Low Distortion Shell Map Generation

Kai Ye* Zhejiang University
Kun Zhou† Microsoft Research Asia
Zhigeng Pan‡ Zhejiang University
Yiying Tong§ Caltech
Baining Guo¶ Microsoft Research Asia

ABSTRACT

A shell map [7] is a bijective mapping between shell space (the space between a base surface and its offset) and texture space. It can be used to generate small-scale features on surfaces using a variety of modeling techniques. In this paper, we present an efficient algorithm, which reduces distortion by construction, for the offset surface generation of triangular meshes. The basic idea is to independently offset each triangle of the base mesh, and then stitch them up by solving a Poisson equation. We then introduce the details for computation of a stretch metric, which measures the distortion of shell maps. Our results show a substantial improvement compared to previous results.

Keywords: Shell Maps, Offset Surface, Volumetric Texture, Geometric Texture Mapping.

1 INTRODUCTION

Geometric details are often used in computer graphics to enhance the visual richness of a 3D surface. In these types of graphical representations, a base surface efficiently models the basic shape of an object, while volumetric texture or other types of geometric details are embedded in a thin layer of three dimensional space above the base surface, called a *shell space* (see Figure 1). The shell space is often constructed as the space between the base surface and an offset surface, created by moving each point along the normal direction of the original surface by a constant (or variable) distance.

Mapping 3D geometric textures onto a base surface requires a 3D parameterization between the shell space and the texture space. Such mappings have been implicitly used since the late 80's [4, 5, 1]. Recently, Porumbescu et al. [7] introduced a bijective mapping between shell space and texture space, called a *shell map*. Given a base surface S, an offset surface S_o that has the same structure as S is generated using the method of Cohen *et al.* [2]. Utilizing the identical structures of S and S_o, the shell space is tiled with prisms (see Figure 2), each of which has a corresponding prism in the texture space. Splitting these prisms into tetrahedra, they establish direct correspondences between tetrahedra in the shell space and tetrahedra in the texture space. The shell map is then defined using the barycentric coordinates of the corresponding tetrahedra. A shell map supports any types of objects that can be placed in texture space, such as geometric objects, procedural volume textures, scalar height fields, or other objects. The mapping is bijective, allowing the use of both feed-forward rendering applications [10] and ray-tracing applications.

In this paper, we study the distortion of shell maps, which is inevitable like that of the planar parameterization of arbitrary surfaces.

*e-mail: kelvinye@cad.zju.edu.cn
†e-mail: kunzhou@microsoft.com
‡e-mail: zgpan@cad.zju.edu.cn (corresponding author)
§e-mail: yiying@cs.caltech.edu
¶e-mail: bainguo@microsoft.com

1.1 Related Work

Shell map distortion comes from its two main building blocks: offset surface generation and parameterization of the base/offset surface. While a variety of tools have been developed to parameterize arbitrary surfaces (see the survey in [3]), only a few methods have been proposed to generate the offset surface in computer graphics.

In [2], Cohen *et al.* proposed to move the vertices along their normals by a given distance in positive and negative directions to construct the *simplification envelope* to control the global error in level of detail approximation. They introduced an iterative way of moving each vertex with adaptive step sizes along the normal in order to deal with self-intersection.

Peng *et al.* [6] proposed the use of an *extended distance function gradient* to move the surface without creating intersections. Compared with this method, the thickness of envelopes in [2] is very low in regions of concavities, and the shape of the surface of the envelope tends to be undesirable in such areas.

Zhou *et al.* [11] adopted the idea of tetrahedralizing the shell space presented in Porumbescu *et al.* [7]. They added an optimization of the shell mapping based on a stretch metric, since low distortion is crucial in their synthesis of geometric texture. They kept the shell space and the parameterization of the base mesh as is, only changing the parameterization of the offset mesh to reduce the stretch of the shell map. As a result, they need to store two separate sets of texture coordinates: one for the base mesh, and the other for the offset mesh. Moreover, as we will demonstrate, fixing the height of each vertex of the offset surface in the texture space is quite restrictive.

1.2 Overview

Despite a few attempts to generate uniform offset surfaces, it remains an open problem to compute an optimal offset surface which reduces the geometric distortion of the shell map. We propose a simple method that combines construction of the offset surface *and* minimization of distortion. To provide additional degrees of freedom in order to allow for low-distortion maps, we let the offset distance fluctuate over the surface (i.e., we do not restrict the location to be exactly at a given distance to the original mesh). Our results confirm the necessity of introducing these *vertical* degrees of freedom. Furthermore, our technique is efficient in both computational time and memory storage.

