View Dependent Modeling

Alexander Kolliopoulos

December 8, 2005

Introduction

There has been considerable interest in sketch based mesh modeling and editing, from the simple, intuitive interface of Teddy [1], to the more recent techniques to preserve detail in an existing mesh while modifying it with sketched strokes [2]. These approaches apply to editing a single mesh that is consistent in all views, but it is often the case in 2D animation that a character will have a drastically different shape depending on the angle at which it is viewed.

View dependent geometry seeks to provide a solution to animating 3D characters with meshes that should deform depending on the viewing angle [3]. With view dependent geometry, a single mesh is represented by several sets of vertex positions specified at different views. Each view corresponds to a point on a sphere, and the mesh vertex positions are interpolated from three key views that form a triangle containing the current view on the surface of the sphere.

This project seeks to combine the idea of view dependent geometry with sketch based mesh editing to create an interface for producing 3D characters that might behave more naturally for producing animation that mimics 2D styles. Specifically, focus is given to the problem of resolving multiple, perhaps inconsistent, strokes drawn by a user to deform a mesh from several views. An approach to view dependent geometry that does not require adequate coverage and spacing of key views is also described.

Interface

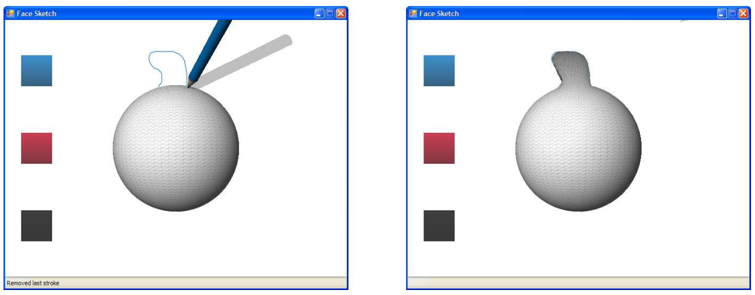

Figure 1: The mesh is deformed to match a stroke drawn by the user.

When the user starts the program, a base mesh is loaded, and the user is given control of a pencil. The base mesh is currently a sphere with about 4,000 vertices, but there is nothing that limits the initial mesh shape. The pencil allows the user to draw a stroke on a plane parallel to the viewing plane which is used to deform the mesh. The pencil wobbles slightly as the user moves it to give a lively feeling to the interface. A shadow is projected onto the drawing plane to indicate whether or not the pen is making contact with the surface. Note, also, that the pen wobble is decreased when a stroke is being drawn, to suggest physical contact. Other tools are available by clicking buttons to activate them, but they currently have no function.

Aside from being parallel to the viewing plane, the drawing plane passes through the mesh. When a stroke is drawn, a corresponding isocurve is found on the mesh, and vertices of the mesh are translated along a vector determined by the nearest point of the isocurve on the mesh (Figure 1).

When a user draws another stroke, a view dependent copy of the mesh is created for that orientation of the mesh relative to the camera (Figure 2). The deformation is applied to all other existing views, but with the original stroke for each view preventing large deformations near the area of that stroke on the mesh. The algorithm is described in detail in the next section.

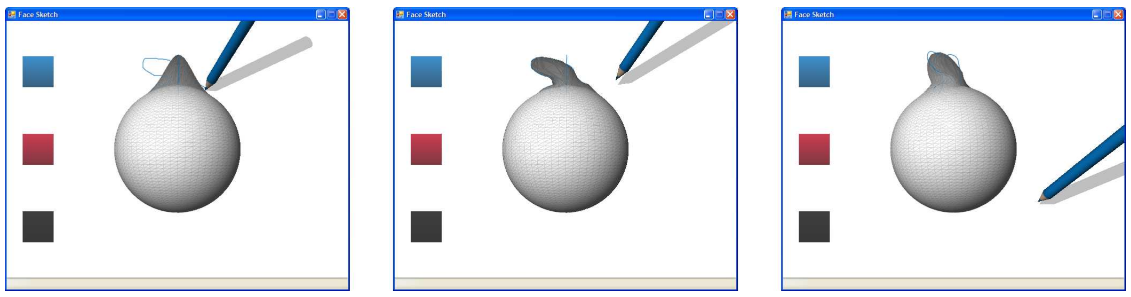

Figure 2: A second stroke creates a new view of the mesh.

Implementation

To map a stroke to an isocurve on a mesh, first every possible candidate isocurve is found—these are the curves on the surface of the mesh that intersect the drawing plane. The isocurve with a point nearest to the starting point of the stroke is then selected. This isocurve will be a complete loop, and it needs to be partitioned into two isocurves by breaking at the point nearest the ending point of the user's stroke. From these candidate isocurves, the final selection is made by choosing that with its midpoint closest to the stroke's midpoint. Figure 3 shows strokes with their corresponding isocurves on the mesh.

