Sliding Deformation: Shape Preserving Per-Vertex Displacement

Dmitriy Pinskiy

Walt Disney Feature Animation

Abstract
We present a novel algorithm for deforming a locally smooth polygonal mesh by sliding its vertices over the surface. This sliding deformation creates the visual appearance of texture animation without requiring an explicit global surface parameterization or the overhead of storing texture coordinates. The proposed deformation algorithm can also be employed to slide vertices over the surface to increase or decrease resolution in desired regions. To produce the sliding displacement, we define a deformation space that couples the precision of a local parameterization with the advantages of a global parameterization, needed for consistency of displacements over the affected region. On demand, we establish a set of local parameterization spaces that minimize distortion error around each displaced vertex. To propagate the displacement direction across the set of spaces while ensuring coherency, we calculate a representation of global direction in each local space. To map a displaced vertex from the deformation space back to the surface with minimal distortion, we use the local parameterization space of the given vertex. Our method is inherently parallelizable, works on arbitrary topology, and provides a user-friendly, intuitive interface.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Geometric algorithms, languages, and systems, I.3.5 [Computer Graphics]: Modeling packages, I.3.7 [Computer Graphics]: Animation

1. Introduction

In character animation, the effect of a surface sliding over itself can be a very powerful tool for enhancing and enriching the results of a base deformation. For instance, surface sliding is important in facial animation, where an animator needs to capture the subtle movement of skin over the skull. This sliding effect is heightened by texture, displacement maps, and/or attached geometry (e.g. sliding portions of the facial skin can naturally drive geometry for eye brows).

One of the major challenges that we address in vertex sliding is to develop an algorithm that does not rely on an input global UV parameterization. In general a given input parameterization might not be appropriate for our task. Moreover, a global parameterization might not be present at all, especially if high-quality texturing solutions such as per-face texturing (e.g. Ptex [BL08]) are employed. These solutions are particularly attractive as they provide an elegant and very natural way to handle spatial and temporal antialising, relieving users from storing UV coordinates and calculating a global surface parameterization. Therefore, techniques based on UV texture animation are simply not available for our use.

We propose to create the visual appearance of sliding by changing geometry, instead of texture coordinates. This approach has the advantage that it does not require a global parameterization. In our algorithm we use the following two assumptions: (1) input polygonal surfaces are locally smooth and (2) per-vertex displacement is small relative to the total affected area. Given that most character meshes are polygonal approximations of NURBS and subdivision surfaces, the first requirement is the norm in production environments. Similarly, since sliding effects generally capture secondary motion which most often is local in nature, local displacements are sufficient to represent the desired motion.

Because our sliding approach modifies the actual geometry, not the texture coordinates, it can also be used as a modeling tool. Modelers often need to increase resolution in particular areas of a mesh without altering the shape. This task can be accomplished by sliding vertices along the mesh to create areas with a high concentration of vertices or to produce the opposite effect and slide vertices away from a particular area.

The first step of the proposed sliding algorithm is to form a deformation space. This is done by establishing a set of local geodesic-aware parameterizations, each of which is independently calculated and aims to minimize the distortion error for its corresponding vertex neighborhood. We efficiently parameterize on demand, by initially including only the minimum neighborhoods in the local parameterizations, and then expanding them as the deformation requires. To insure consistency of displacement across the set of local spaces, we establish a global region direction and derive its representation in each local space. Thus the input displacement, expressed in global coordinates, can be consistently applied in each local space.

Finally, to accurately map a displaced vertex from the displacement space to the surface, we rely on the precision of the per-vertex local parameterization space.