The basic idea of our approach is to treat the shell space as a set of prisms, each of which is the space between a triangle of the base mesh and its offset. We first allow the prisms to be detached so that they are all right triangular prisms, and then assemble them using linear least squares optimization. In our case, the gradient fields are stored on tetrahedra, thus, we use the three dimensional discrete differential operators and the Poisson equation developed in Tong *et al.* [9].

Later in this paper, we discuss the stretch metric as proposed in Zhou *et al.* [11] in great detail, as it is crucial for comparison of the quality of our results and that of the previous methods. We also show that optimization guided by this measure can help to lower the distortion of the shell space produced by other methods as well. Nevertheless, our results *without* additional optimization already show a substantial improvement even when compared to the previous results *with* this optimization.

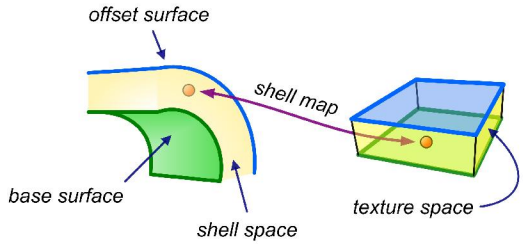


Figure 1: The shell space is the region between a base surface and its offset surface. A shell map is a one-to-one function between texture space and shell space [Porumbescu et al. 2005].

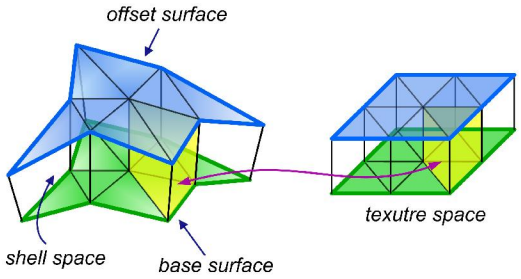


Figure 2: Prisms in shell space correspond to prisms in texture space [Porumbescu et al. 2005].

2 Poisson Offset Surface Generation

In this section, we introduce the method for the construction of the low distortion offset surface simply by solving a Poisson equation on a thin layer upon the tetrahedral mesh.

2.1 Setup of the Problem

Given a base surface S, our goal is to generate a low distortion offset surface S' with the following properties, similar to those considered in [7]:

- S' must have the same connectivity as the base surface.
- The mapping from the texture space (a thin layer of rectangular space containing volumetric texture, geometric texture, etc.) to the shell space (the layer between S' and S) should be of low distortion.

A triangle of the base surface and its corresponding triangle of the offset surface form a prism P (see Figure 2). Each prism P in the shell space has a corresponding right triangular prism P_t in texture space. In order to get a low distortion mapping, the prisms in the shell space should be similar to the right triangular prisms.

2.2 Decompose and Assemble

First, we move each triangle along its normal by a specified distance. Thus, the shell space is composed of right triangular prisms, but they are disjoint (see Figure 3(b)), *i.e.*, the location of points in the adjacent triangles on the offset surface are discontinuous on the common edges. Therefore, we take the gradients of the coordinates of the points in the right triangular prisms as guidance vector fields to be used in a Poisson equation to solve for the position of each vertex on the offset surface. Since the gradients in a tetrahedron can be computed more easily than in a triangular prism, we split each prism into three tetrahedra as done in [7].

Our method is based on the discrete Poisson equation, which can be expressed as

$$Div(\nabla f) = Div(\mathbf{w}) \tag{1}$$

where f is the unknown scalar field function, and $\mathbf{w}$ is a guidance vector field.

The definition of the scalar field f in the domain considered (i.e., the shell space) is piecewise linear:

$$f(x) = \sum_i f_i \phi_i(x) \tag{2}$$

where f_i is a coordinate of a vertex v_i on either the base surface or the offset surface, and ϕ_i is a piecewise linear basis function valued 1 at the vertex v_i and 0 at all other vertices.

