Real-time Dynamics for Geometric Textures in Shell

Jin Huang[†][*] Hanqiu Sun[‡], Kun Zhou[†] and Hujun Bao[†]
[†]State Key Lab of CAD&CG, Zhejiang University, China
[‡]The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong

Abstract

Embedding geometric textures in a shell space around an arbitrary surface has been a popular way to add highly detailed geometric details and enhance visual richness in graphics community, but the dynamic effects of geometric textures have not been modeled and simulated. In this paper, we introduce an efficient algorithm for deforming geometric textures with dynamic effects. The algorithm consists of two steps. First, it computes a deformed shell space by optimizing a material related energy function, which is then used to evaluate the equilibrium position of the geometric texture. Second, an explicit time integration scheme is applied for vibrating the geometric texture around its equilibrium position. Users can deform the geometric textures by dragging its vertices directly, and the dynamic behavior of the geometric textures can be changed by adjusting several material parameters. The dynamic simulation of geometric textures can be easily implemented on GPU and runs at real-time rates.

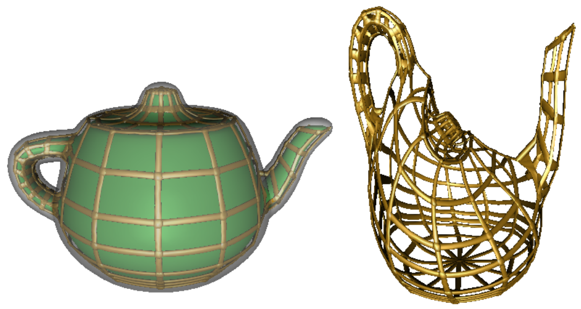


Figure 1: *Dynamic effects of geometric textures can be simulated efficiently by our two-step scheme. In the left, we render the base mesh as a green solid, and the offset mesh as a translucent surface. The right figure shows one frame of the dynamic deformation of the geometric textures which are embedded between them.*

1 Introduction

Embedding geometric textures in a shell space around an arbitrary surface has been used in computer graphics as a way to enhance the visual richness of a 3D surface [Porumbescu et al. 2005; Zhou et al. 2006]. But the dynamic effects have never been modeled for the embedded geometry details when the mesh deforms. In this paper, we present an efficient method to simulate the dynamics of geometric textures with alterable material properties.

Considering the complex structure and huge number of vertices of the embedded geometry details, directly simulating the effects by physically dynamic model is not applicable. Generating the shell at each frame of mesh deformation can hardly take the material into account. And simply embedding the geometry details in the shell will lose high frequency vibrations.

To overcome the above problems, we propose a novel two-step scheme for this application. Given a sequence of mesh deformation which is supplied by arbitrary methods, we setup a material related energy function which is easy to be optimized to measure the distortion of the shell, and evaluate the equilibrium state of the shell by optimizing it under the constraints of the deformed mesh. Different from the accurate physical measurement which leads to highly nonlinear energy, our distortion measurement can provide plausible material effects by solving a simple quadratic energy. After evaluating the equilibrium state of the embedded geometry details by interpolation, we vibrate the geometry details around it by an explicit time integration (Figure 1). We also propose a method to directly manipulate the geometric textures which can transfer the position constraints to the external base mesh deformation algorithm.

The contributions of this paper include:

1. An efficient method to deform the offset mesh with the control of Poisson's ratio.
2. Real-time simulation of the dynamic effects for geometric textures with variant material effects.
3. Direct interaction with the geometric textures.

[*]Correspondence author: hj@cad.zju.edu.cn

1.1 Related Work

Deforming the offset mesh according to the deformation of base mesh is related to many mesh deformation methods. Gradient domain techniques preserve surface details by maintaining the length and orientation of the differential coordinates or mean curvature normal [Yu et al. 2004; Sorkine et al. 2004]. Various techniques have been proposed for manipulating the orientation, including distance based propagation [Yu et al. 2004; Zhou et al. 2005], harmonic guidance based interpolation [Zayer et al. 2005], and rotation-invariant representation [Lipman et al. 2005]. In [Huang et al. 2006], the orientation is automatically optimized, which eliminates some manual interactions. [Sumner and Popović 2004] solves the deformed shapes based on triangle-to-triangle transformation. They copy the target deformation gradient from the source, and then obtain the deformation results by solving linear systems. But in our application, these methods can hardly prevent serious self intersection between the base mesh and offset mesh. [Botsch et al. 2006] augments the surface meshes with rigid prisms and minimizes the elastic energy between neighboring prisms. The method needs to track the local rotation by time consuming polar decomposition,and cannot take the important material property, Poisson's ratio, into account.