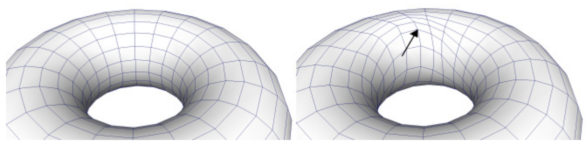

Figure 1: *Sliding along the polygonal surface is done in the direction of the black arrow.*

2. Related Work

The type of deformation that we employ can be classified as transformation propagation. As defined by Sorkin and Botsch [SB09], transformation propagation is characterized by a handle (vertex or CV), directly deformed by the user, and by the support region that surrounds the handle. The purpose of the support region is to smoothly propagate the handle's displacement toward the boundary of the region, blending the deformation with the unaffected portion of the mesh. Substantial work has been done in this area (e.g. [PK*03]); all of it primarily focuses on making shape deformation smooth and intuitive, following prescribed differential properties. Unfortunately, these techniques do not address our needs since, when we slide geometry, we strive to preserve the shape.

The choice of parameterization space is the key component of our deformation algorithm. The main challenge of parameterization is minimizing distortion error. Typically, parameterization is addressed as an optimization problem for various geometric metrics (e.g. area, edge length, angles) over the whole mesh. These global parameterization algorithms [e.g. see survey SP*06] provide very powerful tools, but lack interactivity and are overkill for our purposes since we are interested, not in overall distortion minimization, but in achieving precision only around specific vertices (the ones that are affected by deformations).

A number of efficient local parameterization schemes with accuracy biased to a prescribed vertex have been proposed [GS*02, SS09]. The most relevant local parameterization scheme was developed by Schmidt et al. in the context of placing a decal on an arbitrary polygonal mesh [SG*06]. This approach uses exponential maps, which provide an efficient tool to capture the geodesic properties of the surface in the parameterization space. We also employ a parameterization based on exponential maps, but we derive our maps in a way that allows their computation to be done in parallel. Unfortunately, such a local parameterization, by definition, lacks the notion of global UV direction, which we need for consistent displacement propagation from a handle to the rest of support region vertices. Therefore, local parameterization alone is not sufficient for our approach.

The work of Praun et al. addresses this lack of global direction in a local parameterization [PF*00]. A single global vector field is used to ensure alignment across neighboring patches. The surface is then textured with a small patch using local UVs. We are taking the opposite approach. First we define a set of local parameterizations, which best suits the geometry of interest. Then, based on the local parameterizations, we derive the global region direction, making sure that we have the most precision in the area displaced the most. Thus, we combine the precision of a local parameterization scheme with the consistency of global parameterization.

Skinning presents an orthogonal, but related, approach to our sliding deformation. For our purposes, it has the disadvantage of requiring influence objects (e.g. bones, joints), as well as the tuning of weight maps. Thus it is less attractive than a stand-alone deformation technique as the basis for a simple interactive modeling and animation tool.

3. Algorithm

We start our discussion of the algorithm with a definition of the input.

• *M* – polygonal quad mesh. Although we do not impose any requirements on the overall shape of the mesh, we assume that the surface is locally smooth, meaning the tangents (and normals) vary locally by a small amount.

• $\boldsymbol{P_0}$ – position of a handle vertex to be directly displaced by the user. Conceptually, $\boldsymbol{P_0}$ drives the sliding of the region.

• *r* – region radius in 3D, a real number that together with the handle defines the support region, *SR*. A vertex $\boldsymbol{P_i}$, contained in *SR* ($\boldsymbol{P_i} \in SR$), is called an active vertex.

• $\boldsymbol{d}$ – 3D displacement, prescribed for $\boldsymbol{P_0}$. We assume the magnitude of $\boldsymbol{d}$ is small since we primarily address secondary deformations which produce relatively small displacements along the surface.

• *fallOff()* – non-linear decreasing damping function (with range from 0.0 to 1.0) that takes the distance from a given vertex to the handle and outputs a numerical value.

The goal of our algorithm is to slide vertices in accordance with the input, producing intuitive and consistent displacements for active vertices, seamlessly handling arbitrary manifold topology (including extraordinary vertices), and without requiring an input global parameterization. The positions of the active vertices should be constrained to the surface of the mesh (see Figure 1). We place a special focus on performance, and therefore, computationally intensive operations, such as projecting points on the mesh by simply finding the closest point, are outside of the tool set available for our approach.