The discrete gradient of the shell scalar function f is evaluated as

$$\nabla f(x) = \sum_i f_i \nabla \phi_i(x) \tag{3}$$

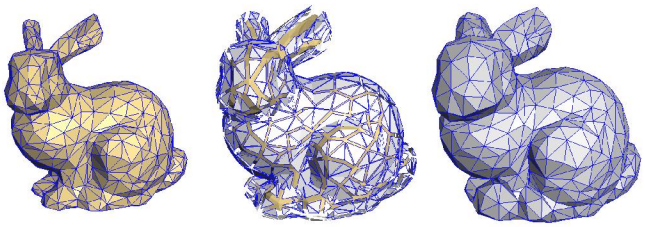


Figure 3: (a) Base mesh. (b) Each triangle is moved along its normal, and the shell space is composed of disjoint right triangular prisms. (c) Our algorithm assembles the prisms to obtain the offset surface.

The discrete divergence of $\mathbf{w}$ at vertex v_i is defined as

$$(Div\,\mathbf{w})(v_i) = \sum_{T_k \in N(i)} \nabla \phi_{ik} \cdot \mathbf{w}(T_k) |T_k| \tag{4}$$

where ϕ_{ik} is the function ϕ_i restricted to a tetrahedron T_k in the disjoint shell space, $|T_k|$ is the volume of the tetrahedron T_k, and $\mathbf{w}(T_k) = \sum_i f'_{ik} \nabla \phi_i(x)$ is the guidance vector field precomputed using the right triangular prism, with f'_{ik} being the coordinate corresponding to v_i seen from the tetrahedron T_k of the disjoint shell space.

Finally, we conclude as follows:

$$\begin{aligned} &Div(\nabla f) = Div(\mathbf{w}) \Leftrightarrow \\ &\sum_{T_k \in N(i)} \nabla \phi_{ik} \cdot (\sum_j f_j \nabla \phi_j(x)) |T_k| \\ &= \sum_{T_k \in N(i)} \nabla \phi_{ik} \cdot \mathbf{w}(T_k) |T_k| \end{aligned} \tag{5}$$

2.3 Final Linear System

With the boundary conditions specified, Equation (5) written for each vertex of the coordinates of the base mesh leads to a large, sparse linear system

$$\mathbf{A}\mathbf{f} = \mathbf{b} \tag{6}$$

where $\mathbf{A}$ is a matrix representing the Laplacian operator, $\mathbf{f}$ is f_i assembled in a vector, and $\mathbf{b}$ is the divergence of $\mathbf{w}$.

The solution to Equation (6) is

$$\mathbf{f} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \tag{7}$$

Since $v_i (i = 1 \ldots n)$ are the base surface vertices, we assign the base surface vertices coordinates to f_i. $v_i (i = n+1 \ldots 2n)$ are the offset surface vertices, and we denote their coordinates also by f_i, which are the unknowns. Thus, we can obtain them through the linear system above ($\mathbf{A}$ is a $2n \times n$ matrix, and b is a vector with $2n$ components). The system is solved three times to get x, y, and z coordinates for each vertex on the offset surface.

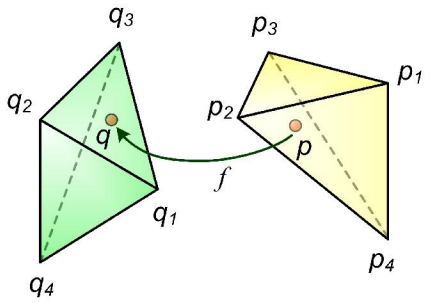


Figure 4: The unique affine mapping between two tetrahedra.

2.4 Extension to Multi-Layer Shell Space

When additional memory storage is allowed, the linear system above can be *directly* extended to the construction of a multi-layer shell space. In that case, each triangle in the base mesh creates a series of (k) right prisms, vertically piled up, which are then used to compute the guidance field in each tetrahedron in the shell space. The exact same method can be applied, leading to a final system with now kn unknowns.

2.5 Preventing Self-Intersection

The offset surface solved from Equation (7) may still contain self-intersections. For applications with overlapping (e.g., fur or hair) or self-intersecting structures, a bijective mapping is not always required. For most other applications, preventing self-intersection of the offset surface is necessary to create a bijective mapping. We use the technique from [2] to avoid self intersection in our algorithm, and the following is the basic idea of [2].

First, an attempt is made to displace each triangle the full offset distance specified by the user along the normal direction. Then the offset surface is solved using Equation (7). Using an octree data structure, the algorithm efficiently detects whether each triangle displacement leads to self-intersection. If it does, the algorithm rejects the displacement and begins an adaptive stepping procedure at a fraction of the user-specified offset distance. This procedure terminates if no triangles can be moved. Thus, most of the vertices reach the full offset distance in the end, while the ones in regions of big concavity go as far as possible.