Many physically-based techniques have been proposed over the past few years to simulate the dynamics of deformable objects. Most algorithms focus on deformable solid objects [Debunne et al. 2001; Müller and Gross 2004; Irving et al. 2006]. Thin shell objects, such as cloth, plate and paper are also addressed by [Baraff and Witkin 1998; Bridson et al. 2003; Grinspun et al. 2003]. However, to the best of our knowledge, few algorithms address the deformation and dynamics of geometric textures embedded in a "thick" shell space. Geometric textures are often modeled as polygon soups, and contain complex structure and huge number of vertices [Zhou et al. 2006]. It is thus prohibitive to directly simulate the effects using physically dynamic model. Although [Galoppo et al. 2006] proposed a method to represent deformable object as a soft shell over a rigid core, our method can handle deformable base mesh.

Recent geometrically based approaches are capable of generating physically plausible simulation results with interactive performance. Müller et al. [2005] introduced an excellent deformable model for point-base objects. At each time step of the dynamics simulation, each point is pulled toward its goal position which comes from shape matching. This approach provides unconditionally stable dynamic simulations with simple computation, and has been extended to general constraints [Müller et al. 2007]. A linear-time fast summation algorithm is proposed in the FLSM algorithm [Rivers and James 2007] to accelerate the simulation and computation discussed in [Müller et al. 2005] on a regular volumetric lattices. In this paper, we adopt the explicit time integration proposed in [Müller et al. 2005]. However, because of the highly complex topology structure of geometric textures and huge number of vertices, using shape matching to evaluate the goal positions is unaffordable for real-time applications, and it is hard for the shape matching approaches to simulate different material properties such as Poisson's ratio and anisotropic.

1.2 Overview

Suppose that the geometric textures are represented as an arbitrary polygon soup with a set of vertices $\mathbf{p} = \{\mathbf{p}_i\}$, which are embedded in a shell space between a base mesh $\mathbf{M}^b$ and an offset mesh $\mathbf{M}^o$. $\mathbf{M}^b = (\mathbf{x}^b, \mathbf{Tri})$ consists of N vertices $\mathbf{x}^b = (\mathbf{x}_1^b, \mathbf{x}_2^b, ..., \mathbf{x}_N^b)^T, \mathbf{x}_i^b \in R^3$, and a set of triangles $\mathbf{Tri}$. $\mathbf{M}^o = (\mathbf{x}^o, \mathbf{Tri})$ also has N vertices $\mathbf{x}^o$ and shares the same connectivity as $\mathbf{M}^b$. The shell mapping technique [Porumbescu et al. 2005] is used to tessellate the shell space as $\mathbf{S} = (\mathbf{x}, \mathbf{Tet})$, where $\mathbf{x} = (\mathbf{x}^b, \mathbf{x}^o)$ and $\mathbf{Tet}$ is the collection of the tetrahedrons. The vertices of the geometric texture are embedded into the tetrahedra by barycentric coordinates, and can be interpolated by the matrix-vector multiplication $\Phi\mathbf{x}$.

Given a base mesh animation which is deformed by any standard mesh deformation approach, we simulate the dynamic effects of the geometric textures with variant material effects using the following two-steps:

1. Compute the deformed shell $\hat{\mathbf{S}}$ according to $\mathbf{M}^b$.
2. Interpolate the goal position $\mathbf{g}$ of the geometric textures from $\hat{\mathbf{S}}$, then add dynamics to the geometric textures by vibrating its vertices around $\mathbf{g}$.

In our approach, the volume preservation property (i.e., Poisson's ratio ν) is considered in Step 1, and other material properties such as the stiffness related vibration frequency ω, energy dissipation (damping ζ) and anisotropic are considered in Step 2. The soft and stiff effects of the geometric textures can also be reflected when directly manipulating the geometric textures.

2 Deformation of the Offset Mesh

Physically based deformation may be the most straightforward method to deform the shell with material effects. For each tetrahedron element with element stiffness matrix $\mathbf{K}_e$ which contains the Young's modulus E and Poisson's ratio ν, one can formulate the following deformation energy as described in [Müller and Gross 2004]:

$$V_e(\hat{\mathbf{x}}) = \|(\mathbf{R}_e^T\hat{\mathbf{x}} - \mathbf{x})^T\mathbf{K}_e(\mathbf{R}_e^T\hat{\mathbf{x}} - \mathbf{x})\|^2$$

where $\mathbf{R}_e$ is the local rotation. Although such a formulation can be solved more efficiently than using Green's strain tensor, it still requires iteratively tracking the local rotation, updating stiffness matrix and solving a linear system with variant coefficient matrix.

Because the embedded geometry details can only vibrate in a shell space around the driven mesh, we can estimate the local rotation in the tetrahedron from the associated triangle. But even though, polar decomposition is still a time consuming procedure, when considering the tetrahedron elements in the shell are often inverted, more stable and complex polar decomposition [Irving et al. 2006] is required.

