A Tool for Subdivision of Quad/Tri Meshes with Sharp Features

Soham Uday Mehta

University of California, Berkeley [*]

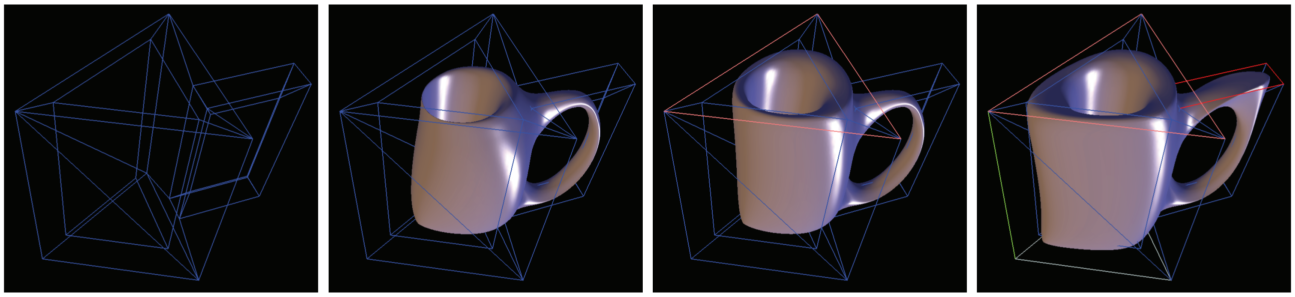


Figure 1: *(a) An example mesh with both Triangles and Quadrilaterals, in wireframe (b) The Subdivided Mesh, using the hybrid Quad/Tri subdivision algorithmn: No sharpness gives a very smooth surface which barely resembles the base mesh (c) Introducing sharpness at selected locations gives a different look (d) Another example with different sharpness values gives a still different appearance to the subdivision surface*

Abstract

Designers often want the added flexibility of having both quads and triangles in their models. It is also well known that triangle meshes generate poor limit surfaces when using a quad scheme, while quad-only meshes behave poorly with triangular schemes. Further, regular subdivision produces smooth surfaces even from sharp-cornered base meshes and sometimes sharpness in the subdivision surface is desrired. In this project we introduce a new tool for subdivision of meshes with triangles and and quadrilaterals, with user-specified sharpness on edges which permits great flexibility in controlling the output subdivision surface. This project essentially integrates two separate algorithms introduced by previous work into one framework, and provides a user-interface for interactive subdivision of quad-tri meshes.

1 Introduction and Related Work

Subdivision surfaces are currently one of the most powerful surface representations used to model smooth shapes. Unlike regular surface splines, such as NURBS, subdivision surfaces can handle shapes of arbitrary topology in a unified framework. Also, unlike polygonal meshes, subdivision surfaces generate smooth surfaces, which is important in designing aesthetically pleasing shapes. Subdivision surfaces were introduced by Catmull and Clark [Catmull and Clark 1978] and Doo and Sabin [Doo and Sabin 1978]. They both generalized tensor product B-splines of bi-degree three and two, respectively, to arbitrary topologies by extending the refinement rules to irregular parts of the control mesh. Later, in 1987 Loop [Loop 1987] generalized triangular Box splines of total degree four to arbitrary triangular meshes. These subdivision schemes have the desirable property that they admit a polynomial representation on the regular part of the mesh. Consequently, these surfaces are curvature continuous almost everywhere [Reif 1995]. The visual quality of a subdivision surface depends in a crucial way on the initial, or base, mesh of control vertices. For general shapes designers often want to model certain regions with triangle patches and others with quad patches. Unfortunately, both Catmull-Clark and Loop surfaces require that all patches be quadrilateral or triangular, respectively. In theory this is not a problem, since any triangle can be converted into three quads and any quad can be tesselated. However, as illustrated in Figure 2, Catmull-Clark surfaces behave very poorly on triangle-only base meshes: the resulting surface exhibits annoying undulating artifacts. Similarly, Loop schemes do not perform well on quad-only meshes. More importantly, designers often want to preserve quad patches on regular areas of the surface where there are two natural directions. Consequently it is often desirable to have surfaces that have a hybrid quad/triangle patch structure. A few schemes have been proposed to subdivide quad-tri meshes in a unified framework, approximating ones such as [Stam and Loop 2003], [Schaefer and Warren 2005] and interpolatory ones such as [Jiang et al. 2009]. The Hybrid subdivision algorithm in [Stam and Loop 2003] is surprisingly simple and is based on the fact that both Loop and Catmull-Clark surfaces can be generated by a linear subdivision step followed by vertex-smoothing, and this is the one we chose to implement. Their theory is discussed in Section 2. As can be observed in Fig2 (a), regular Quad/Tri subdivision produces extremely smooth surfaces, even starting from a relatively sharp-cornered mesh, and while this is mostly good, sometimes it is desirable to have some sharpness in the subdivision surface. Ideally we would like to be able to control the sharpness locally, without having to cram vertices in the base mesh. [Hoppe et al. 1994] introduce a scheme to modify the ubdivision rules to create piecewise smooth surfaces containing infinitely sharp features such as creases and corners. This is illustrated in Figure 1 (c). Extending this idea, [DeRose et al. 1998] give special rules for semi-sharp creases. We discuss these rules in Section 3.