3 Shell Texture Space Optimization

Although we have generated a low distortion offset surface, the distortion cannot be entirely eliminated. Thus, we propose using the Tetrahedron Stretch Metric [11] to measure the distortion in the mapping between the shell space and the texture space. Moreover, we may use the same metric to further optimize the texture space.

3.1 Tetrahedron Stretch Metric

Given a tetrahedron T with its coordinates q_1,q_2,q_3,q_4, and the corresponding texture coordinates p_1,p_2,p_3,p_4, we use barycentric coordinates to define the unique affine mapping $f(p)=f(u,v,w)=q$ as follows:

$$f(p)=\frac{\begin{pmatrix}<p,p_2,p_4,p_3>q_1+<p,p_3,p_4,p_1>q_2\\+<p,p_4,p_2,p_1>q_3+<p,p_1,p_2,p_3>q_4\end{pmatrix}}{<p_1,p_2,p_3,p_4>} \tag{8}$$

where $<a,b,c,d>$ denotes the volume of the tetrahedron $abcd$.

The Jacobian of the mapping function f is $[f_u,f_v,f_w]$. Since the mapping is affine, partial derivatives of f are all constant. They can be straightforwardly computed—see Appendix A for detailed formulas.

We denote the three singular values of the Jacobian $J=[f_u,f_v,f_w]$ by Γ_1,Γ_2, and Γ_3 respectively. Geometrically, they represent the principal stretches, and their squares are the eigenvalues of the Cauchy deformation tensor J^tJ. Thus, we can express the L^2 stretch norm over a tetrahedron T as

$$\begin{aligned}L^2(T)&=\sqrt{\frac{1}{3}(\Gamma_1^2+\Gamma_2^2+\Gamma_3^2)}\\&=\sqrt{\frac{1}{3}(f_u\cdot f_u+f_v\cdot f_v+f_w\cdot f_w)}\end{aligned} \tag{9}$$

The overall norm for the entire mesh shell space $M=\{T_i\}$ can be given as

$$L^2(M)=\sqrt{\sum_{T_i\in M}(L^2(T_i))^2V(T_i)/\sum_{T_i\in M}V(T_i)} \tag{10}$$

where $V(T_i)$ is the volume of the shell tetrahedron T_i. The L^2 norm measures the overall distortion of the whole map between the shell space and the texture space. We normalize the stretch values in order to make the lower bound of the L^2 norm 1.0 for any parameterization by multiplying it with the factor

$$\sqrt[3]{\sum_{T_i\in M}V'(T_i)/\sum_{T_i\in M}V(T_i)}$$

where $V'(T_i)$ is the volume of the texture space tetrahedron T_i.

3.2 Texture Space Optimization

We now consider the problem of minimizing the total L^2 norm of the whole shell texture map. To this purpose, we use an optimization algorithm based on Sander *et al.* [8] to optimize the texture space.

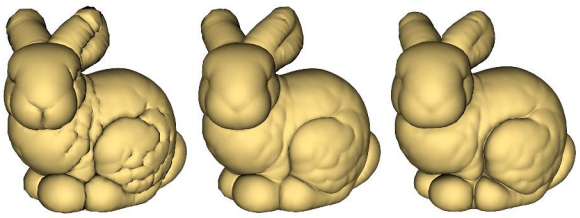
First, we obtain an initial parameterization using the *Iso-charts* technique [12] for the base surface, which provides a low distortion multi-chart parameterization. After that, the texture space is constructed by the initial parameterization, and composed of right triangular prisms. To apply our tetrahedron stretch metric, we split each prism into tetrahedra both in the texture space and in the shell space in a consistent way using the method described in [7]. Then we perform several optimization iterations for each chart of the texture space.

To preserve the parameterization of the base mesh, we only adjust the upper level vertices in the texture space, which correspond to the offset surface vertices in the shell space. For each iteration, we sort these vertices by their L^2 stretches in decreasing order. Then we process each vertex in turn. For each vertex, we randomly choose several directions, and perform binary search along the chosen direction within a specified distance. Each vertex can only move on the upper plane of the texture space, and the boundary vertices of each chart are fixed to keep the mapping free of foldovers.

4 Experimental Results

We have implemented the algorithm described above on a 3.2GHz Pentium 4 workstation with 1GB memory. We have also implemented two previous methods (Cohen *et al.* [2]; Peng *et al.* [6]) for comparison.