In this section, we describe an efficient method to compute the deformed shell $\hat{\mathbf{S}}$ by using *deformation gradients*. Deformation gradient is widely used in mesh deformation techniques [Sumner and Popović 2004]. It has a simple formulation for a tetrahedron element:

$$\begin{pmatrix}\hat{\mathbf{q}}_1 - \hat{\mathbf{q}}_4 & \hat{\mathbf{q}}_2 - \hat{\mathbf{q}}_4 & \hat{\mathbf{q}}_3 - \hat{\mathbf{q}}_4\end{pmatrix} \cdot \begin{pmatrix}\mathbf{q}_1 - \mathbf{q}_4 & \mathbf{q}_2 - \mathbf{q}_4 & \mathbf{q}_3 - \mathbf{q}_4\end{pmatrix}^{-1}$$

where $\hat{\mathbf{q}}_i$ and $\mathbf{q}_i$ are the position of the tetrahedron vertices at deformed state and the rest state respectively. After unrolling and packing the deformation gradients of all tetrahedrons of the shell $\mathbf{S}$ into a high dimensional vector $\mathbf{m}$, we can simply evaluate it as: $\mathbf{m}' = \mathbf{G}\hat{\mathbf{x}}$, where $\mathbf{G}$ is a sparse matrix decided by the rest pose of the shell.

Our basic idea is estimating the deformation gradient $\mathbf{m}'$ for each tetrahedron from $\hat{\mathbf{M}}^b$, and then deforming the shell by minimizing the following quadratic energy using least squares optimization:

$$\begin{aligned} V^o(\hat{\mathbf{x}}^o) &= \|\sqrt{\mathbf{EV}}(\mathbf{G}\hat{\mathbf{x}} - \mathbf{m}')\|^2 \\ &= \left\|\sqrt{\mathbf{EV}}\left(\begin{pmatrix}\mathbf{G}^b & \mathbf{G}^o\end{pmatrix}\begin{pmatrix}\hat{\mathbf{x}}^b \\ \hat{\mathbf{x}}^o\end{pmatrix} - \mathbf{m}'\right)\right\|^2 \\ &= \|\sqrt{\mathbf{EV}}(\mathbf{G}^o\hat{\mathbf{x}}^o + (\mathbf{G}^b\hat{\mathbf{x}}^b - \mathbf{m}'))\|^2 \end{aligned} \quad (1)$$

where $\mathbf{E}$ is a diagonal matrix whose entries come from the Young's modulus E, and $\mathbf{V}$ is also a diagonal matrix which contains the volume of tetrahedrons.

To estimate the deformation gradient $\mathbf{m}'$, for each triangle on the base mesh $(\mathbf{x}_i^b, \mathbf{x}_j^b, \mathbf{x}_k^b)$ with normal $\mathbf{n}$, we create a reference tetrahedron by introducing the fourth node:

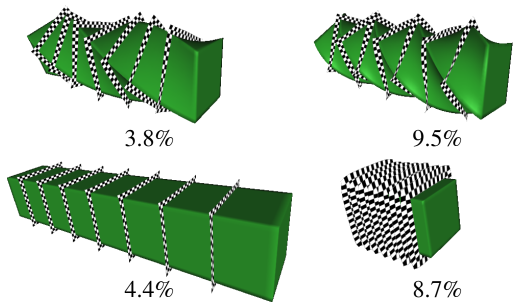
$$\frac{\mathbf{x}_i^b + \mathbf{x}_j^b + \mathbf{x}_k^b}{3} + \left\langle \frac{\mathbf{x}_i^o + \mathbf{x}_j^o + \mathbf{x}_k^o}{3} - \mathbf{x}_i^b, \mathbf{n} \right\rangle \mathbf{n} = \mathbf{x}_c^b + h\mathbf{n}$$

where $\mathbf{x}_c^b$ is the center of the base triangle, and h is the distance from the center of the associated offset triangle to the base triangle. For each triangle of the deformed base mesh, we can construct the deformed reference tetrahedron, compute its deformation gradient and fill the results into the entries in $\mathbf{m}'$ which correspond to the three tetrahedrons extruded from the triangle.

The above estimation of the deformation gradient does not consider Poisson's ratio of the material. When a sample of material with Poisson ratio ν is stretched from length l to $\hat{l}$ in one direction, it tends to contract (or rarely, expand) in the other two directions by the ratio $(\hat{l}/l)^{-\nu}$ [Lemaitre and Chaboche 1990]. Simply constructing the fourth node of the deformed reference tetrahedron as $\hat{\mathbf{x}}_c^b + h\hat{\mathbf{n}}$ cannot simulate the above effect. To overcome this problem, we use the following equation to evaluate the position of the fourth node:

$$\hat{\mathbf{x}}_c^b + \left(\frac{\hat{A}}{A}\right)^{\frac{\nu}{\nu-1}} h\hat{\mathbf{n}} \quad (2)$$

where $\hat{A}$ and A are the areas of the triangle at current state and the rest state respectively. Please see Appendix A for the detailed proof.