We are not aware of any tool which combines these two algorithms, and we also feel the need for a convenient user interface to interact with Quad/Tri meshes and edit any edge's sharpness as desired. We discuss the implementation of the above two

[*]This work is the final project for CS284, Fall 2012, instructed by Prof. Carlo Sequin. The author can be contacted at sohamum@eecs.berkeley.edu

algorithms in one framework, in Section 4. Our results and User-Interface are shown in Section 5.

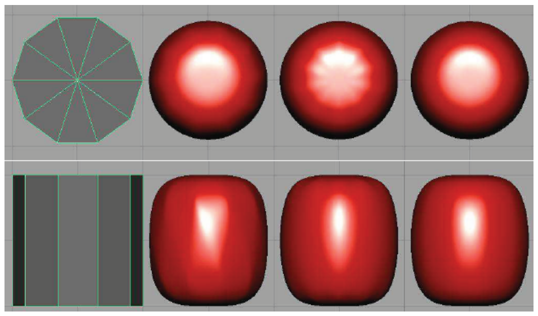


Figure 2: *On the left is a cylinder with both Quads and Tris. Loop (second from left) performs poorly on triangulated Quad meshes and Catmull Clark (second from right) performs poorly on quadrangulated Triangle meshes. The hybrid [Stam and Loop 2003] scheme (right) performs best. [Figure adapted from [Stam and Loop 2003]]*

2 Quad-Tri Subdivision

Here we breifly describe the quad/tri subdivision algorithm of [Stam and Loop 2003]. We start with a base mesh that is formed of quads and triangles only (e.g. Fig. 3(a)). The algorithm comprises two steps. In the first step, each each edge into two, each quad into four and each triangle into three, as shown in Fig 3 (b). Then in a second step we replace each vertex with a linear combination of itself and its direct neighbors resulting in the mesh shown in Fig 3 (c).

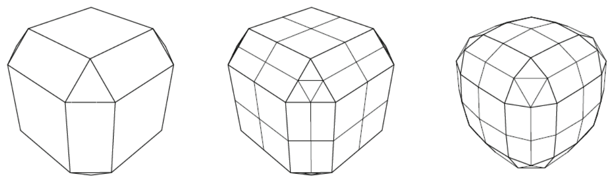


Figure 3: *The main steps in the Quad/Tri subdivision algorithm (a) A quad-triangle base mesh (b) After linear subdivision and (c) After the smoothing pass [Figure adapted from [Stam and Loop 2003]]*

2.1 Smoothing Pass

First consider the regular case when a vertex is surrounded by two adjacent quads and three adjacent triangles. In this case the obvious choice is the mask shown in Figure 4 (a), which is a simple average of the regular Loop and Catmull-Clark masks. Now let n_E be the number of edges emanating from the vertex and let n_Q be the number of quads surrounding the vertex. As shown in Fig. 4(b), denote by α the weights associated with the irregular vertex and let β and γ be the weights associated with the neighboring edge and face vertices, respectively. As observed by [Stam and Loop 2003], a natural generalization of the regular case is to let the weights be equal to

$$\beta = \alpha/2 \text{ and } \gamma = \alpha/4 \qquad (1)$$

The weights must sum to 1, $\alpha + n_E\beta + n_Q\gamma = 1$. Therefore, we set

$$\alpha = \frac{1}{1 + n_E/2 + n_Q/4} \qquad (2)$$

We use the above set of weights to smoothen each vertex in the linearly subdivided mesh of Fig. 3(b).