Figure 5 shows the offset surfaces for the bunny model. The offset height is set to 10% of the diagonal length of the model's bounding box. From the values of the L^2 stretch of the shell map, our algorithm achieves the lowest distortion. We also calculate the average distance between the offset surface and base surface. For each vertex in the offset/base surface, its shortest distance to the base/offset surface is computed. Then the average distance can be

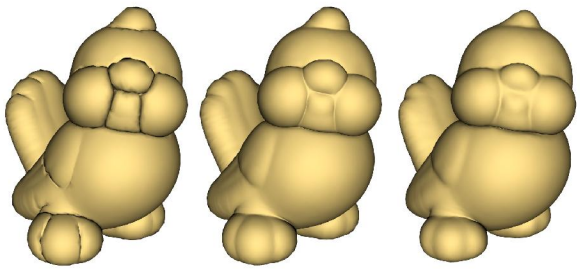


[Cohen et al. 1996] [Peng et al. 2004] our algorithm

L^2 = 1.5384 L^2 = 1.3686 L^2 = 1.2359

Avg Dist: 34.46374 Avg Dist: 37.36883 Avg Dist: 38.55600

Figure 5: Comparison of offset surfaces generated by three different methods.

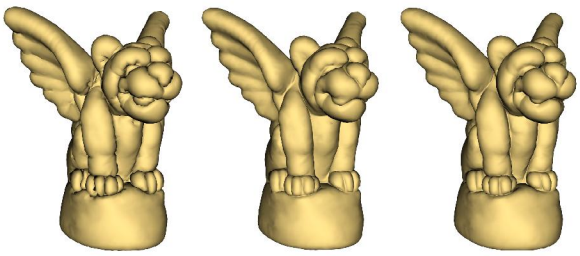


[Cohen et al. 1996] [Peng et al. 2004] our algorithm

L^2 = 1.5328 L^2 = 1.3868 L^2 = 1.1792

Avg Dist: 34.09851 Avg Dist: 35.61834 Avg Dist: 35.64700

Figure 6: Comparison of offset surfaces generated by three different methods.



[Cohen et al. 1996] [Peng et al. 2004] our algorithm

L^2 = 1.5934 L^2 = 1.3689 L^2 = 1.2601

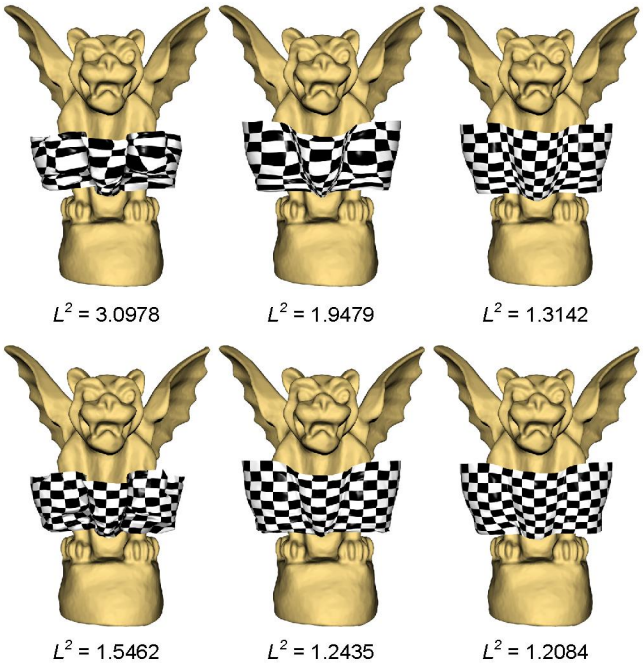
Avg Dist: 15.07453 Avg Dist: 15.19287 Avg Dist: 15.81458

Figure 7: Comparison of offset surfaces generated by three different methods.

calculated as the average value of all these distances. Not surprisingly, the average distances from the three algorithms are almost the same. It takes our algorithm around 5 seconds to compute the offset surface for the bunny model with 10k vertices.

Figures 6 and 7 show the offset surfaces for the tweety and gargoyle models respectively. As demonstrated by these examples, the thickness of the shell space generated by [2] is very small in the regions of concavities, since each vertex is moved strictly along its normal direction independently. On the other hand, [6] moves each vertex along the gradient direction of a global distance function. Therefore it produces a smoother offset surface. Our algorithm can produce not only a smoother offset surface but also a lower distortion shell map.