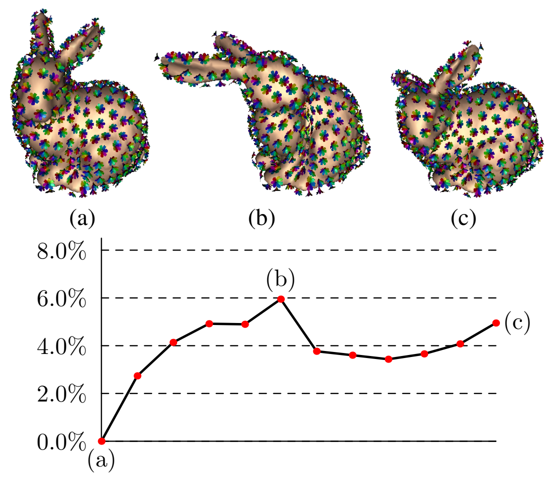


Figure 2: *The upper box model shows the effect of incompressible material ($\nu = 0.5$). The change in volume is shown by the values below the figures. The lower bunny model shows the comparison with the physically based approach [Müller and Gross 2004] ($\nu = 0.3$). The original bunny model (a) is deformed to (b), then (c). The diagram below the bunny figures shows the average difference in Frobenius norm of the deformation gradient in the tetrahedrons generated by the two methods.*

As shown in the upper part of Figure 2, we can simulate incompressible material $\nu = 0.5$ without great error. The comparison between our method and the physically based technique proposed in [Müller and Gross 2004] with Poisson's ratio $\nu = 0.3$ is shown in the lower part of Figure 2. Because we only solve a time-invariant linear system, the performance is 10 times faster than the non-linear physically based simulation [Müller and Gross 2004].

3 Dynamics of Embedded Geometry

Given the deformed shell $\hat{\mathbf{S}}$, the deformed vertices $\hat{\mathbf{p}}$ in the embedded geometry can be easily evaluated by barycenter interpolation. Although $\hat{\mathbf{p}}$ may appear some dynamic effect which comes from the animation of base mesh, the dynamics is often highly damped and has no high frequency vibration.

To complement the vibration for more realistic effect, we adopt the stable explicit time integration scheme proposed in [Müller et al. 2005] to vibrate the vertices around the goal position, i.e. the equilibrium position $\mathbf{g}$ which is interpolated from the deformed shell:

$$\begin{aligned}\mathbf{v}(t+\Delta t) &= \mathbf{v}(t) + \omega\frac{\mathbf{g}-\hat{\mathbf{p}}(t)}{\Delta t}\\ \hat{\mathbf{p}}(t+\Delta t) &= \hat{\mathbf{p}}(t) + \Delta t\mathbf{v}(t+\Delta t)\end{aligned} \tag{3}$$

where the coefficient ω is in range $(0, 1]$ used to control the stiffness of the geometry details, which affect the frequency of vibration. To introduce damping to the vibration, we can simply scale the velocity $\mathbf{v}$ by $(1 - \zeta)$ after updating the position $\hat{\mathbf{p}}$, where ζ is the damping coefficient in range $[0, 1]$.

Overshoot As shown in Figure 3, the above method cannot make the geometry details follow rapidly deformed mesh well, especially when the stiffness ω is close to 1 and damping ζ is close to 0.

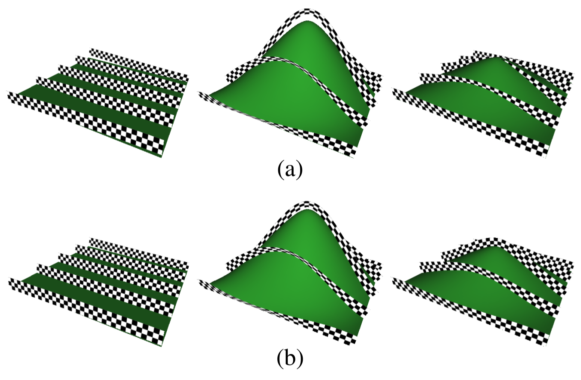


Figure 3: *As the plane is bulging and denting, the goal position of textured strips raises up and drops down. Without the restriction of vibration amplitude, the strips often overshoot the desired position (a). We smoothly move the vertices of geometric textures towards the goal position to prevent such artifacts (b).*