The main steps of the algorithm are the creation of a deformation space and vertex displacement. To ensure that the deformation space accurately reflects local geometry around each affected vertex, we form the space as a set of local geodesic-aware parameterizations. Then we establish the common surface direction across the spaces by calculating a representation of the support region's coordinate basis in each local parameterization space. We use these local representations of the global direction to ensure intuitive, consistent results across the affected faces.

3.1. Local Parameterization

The goal of this step is to produce a set of local parameterization spaces that we will use to map from the deformation space to the surface. Each parameterization space is a local coordinate system, centered at a particular active vertex $\boldsymbol{P}$, and is dedicated to the displacement of $\boldsymbol{P}$. Therefore, each space covers the area of the potential displacement for $\boldsymbol{P}$ and is parameterized with the goal of capturing geodesics, splayed out radially from $\boldsymbol{P}$. Initially the local parameterization of $\boldsymbol{P}$ includes only its topological disk, but then, to accommodate the displacement of $\boldsymbol{P}$, the local parameterization dynamically expands to cover more points in the neighborhood of $\boldsymbol{P}$ as needed.

The parameterization is constructed in $\boldsymbol{\Psi_P}$, the tangent space of $\boldsymbol{P}$. For a mesh point $\boldsymbol{Q}$ in the neighborhood of $\boldsymbol{P}$, we find $\boldsymbol{Q'}$, its image in $\boldsymbol{\Psi_P}$:

$$\boldsymbol{Q'} = \boldsymbol{P} + \boldsymbol{u}_{PQ} \qquad (1)$$

where $\boldsymbol{u}_{PQ}$ is a vector in $\boldsymbol{\Psi_P}$, pointed from the origin $\boldsymbol{P}$ of the local parameterization to $\boldsymbol{Q'}$ (see Figure 2).

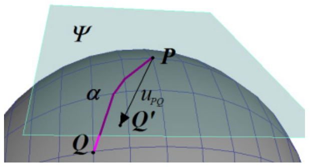

Figure 2: *Mapping between the geodesic line* $\boldsymbol{\alpha}$ *(from* $\boldsymbol{P}$ *to* $\boldsymbol{Q}$*) and the vector* $\boldsymbol{u}_{PQ}$ *in* $\boldsymbol{\Psi_P}$ *(the tangent space of* $\boldsymbol{P}$*).*

We derive $\boldsymbol{u}_{PQ}$ using an exponential map $\hat{\boldsymbol{u}}_{PQ} = exp_P\ (\boldsymbol{Q})$ – a mapping between a given point $\boldsymbol{Q}$ in the neighborhood of $\boldsymbol{P}$ and a unit vector $\hat{\boldsymbol{u}}_{PQ}$ that is tangent at $\boldsymbol{P}$ to α_{PQ}, the $\boldsymbol{PQ}$ geodesic line (i.e. $\alpha_{PQ}(0) = \boldsymbol{P}$ and $\alpha'_{PQ}(0) = \hat{\boldsymbol{u}}_{PQ}$) [DoC76]. Then we rewrite (1) as

$$\boldsymbol{Q'} = \boldsymbol{P} + |\alpha_{PQ}|\ \hat{\boldsymbol{u}}_{PQ} = \boldsymbol{P} + |\alpha_{PQ}|\ exp_P\ (\boldsymbol{Q}) \qquad (2)$$

where $|\alpha_{PQ}|$ is the geodesic distance from $\boldsymbol{P}$ to $\boldsymbol{Q}$.