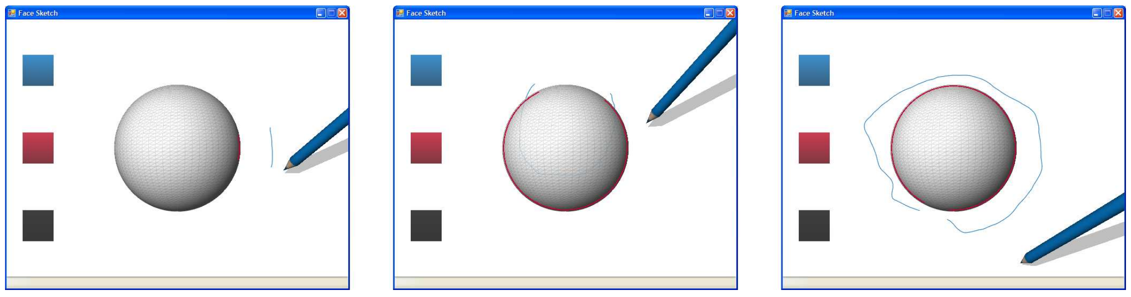

Figure 3: Strokes (blue) are mapped to the correct isocurve on the mesh (red).

To deform the mesh according to a stroke, a translation vector for each sample point on the isocurve to its corresponding point along the length of the stroke is calculated and stored. Each vertex, v_i, of the mesh is considered independently. The distance, d_i, to the nearest point on the isocurve is found, and the vertex is then translated along that point's vector, scaled by $\Gamma(d_i) = \exp(\frac{-d_i^2}{\varsigma})$, where $\varsigma = 2\sigma^2\alpha$, with σ being a constant variance, and α being the stroke arclength. Hence, large strokes will affect a wider region of the mesh.

When a stroke is drawn in another view, that deformation is applied to all other views where it does not conflict with an existing stroke. To determine whether strokes conflict, the existing stroke is assigned an influence value of $\iota_i = \Gamma(\tilde{d_i})^\chi$ over each vertex v_i, where χ is some constant that controls how broad the influence region should be. The scaling of the translation of v_i in this view is then calculated as $(1-\iota_i)\Gamma(d_i)$.

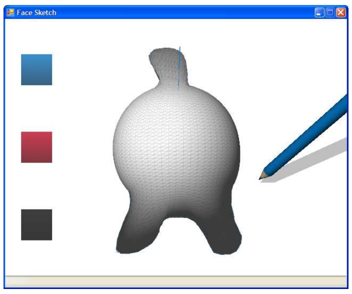

Figure 4: Strokes in nearby views are handled automatically.

The algorithm for determining the mesh vertex positions given a view allows any number of views with any spacing. Rather than interpolating the three surrounding views, a weighted sum of all views is taken. Let $\hat{\omega}$ be a point on the unit sphere corresponding to the current viewing orientation, and ω_j be the point corresponding to a set of vertex positions p_j, for view j. Then, the mesh vertex positions for view $\hat{\omega}$ may be computed as $\sum_j p_j \frac{(1-\|\hat{\omega}-\omega_j\|/2)^\zeta}{\sum_k (1-\|\hat{\omega}-\omega_k\|/2)^\zeta}$, with ζ controlling the sharpness of the transitions between views. Hence, an artist need not be concerned with the coverage of views or avoiding the creation of new views near existing ones. Figure 4 shows two strokes drawn from the same orientation, which does not cause any problems with the view dependent mesh calculation strategy described here.

Conclusions

The current system shows that there is some potential to this approach of sketch based modeling, but it needs quite a bit of work to become very useful. For example, the mesh cannot be translated through the drawing plane, so modifications can only be made near vertices that may not be in a convenient position for editing. Furthermore, the view dependent effects are not always ideal. One might make an edit on the left side of the object, and find that the edit has no effect on the view from the right side. It might be helpful to have an editing mode that allows strokes to be applied in the current view and the opposite view.

There are place holders for two other tools—a red pencil and a technical pen. The behavior of these tools has not been implemented, but a potential use for the red pencil would be as a stroke editor, allowing one to move a stroke that was made perhaps much earlier in the modeling process. The technical pen is intended to allow one to draw decals on the mesh. Beyond that an interesting application to explore is allowing one to draw contours that attempt to stay on a fixed length of the silhouette of a mesh while maintaining a shape drawn by the user. This idea is explored in [4], but it is not clear how robust such a system can be made.

No attention has been given to the problem of creating very stretched edges and faces, which can occur since the base mesh is never resampled. This is complicated by the fact that each vertex must be represented simultaneously in many views. Broader goals such as designing articulated characters and animations within the context of the interface described in this paper are further areas of future work.

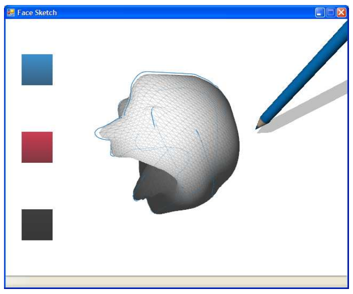

Figure 5: A view dependent mesh made with the program.

References

[1] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[2] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A Sketch-Based Interface for Detail-Preserving Mesh Editing. *ACM Transactions on Graphics*, 24(3):1142–1147, 2005.

[3] Paul Rademacher. View-Dependent Geometry. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 439–446. ACM Press/Addison-Wesley Publishing Co., 1999.

[4] Roman Zenka and Pavel Slavik. New Dimension for Sketches. In *SCCG '03: Proceedings of the 19th Spring Conference on Computer Graphics*, pages 157–163, New York, NY, USA, 2003. ACM Press.