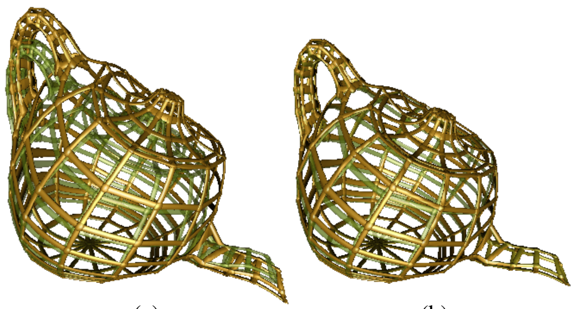
To solve this problem, we first calculate the distance d_i between the current position $\hat{\mathbf{p}}_i(t)$ and the goal position $\mathbf{g}_i$, then move $\hat{\mathbf{p}}$ towards $\mathbf{g}$ before vibrating the geometry details by Equation 3 if the max distance $\max\{d_i\}$ is larger than a threshold r:

$$\hat{\mathbf{p}}(t) \leftarrow \mathbf{g} + \frac{r}{\max\{d_i\}}(\hat{\mathbf{p}}(t) - \mathbf{g}) \tag{4}$$

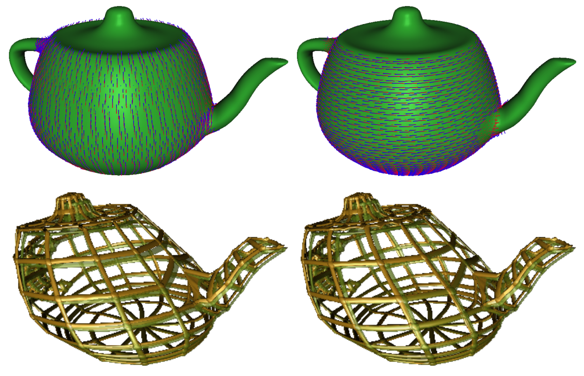
In all of our experiments, the threshold r is set to 50% of the thickness of the shell.

Uniform vs. Non-uniform Varying the material over the object can produce interesting effects, which can be incorporated into our algorithm by assigning different stiffness parameter ω and damping parameter ζ for the vertices in the geometric textures. Figure 4 shows such a result to demonstrate the non-uniform material effect. To simulate a teapot model made by the brass straps, uniform stiff and damping parameters lead to unrealistic result as shown in (a). To achieve nearly rigid effect on the handle in (b), we use larger ω, ζ in these regions, and set smaller ω, ζ over the body of the teapot.

Isotropic vs. Anisotropic Different from the isotropic material, behavior of the anisotropic material is related to local directions. The local frame (e_1, e_2, e_3) of the material is often evaluated by diagonalizing the deformation gradient which is known from step 1 of our algorithm. In our implementation, just transforming the directions $e_i, i = 1, 2, 3$ to $\hat{e}_i = m'e_i/\|me_i\|$ by the estimated deformation gradient m' at each triangle also produces plausible results, although the transformed three directions $\hat{e}_i$ may not be orthogonal anymore. When vibrating a vertex in the geometric textures, we decompose the velocity and the offset between its current position and goal position to these directions, and apply different material parameters to each component. In Figure 4, the vibrations of the strips are highly damped in different directions.



(a) (b)



(c) (d)

Figure 4: *(a) is the dynamic result of uniform stiffness and damping parameters. As shown in (b), different parameters can make the body of the teapot vibrates in larger amplitude and lower frequency than other regions. The goal position is rendered in translucent appearance. Anisotropic effects by assigning direction related damping coefficients. (c) and (d) are both driven by the same base mesh animation. The damping coefficient is smaller along the directions shown in the upper row. Such an anisotropic material makes the strips, which are orthogonal to the directions, vibrate less.*

4 Directly Manipulating the Geometric Textures

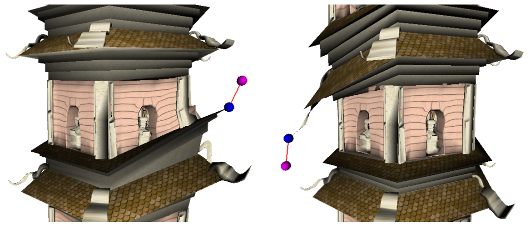
In addition to driving the dynamics of geometric textures by the base mesh, users can also directly manipulate the vertex in the geometry textures.