Thus to define exp_P, we need to find the geodesics corresponding to $\boldsymbol{u}_{PQ}$. Schmidt et al. use the fact that if $\boldsymbol{Q}$ is in $\boldsymbol{P}$'s first ring of neighbors, it is trivial to compute α_{PQ} (and so $\boldsymbol{u}_{PQ}$) [SG*06]. When $\boldsymbol{Q}$ is not in the first ring, they express $\boldsymbol{u}_{PQ}$ in terms of $\boldsymbol{u}_{RQ}$, where $\boldsymbol{u}_{RQ}$ is given by exp_R, the exponential map of some other vertex $\boldsymbol{R}$, directly connected to $\boldsymbol{Q}$. Thus, this computation of $\boldsymbol{u}_{PQ}$ introduces dependencies in the construction of the exponential map.

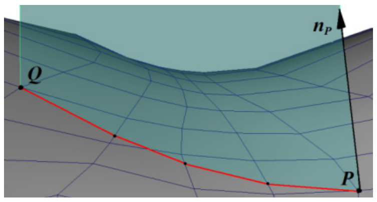

Figure 3: *The normal curve from* $\boldsymbol{P}$ *to* $\boldsymbol{Q}$ *is obtained as the intersection of the polygonal surface and the plane, spanned by* $\boldsymbol{n_P}$ *(normal at* $\boldsymbol{P}$*) and direction* $\boldsymbol{PQ}$*.*

In contrast, we use a normal-curve based approach with the advantage that the calculation of an exponential map is independent from its neighbors. As Zorin states, a normal curve is defined as the intersection between the surface and a plane that contains the normal; and consequently for any tangent direction and the normal at a particular point, one can define a plane, spanned by these two directions, and thus obtain a unique normal curve [Zor02]. Since a geodesic curve locally is a normal curve [Zor02], under our assumption about local smoothness we approximate the geodesic curve from $\boldsymbol{P}$ to $\boldsymbol{Q}$ by a sequence of line segments, calculated as intersections between the faces and a plane that is spanned by direction $\boldsymbol{PQ}$ and the normal at $\boldsymbol{P}$ (see Figure 3). We sum the length of the face-plane intersections to approximate the geodesic distance $|\alpha_{PQ}|$. The intersection of the plane with the tangent space, $\boldsymbol{\Psi_P}$, defines the direction for $\boldsymbol{u}_{PQ}$.

After we have calculated the exponential map for each vertex $\boldsymbol{Q}$ in $\boldsymbol{P}$'s neighborhood, we also establish a local coordinate system with origin at $\boldsymbol{P}$ and an orthonormal basis $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ for $\boldsymbol{\Psi_P}$. (Note: $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ can be trivially computed by applying Gram-Schmidt orthogonalization to any two projected edges incident to $\boldsymbol{P}$.) Thus $\boldsymbol{\Psi_P}$, a subspace of $\boldsymbol{R}^3$, can be defined as

$$\boldsymbol{\Psi_p} = \operatorname{span}\{\boldsymbol{b}_1, \boldsymbol{b}_2\} \qquad (3)$$

3.2. Correspondence Across Local Parameterizations

Once we establish a set of independently-built local parameterizations, we need to find their orientation relative to the whole support region. We define $\{\boldsymbol{a}_1, \boldsymbol{a}_2\}$, the basis for the global region's coordinate system, as the orthonormal basis of the handle's tangent space, and then we calculate the representation of $\{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ in each tangent space.

Expanding in breadth-first order, we propagate the definition for $\{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ from the handle's tangent space to the tangent spaces of the rest of the active vertices. The actual propagation is done by projecting representations of $\{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ from already-processed tangent spaces onto their immediate neighbors. Thus, for a point $\boldsymbol{P}$, surrounded by m processed neighboring vertices, we derive $\{\boldsymbol{a'}_1, \boldsymbol{a'}_2\}$, the representation in $\boldsymbol{\Psi_P}$ (the local space of $\boldsymbol{P}$), as

$$\boldsymbol{a}'_i = \sum_{j=1}^{m} \frac{1}{m} \begin{bmatrix} | & | \\ \boldsymbol{b}_1 & \boldsymbol{b}_2 \\ | & | \end{bmatrix} \times \begin{bmatrix} -\boldsymbol{b}_1- \\ -\boldsymbol{b}_2- \end{bmatrix} \times \boldsymbol{a}_{ij} \qquad (4)$$

where $i \in \{1, 2\}$; $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ is the orthonormal basis of $\boldsymbol{\Psi_P}$; $\{\boldsymbol{a}_{1j}, \boldsymbol{a}_{2j}\}$ are the representations, computed in the neighboring tangent local spaces.