Figure 8 shows the offset surface distortion of the shell map using a regular checkerboard pattern, while Figure 9 shows a cut in the shell space for inspection of the inner distortion. In these figures, we also show both the stretches before and after shell texture space optimization for all three methods. The offset surface from our method before optimization has already obtained a perceivably lower distortion shell map both by visual comparison and by the L^2 stretch. After using the texture space optimization, the results from [2] and [6] are also improved.

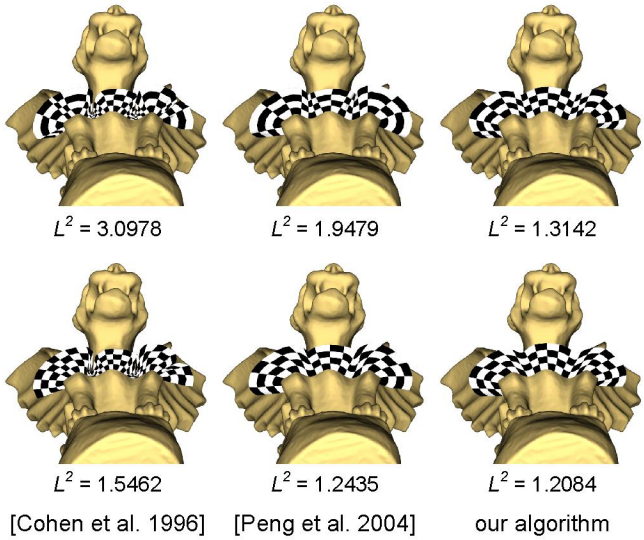


L^2 = 3.0978 L^2 = 1.9479 L^2 = 1.3142

L^2 = 1.5462 L^2 = 1.2435 L^2 = 1.2084

[Cohen et al. 1996] [Peng et al. 2004] our algorithm

Figure 8: The offset surface distortion of the shell map.



L^2 = 3.0978 L^2 = 1.9479 L^2 = 1.3142

L^2 = 1.5462 L^2 = 1.2435 L^2 = 1.2084

[Cohen et al. 1996] [Peng et al. 2004] our algorithm

Figure 9: The inner distortion of the shell map.

Mapping geometric textures onto a surface also clearly illustrates the advantage of our low distortion offset surface and the texture space optimization. In Figures 10 and 11, we use the bunny and the gargoyle as the base meshes, and map the lion and feline meshes from the texture space to the shell space. The objects mapped using the shell maps of [2] and [6] are severely distorted.

5 Conclusion

In this paper, we present an efficient triangular mesh offset algorithm, which reduces the distortion of the corresponding shell map. The basic idea is to stitch up independently offset triangles of the base mesh by solving a Poisson equation. A substantial improvement to the quality of shell maps can be achieved, as demonstrated by our results. We believe that our low-distortion shell mapping should be valuable in a large number of applications.

Acknowledgements

This work was supported by Key NSFC project on Digital Olympic Museum (grant no: 60533080), and 863 high-tech project (no: 2006AA01Z303)

References

[1] Y. Chen, X. Tong, J. Wang, S. Lin, B. Guo, and H.-Y. Shum. Shell texture functions. *ACM Trans. Graph.*, 23(3):343–353, 2004.

[2] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. V. Wright. Simplification envelopes. In *Proceedings of SIGGRAPH 1996*, pages 119 – 128, August 1996.

[3] M. Floater and K. Hormann. Recent advances in surface parameterization. *Multiresolution in Geometric Modelling Workshop*, 2003.

[4] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *Proceedings of SIGGRAPH 89*, pages 271–280, 1989.

[5] F. Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, 1998.

[6] J. Peng, D. Kristjansson, and D. Zorin. Interactive modeling of topologically complex geometric detail. *ACM Trans. on Graphics*, 23(3):635–643, Aug. 2004.

[7] S. D. Porumbescu, B. Budge, L. Feng, and K. I. Joy. Shell maps. *ACM Trans. on Graphics*, 23(3):626–633, Aug. 2005.

[8] P. Sander, J. Snyder, S. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *Proceedings of SIGGRAPH 2001*, pages 409–416, 2001.

[9] Y. Tong, S. Lombeyda, A. N. Hirani, and M. Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graph.*, 22(3):445–452, 2003.

[10] X. Wang, X. Tong, S. Lin, S.-M. Hu, B. Guo, and H.-Y. Shum. Generalized displacement maps. In *Rendering Techniques 2004*, pages 227–234, 2004.