After a vertex $\hat{\mathbf{p}}_c$ in the geometric textures is selected, we create a spring to connect it and the target position $\mathbf{q}_c$ specified by the user. Because the vertex vibrates around its goal position in small amplitude, we replace the position of one end of the spring from $\hat{\mathbf{p}}_c$ to $\mathbf{g}_c$. Then the following energy in the springs are put into Equation 1:

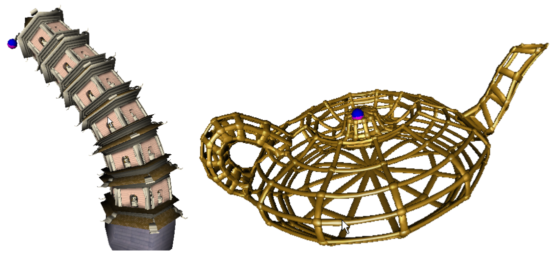
$$\begin{aligned} V_c^o(\hat{\mathbf{x}}^o) &= k^o \left\|\hat{\mathbf{p}}_c - \mathbf{q}_c\right\|^2 \\ &\approx k^o \left\|\mathbf{g}_c - \mathbf{q}_c\right\|^2 \\ &= k^o \left\|\Phi_c^o \hat{\mathbf{x}}^o + (\Phi_c^b \hat{\mathbf{x}}^b - \mathbf{q}_c)\right\|^2 \end{aligned} \tag{5}$$

where k^o is the stiffness of the spring, and Φ_c^b and Φ_c^o are the submatrices associated with the base mesh and offset mesh respectively in the whole tetrahedron interpolation matrix Φ.

As shown in Figure 5 (a), we can drag a vertex slightly like touching it, and not only the vertex itself will move and vibrate, but the neighbor vertices are also affected, which produces realistic result.



(a)



(b)

Figure 5: *(a):Drag a corner of the tower model slightly, and the geometric texture vibrate locally. (b):By transferring constraints on the geometric detail to the base mesh, we can control the whole shape of the geometric detail.*

Furthermore, when the deformation algorithm for the base mesh is energy based, we can directly manipulate the geometric textures to deform the base mesh. A naive method is modifying the energy in Equation 5 to make $\hat{\mathbf{x}}^o$ as known, and adding the energy to the external algorithm to deform the current base mesh $\hat{\mathbf{x}}^b$ to the next base mesh $\hat{\mathbf{x}}'^b$. But such a method converges very slowly, even is unstable. It can be explained as follows: the deformation of base mesh $\Delta\hat{\mathbf{x}}^b = \hat{\mathbf{x}}'^b - \hat{\mathbf{x}}^b$ fully compensates the offset between $\mathbf{g}_c$ and $\mathbf{q}_c$ by $\Phi_c^b \Delta\hat{\mathbf{x}}^b$, and the shell space between the base mesh and offset mesh is highly distorted, even inverted. After optimizing the energy dominated by Equation 1, the deformation of offset mesh $\Delta\hat{\mathbf{x}}^o$ roughly follows the deformation of base mesh $\Delta\hat{\mathbf{x}}^b$, which contributes similar offset $\Phi_c^o \Delta\hat{\mathbf{x}}^o$ to the constrained vertex, and makes $\mathbf{g}_c$ overshoot $\mathbf{q}_c$ greatly.

Instead of using Equation 5, we formulate another energy to drive external algorithm. The basic idea is to make the position of vertex in the geometry details as independent of the offset mesh as possible. To achieve this goal, we decompose the position of the constrained vertex as:

$$\mathbf{g}_c = \mathbf{g}_c^b + \mathbf{h}_c$$

where $\mathbf{g}_c^b$ is the projection position on the base mesh which is linearly dependent on x^b, and $\mathbf{h}_c$ is the offset vector.

To determine the above projection direction for the vertex in the geometric textures, we first find the tetrahedron which contains it, and further get the associated triangle in the base mesh and offset mesh. If projecting the vertex along the normal of the base triangle, the point $\mathbf{g}_c^b$ may be far away from the base triangle and lead to instability problem, especially in the region with high curvature, e.g. the tip of the bunny ears. Instead, we project the constrained vertex onto the base triangle along the line $\overline{\mathbf{x}_c^b \mathbf{x}_c^o}$ which connects the centers of the related base triangle and offset triangle. Using this projection direction, the point $\mathbf{g}_c^b$ rarely falls outside of the base triangle in all of our experiments, and achieves stable optimization.

The projection point $\mathbf{g}_c^b$ can be represented as a linear combination of the three vertices in the base triangle $\mathbf{g}_c^b = \Psi_c \mathbf{x}^b$, where Ψ_c is a sparse matrix whose non-zero entries correspond to the above three vertices in the base mesh and related barycenter coordinates. Finally, the position of the constrained vertex is expressed as:

$$\mathbf{g}_c = \Psi_c \mathbf{x}^b + \mathbf{h}_c$$

Then we can formulate the following energy to simulate the spring connecting the goal position $\mathbf{g}_c$ of the constrained vertex and $\mathbf{q}_c$, then add it into the deformation energy of external base mesh deformation algorithm [Huang et al. 2006]. The results can be found in Figure 5 (b).