Even though the input surface is locally smooth, distortion can be introduced in the local representation of $\{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ as the front advances. However, due to the decreasing nature of *falloff*(), the damping function used for the magnitude of the displacement, the importance of maintaining an undistorted representation diminishes with progression from the handle outward, with the most precision needed in the handle's neighborhood. Therefore, by choosing the global orientation based on the handle's $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$, we ensure highest precision where it is required.

3.3. Vertex Displacement

Once we have established the deformation space, we displace the active vertices.

We project the input displacement $\boldsymbol{d}$ onto the handle's tangent space and express the displacement in region coordinates (s, t)

$$s = \boldsymbol{d} \bullet \boldsymbol{a}_1 \qquad (5)$$

$$t = \boldsymbol{d} \bullet \boldsymbol{a}_2 \qquad (6)$$

For each active point $\boldsymbol{P}$, we calculate $\boldsymbol{d_p}$, the displacement in $\boldsymbol{\Psi_p}$, using the global region coordinates (s, t) in the local representation of the region basis $\{\boldsymbol{a'}_1, \boldsymbol{a'}_2\}$

$$\boldsymbol{d_p} = falloff(|\boldsymbol{P_0\,P}|)\ (s\ \boldsymbol{a'}_1 + t\ \boldsymbol{a'}_2) \qquad (7)$$

To map a displaced vertex $\boldsymbol{P'}_{new}$ in the deformation space to the position $\boldsymbol{P}_{new}$ on the surface we rely on the local parameterization around $\boldsymbol{P}$. Using the vertex positions in the local parameterized space, we locate F', a projected face that contains the displaced point in $\boldsymbol{\Psi_p}$

$$\boldsymbol{P'}_{new} = \boldsymbol{P} + \boldsymbol{d_p} \in F' \qquad (8)$$

If such a face F' does not exist in the current local parameterization, we find an edge E' that lies on the boundary of the current local parameterization and is intersected by $\boldsymbol{PP'}_{new}$. We expand the parameterized neighborhood by including the vertices of the face incident to this edge. We check whether the newly-added face is F', and if not, we repeat this process until F' is reached. Then we calculate the intrinsic face coordinates (u, v) for $\boldsymbol{P'}_{new}$ inside F'. Finally, to obtain $\boldsymbol{P}_{new}$, we use (u, v) to map the location on F' to the 3D position on the corresponding face F of the surface mesh.

4. Implementation and Results

To serve our production needs, we have implemented the presented algorithm as Maya plugins in the modeling and animation contexts. In both implementations, we effectively abstracted the technical side of the algorithm away from the end user.

For modeling, we wrote an interactive tool, intended for repositioning vertices without introducing any significant changes in the model's shape. This tool has a brush-based interface, where the brush size defines the support region (the set of affected vertices). The vertex that is closest to the brush's center serves as the handle. For *falloff()* we use a decreasing cubic function. On mouse down, we compute the deformation space for the current geometry below the brush. On mouse drag, we displace vertices, computing $\boldsymbol{d}$ by subtracting the initial mouse position from the current one.

In the animation context, we implemented a sliding deformer, keeping in mind that the deformer is set up by a technical rigger and used by animators, the end users. For the rigger the deformer should be flexible enough in its configuration to ensure the desired sliding behavior, while one of the animators' main requirements is a very intuitive simple interface. To achieve specific sliding properties on a given character, the rigger paints a weight map. This map specifies the envelope value per vertex and serves as *falloff()*. The rigger specifies the handle vertex and two Maya locators – start and end. The difference between these two locators defines $\boldsymbol{d}$. Once the character is rigged, the only part of the interface exposed to animators is the end locator.

