spline curves and surfaces. *Computer-Aided Design*, 27:663–675, 1995.

[14] F S Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. Technical report, Georgia Institute of Technology, 1999.

[15] L.A. Piegl and W. Tiller. Parameterization for surface fitting in reverse engineering. *Computer-Aided Design*, 33(9):593–603, 2001.

[16] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer, 1997.

[17] R.Kimmel and J.A.Sethian. Computing geodesic paths on manifolds. In *Proceedings of Natl. Aca. Sci. USA*, pages 8431–8435. Applied Mathmatics, 1998.

[18] S.Dong and S.Kircher. Harmonic functions for quadrilateral remeshing of arbitray manifolds. *Computer-Aided Geometric Design*, 22:392–423, 2005.

[19] G. Taubin. Geometric signal processing on polygonal meshes. EUROGRAPHICS f2000 STAR, 2000.

[20] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceeding of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351 – 358, 1995.

[21] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of 24rd conference on Computer graphics and interactive techniques*. ACM, 1994.

[22] W.Ma and J.P.Kruth. Nurbs curve and surface fitting for reverse engineering. *Advanced Manufacturing Technology*, 14(4):918–927, 1998.