$$\begin{aligned} V_c^b(\hat{\mathbf{x}}'^b) &= k^b \left\|\mathbf{g}_c' - \mathbf{q}_c\right\|^2 \\ &= k^b \left\|\Psi_c \hat{\mathbf{x}}'^b + \mathbf{h}_c - \mathbf{q}_c\right\|^2 \\ &= k^b \left\|\Psi_c \hat{\mathbf{x}}'^b + (\mathbf{g}_c - \Psi_c \hat{\mathbf{x}}^b) - \mathbf{q}_c\right\|^2 \end{aligned} \tag{6}$$

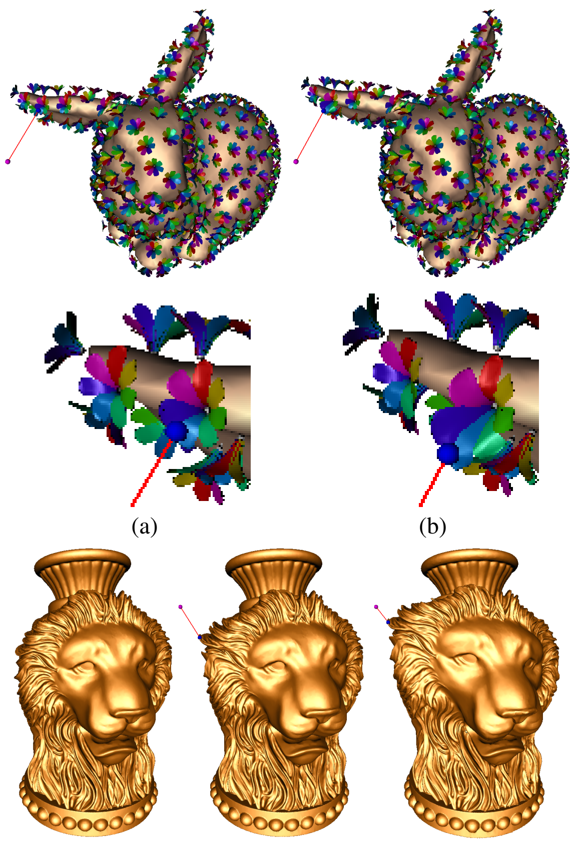


Figure 6: *Bunny: Different material effects of the geometric textures can be achieved by adjusting k^b and k^o. In the figure, the bottom row shows zoomed part in the upper row. (a) and (b) using the same k^b, but k^o is smaller in (a) so that it seems the geometric textures in (a) are stiffer than (b). Vase: We deform the model in (c) with the same k^o, but k^b in (d) is smaller than the one in (e). Such a parameter setting makes the base mesh in (d) is affected less by the manipulation of geometric textures, so that the base mesh is stiffer.*

As shown in Figure 6, different effects can be achieved by tuning the stiffness coefficients k^b and k^o. For example, when k^b is smaller but k^o is larger, the base mesh only deforms a little, and the offset mesh deforms greatly, which leads to an effect that the geometric textures are very soft. On the contrary, the geometric textures will appear stiff.

Kun et al. [Zhou et al. 2007] also proposed a method for manipulating the geometric textures, but they handle the offset mesh and the base mesh together, and cannot achieve different material effects for the base mesh and the geometric textures embedded in the shell space.

5 Conclusion and Future Work

In this paper, we proposed a two-step scheme to simulate the dynamics effect of geometric textures. In the first step, the shell is deformed by optimizing a simple quadratic energy which is much more efficient than physically based method. Poisson's ratio is handled by explicitly estimating the deformation gradient. Stiffness, damping, and anisotropy are taken into account in the second step which vibrates the vertices by stable explicit time integration. We also propose a method to manipulate the geometric textures directly, and can transfer the position constraints to the base mesh by an additional energy to the external base mesh deformation algorithm. The results show that such a solution can stably produce plausible results in real-time (Tabel 1).

model	$\|\mathbf{Tet}\|$	$\|\mathbf{p}\|$	$\hat{S}$	$\hat{\mathbf{p}}(t)$ in CPU	$\hat{\mathbf{p}}(t)$ in GPU
tower	6336	29995	13	50	9
teapot	14208	56316	30	93	12
bunny	29373	509479	56	870	106
vase	1200	200002	5	352	41

Table 1: *The columns $|\mathbf{Tet}|$, $|\mathbf{p}|$ list the number of tetrahedrons in the shell space and the number of vertices in the geometric textures. The latter three columns list the cost (ms as unit) of computing deformed shell $\hat{S}$ and vibrated geometric textures $\hat{\mathbf{p}}(t)$ at each step, which are measured in 3.0GHz Pentium IV machine with a NVIDIA GeForce 7800 GTX graphics card.*

The major limitation of current method is that the offset mesh deformation algorithm cannot be efficiently implemented in GPU, because we need to solve a quite large sparse linear system. Even though our linear method is much faster than physically based method, when the base mesh contains too many triangles, the performance of the whole algorithm will drop down, which can be observed in Tabel 1. In the future, we will try to overcome this problem from two aspects: one is to solve the equation in GPU parallelly by some iterative methods, e.g. the Conjugate Gradient method, to replace the current factorization based method. Another is to explore how to use highly simplified base mesh to driven the geometric textures.