In our implementation, the local parameterization and displacement steps are multithreaded, resulting in very efficient performance. In our test animation scene, to enhance the base animation with sliding, we added the sliding deformer and observed less than 5% drop in frame rate while deforming 500 vertices.

5. Conclusion

We have described a novel deformation algorithm to slide geometry without requiring an input surface parameterization. The deformation space, the key component of our sliding algorithm, combines the advantages of both global and local parameterizations. The set of parameterizations, derived on the local geometry, insures efficiency and precision mapping between the deformation space and the surface. The global parameterization provides consistency of the handle-driven movement across the support region. Our algorithm has a great practical value, and we have demonstrated its successful application in both modeling and animation settings.

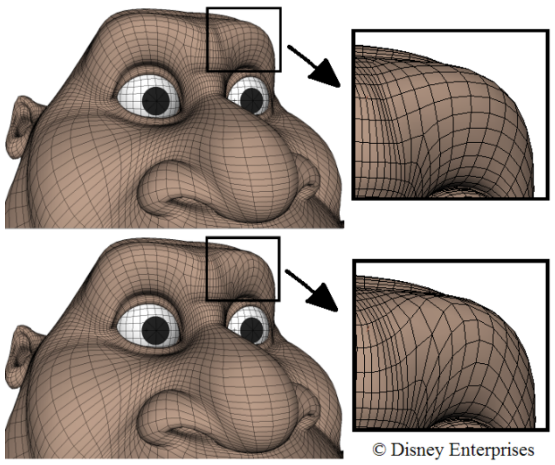

Figure 4: *Surface before and after sliding.*

6. Acknowledgements

First of all, I want to thank Maryann Simmons for helping with developing the idea and writing the paper. I am also very grateful to Andy Milne, Dylan Lacewell, and Brent Achorn, for reviewing the paper and making valuable suggestions. I want to especially acknowledge Carlos Cabral for championing the deformer in facial animation, and James Stapp for test-driving the sliding modeling tool.

References

[BL08] Burley B., Lacewell D.: Ptex: per-face texture mapping for production rendering. In *Eurographics Symposium on Rendering 2008* (2008), pp. 1155-1164.

[DoC76] Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[GS*02] Garimella R., Shashkov M., Knupp P.: Optimization of Surface Mesh Quality Using Local Parameterization. In *The Eleventh International Meshing Roundtable* (2002), pp. 41–52.

[PF*00] Praun E., Finkelstein A., Hoppe H.: Lapped texture. In *ACM SIGGRAPH 2000* (2000), pp. 465-470.

[PK*03] Pauly M., Keiser R., Kobbelt L., Gross M.: Shape modeling with point-sampled geometry. In *ACM SIGGRAPH 2003* (2003), pp. 641–650.

[SB09] Sorkine O., Botsch M.: Interactive Shape Modeling and Deformation. *Eurographics Tutorial (2009),* pp 11-37.

[SG*06] Schmidt R., Grimm C., Wyvill B.: Interactive decal compositing with discrete exponential maps. In *ACM SIGGRAPH 2006* (2006), vol. 25, pp. 605 – 613.

[SP*06] Sheffer A., Praun E., Rose K.: Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision* (January 2006), vol. 2, issue 2, pp. 105 – 171.

[SS09] Shapira L., Shamir A.: Local geodesic parameterization: an ant's perspective. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration.* Springer Berlin Heidelberg, 2009. pp. 127-137.

[Zor02] Zorin D. Curvature and geodesics, discrete laplacian and related smoothing methods. In course notes for *Geometric Modeling (G22.3033-002).* (2002) New York University. http://mrl.nyu.edu/~dzorin/geom04/lectures/lect08.pdf