[11] K. Zhou, X. Huang, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum. Mesh quilting for geometric texture synthesis. In *Proceedings of SIGGRAPH 2006*, 2006.

[12] K. Zhou, J. Snyder, B. Guo, and H.-Y. Shum. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Proceedings of ACM Symposium on Geometry processing*, pages 47–56, 2004.

A Jacobian of Shell Map

$$f_u = \frac{\partial f}{\partial u} = \frac{\begin{pmatrix} q_1(y_2z_4 - y_2z_3 - y_4z_2 \\ +y_4z_3 + y_3z_2 - y_3z_4) \\ +q_2(y_3z_4 - y_3z_1 - y_4z_3 \\ +y_4z_1 + y_1z_3 - y_1z_4) \\ +q_3(y_4z_2 - y_4z_1 - y_2z_4 \\ +y_2z_1 + y_1z_4 - y_1z_2) \\ +q_4(y_1z_2 - y_1z_3 - y_2z_1 \\ +y_2z_3 + y_3z_1 - y_3z_2) \end{pmatrix}}{6V}$$

$$f_v = \frac{\partial f}{\partial v} = \frac{\begin{pmatrix} q_1(-x_2z_4 + x_2z_3 + x_4z_2 \\ -x_4z_3 - x_3z_2 + x_3z_4) \\ +q_2(-x_3z_4 + x_3z_1 + x_4z_3 \\ -x_4z_1 - x_1z_3 + x_1z_4) \\ +q_3(-x_4z_2 + x_4z_1 + x_2z_4 \\ -x_2z_1 - x_1z_4 + x_1z_2) \\ +q_4(-x_1z_2 + x_1z_3 + x_2z_1 \\ -x_2z_3 - x_3z_1 + x_3z_2) \end{pmatrix}}{6V}$$

$$f_w = \frac{\partial f}{\partial w} = \frac{\begin{pmatrix} q_1(x_2y_4 - x_2y_3 - x_4y_2 \\ +x_4y_3 + x_3y_2 - x_3y_4) \\ +q_2(x_3y_4 - x_3y_1 - x_4y_3 \\ +x_4y_1 + x_1y_3 - x_1y_4) \\ +q_3(x_4y_2 - x_4y_1 - x_2y_4 \\ +x_2y_1 + x_1y_4 - x_1y_2) \\ +q_4(x_1y_2 - x_1y_3 - x_2y_1 \\ +x_2y_3 + x_3y_1 - x_3y_2) \end{pmatrix}}{6V}$$

where (u,v,w), (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3), and (x_4,y_4,z_4) are the coordinates of p, p_1, p_2, p_3, and p_4 respectively; and V is the volume $< p_1,p_2,p_3,p_4 >$:

$$V =< p_1,p_2,p_3,p_4 >= \frac{\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}}{6}$$