6 A Explicit Poisson's Ratio

Given an isotropic material with Young's modulus E and Poisson's ratio ν, the strain and the stress have the following relationship:

$$\begin{aligned} \varepsilon_x &= (\sigma_x - \nu(\sigma_y + \sigma_z))/E \\ \varepsilon_y &= (\sigma_y - \nu(\sigma_x + \sigma_z))/E \\ \varepsilon_z &= (\sigma_z - \nu(\sigma_x + \sigma_y))/E \end{aligned} \tag{A-1}$$

The strain and the stress in the tetrahedrons have some special properties, which leads to our explicit method to handle Poisson's ratio. Without the loss of generality, we assume that direction z is the normal direction of the base mesh. Because the reference tetrahedrons used for estimating the deformation gradient are independent of each other, the fourth node can move freely, and the stress in this direction σ_z is negligible compared with the other two. Based on this assumption, we have:

$$\varepsilon_z = \frac{\nu}{\nu - 1}\varepsilon_t \tag{A-2}$$

where $\varepsilon_t = \varepsilon_x + \varepsilon_y$.

But Equation A-2 is true only in the case of small deformations. To derive the more precise formula for large deformation, we take step size δ_t to apply the strain ε_t by $n = \frac{\ln(1+\varepsilon_t)}{\ln(1+\delta_t)}$ steps; meanwhile, the strain in the z direction changes by step size $\delta_z = \frac{\nu}{\nu-1}\delta_t$ which comes from Equation A-2, and after n steps the final strain is $\varepsilon_z = (1+\delta_z)^n - 1$.

Since:

$$\begin{aligned}\ln(1+\varepsilon_z) &= \lim_{\delta_t \to 0} n \ln\left(1 + \frac{\nu}{\nu-1}\delta_t\right) \\ &= \lim_{\delta_t \to 0} \frac{\ln\left(1 + \frac{\nu}{\nu-1}\delta_t\right)}{\ln(1+\delta_t)} \ln(1+\varepsilon_t) \\ &= \frac{\nu}{\nu-1}\ln(1+\varepsilon_t)\end{aligned} \tag{A-3}$$

Finally we have $1 + \varepsilon_z = (1+\varepsilon_t)^{\nu/(\nu-1)}$. ε_x and ε_y can be obtained by diagonalizing the deformation gradient over the base mesh. However, in many cases, we found the stretch in the tangent plane of the base mesh is often along one direction, and the strain along the other direction in the tangent plane is relatively small. Thus approximating ε_t by $\hat{A}/A - 1$ can improve the performance and without leading to big error.

Acknowledgments

We would like to thank the reviewers for their valuable comments. This work is supported in part by National Natural Science Foundation of China (No. 60703039,), the National High Technology Research and Development Program of China (No. 2007AA01Z336), RGC research grant (No. 416007), 973 program of China (No. 2009CB320804) and Kun Zhou is supported by NSFC (No. 60825201).

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 43–54.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, 11–20.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 28–36.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 31–36.

GALOPPO, N., OTADUY, M. A., MECKLENBURG, P., GROSS, M., AND LIN, M. C. 2006. Fast simulation of deformable models in contact using dynamic deformation textures. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 73–82.

GRINSPUN, E., HIRANI, A. N., DESBRUN, M., AND SCHRÖDER, P. 2003. Discrete shells. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 62–67.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3, 1126–1134.

IRVING, G., TERAN, J., AND FEDKIW, R. 2006. Tetrahedral and hexahedral invertible finite elements. *Graph. Models 68*, 2, 66–89.

LEMAITRE, J., AND CHABOCHE, J. 1990. *Mechanics of Solid Materials*. Cambridge University Press, England, Cambridge.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3, 479–487.

MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, 239–246.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3, 471–478.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2, 109–118.

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. *ACM Trans. Graph. 24*, 3, 626–633.

RIVERS, A. R., AND JAMES, D. L. 2007. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. 26*, 3, 82.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proceedings of the Eurographics symposium on Geometry processing*, 179–188.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3, 399–405.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.

ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. *Computer Graphics Forum, Proceedings of Eurographics 2005 24*, 3, 601–609.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3, 496–503.

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2006. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3, 690–697.

ZHOU, K., HUANG, X., XU, W., GUO, B., AND SHUM, H.-Y. 2007. Direct manipulation of subdivision surfaces on gpus. *ACM Trans. Graph. 26*, 3, 91.