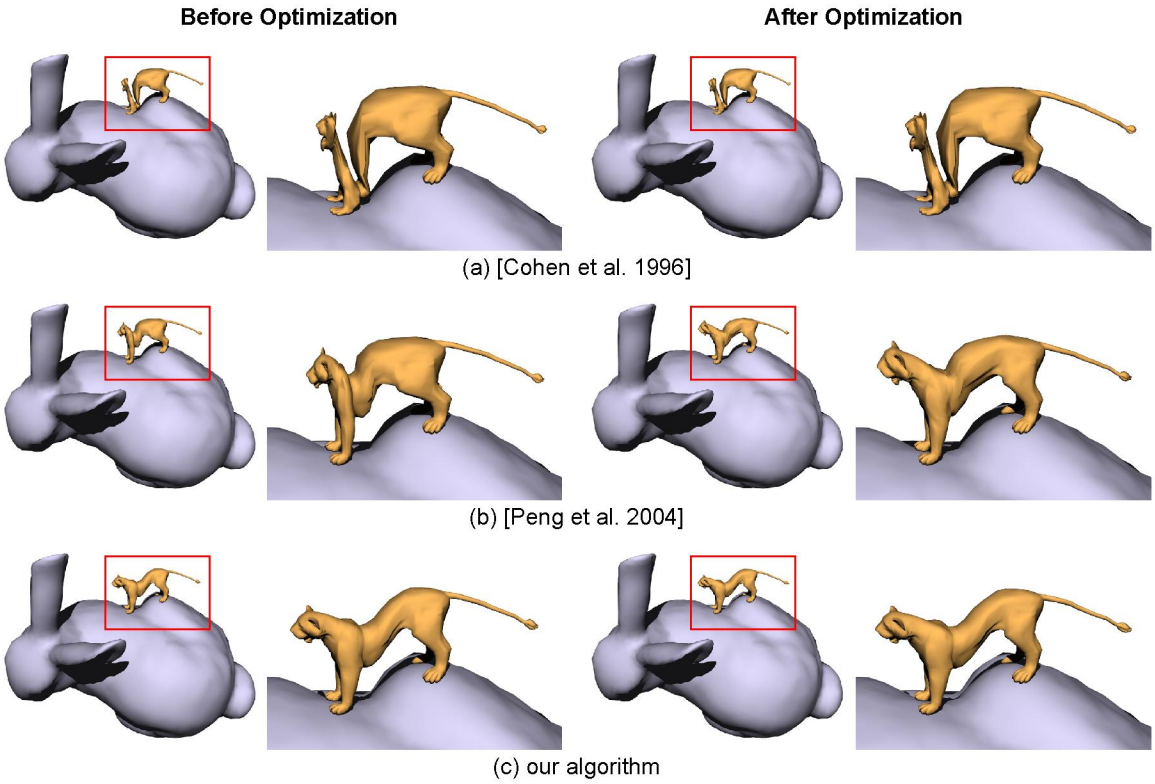


Figure 10: Mapping a Lion mesh onto a Bunny surface.

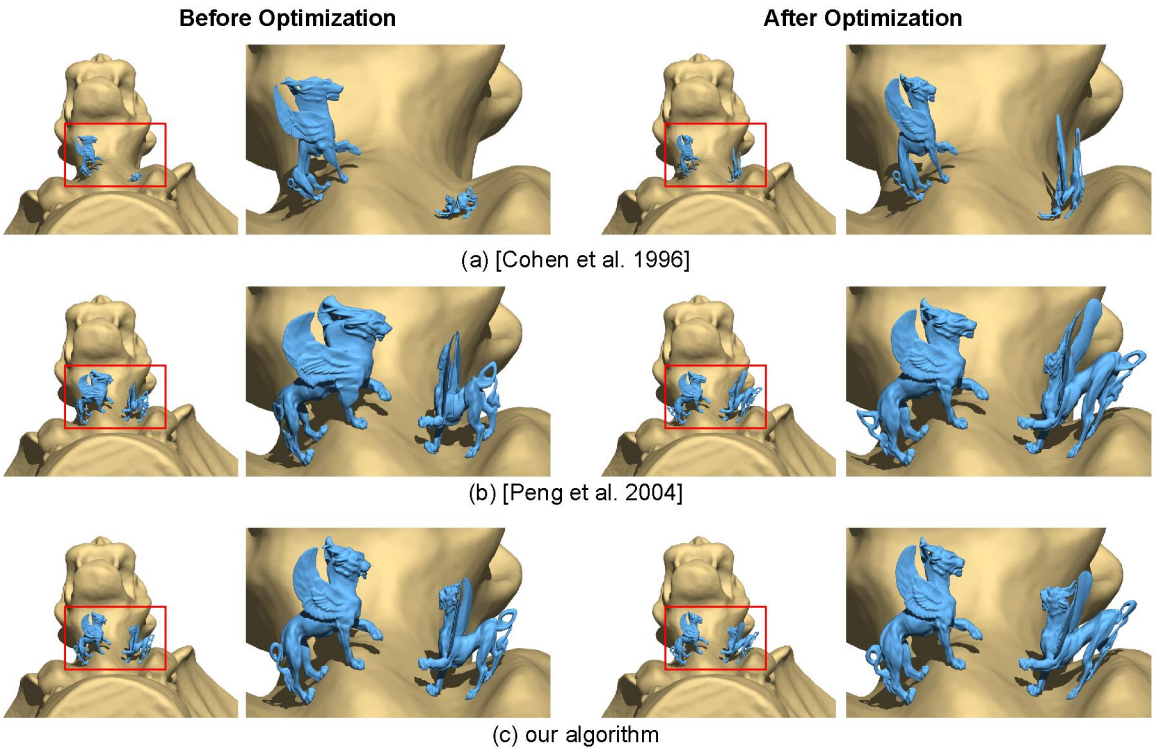


Figure 11: Mapping two Feline meshes onto a Gargoyle surface.