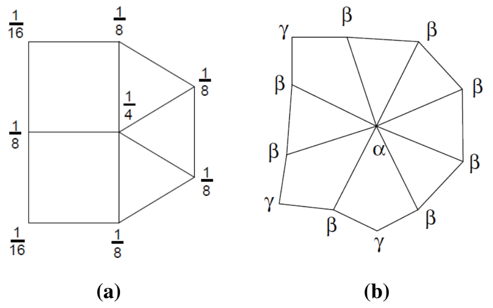


Figure 4: *(a) At a vertex after linear sudivision, the subdivision mask for the regular case (with 2 quads and 3 tris) is a hybrid of Loop and Catmull Clark, (b) The subdivision mask for the general case is extrapolated from the regular case, so that $\beta = \alpha/2$ and $\gamma = \alpha/4$ [Figure adapted from [Stam and Loop 2003]]*

2.2 Vertex Correction

The choice for the mask is quite arbitrary as there are potentially many other affine invariant generalizations. Ideally we want a mask which gives the most aesthetically pleasing surfaces. One way to formalize this requirement is to make the curvature well behaved at each vertex. [Stam and Loop 2003] observe that for $n_E < 5$ their algorithm produces pinching artifacts as shown in Fig 5(a). To limit surface at the corners, it should be drawn more 'inward'. This idea of a correction step was first introduced implicitly by Catmull and Clark to improve the behavior of the surface at a vertex corner where three quads meet. In essence, the correction step translates the smoothed vertex along the direction defined by the difference between its smoothed position $\mathbf{v_1}$ and its previous (after linear subdivision) position $\mathbf{v_0}$ by an amount η:

$$\mathbf{v_2} = \mathbf{v_1} + \eta(\mathbf{v_1} - \mathbf{v_0}) \qquad (3)$$

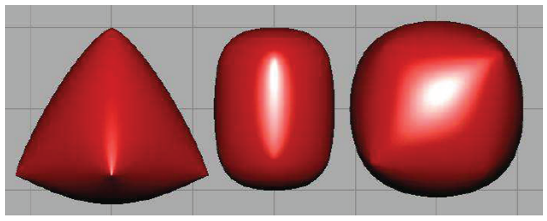
If $\eta > 0$, the vertex is drawn more inward, and if $\eta < 0$, the vertex is pused outward. To understand how to compute η, we must look at the eigenstructure of the subdivision matrix of the hybrid scheme. Let the eigen values of the matrix be

$$1 \geq \lambda_1 \geq \lambda_2 \geq \mu_1 \geq \mu_2 \ldots \qquad (4)$$

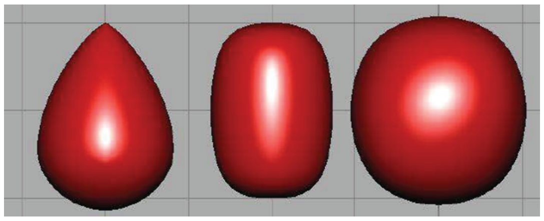
Then [Stam and Loop 2003] observe that the characteristic factor ρ determines the local curvature smoothness of the mesh.

$$\rho = \lambda_1/\mu_1^2 \qquad (5)$$

It is well known that $\rho = 1$ for $\mathbf{C^2}$ continuity. [Stam and Loop 2003] numerically determine the optimal value of η which makes ρ close to 1. The reader is encouraged to look up their paper for the specific values they propose, omitted here for brevity. It is worth mentioning that except for $n_E = 3$, they are able acheive a $\rho = 1$, hence they only have $\mathbf{C^1}$ continuity for the $n_E = 3$ case.



(a)



(b)

Figure 5: *(a) The basic smoothing pass still leaves undesirable sharp corners at vertices with less than 5 incident edges (b) The Vertex correction eradicates this issue [Figure adapted from [Stam and Loop 2003]]*

3 Sharp Features

Let us now discuss the sharp[1] subdivision scheme of [DeRose et al. 1998]. Although they propose their rules in a Catmull-Clark quad-only setup, it wasnt too hard to adapt their rules to the quad/tri subdivision algorithm. On a high-level, their hybrid subdivision algorithm uses infinitely sharp subdivision rules for the first s subdivision steps on an edge with sharpness s, followed by the smooth subdivision rules applied to the limit. Intuitively this leads to surfaces that are sharp at coarse scales, but smooth at finer scales. Let us first discuss the infinitely sharp subdivision rules.

An edge $\mathbf{p_1p_2}$ is tagged 'sharp' if its sharpness is $s \geq 1$. Then the new edge-point (also called odd vertex) $\mathbf{p}_o$ is placed at the midpoint $\mathbf{p}_o^{sharp} = 0.5(\mathbf{p_1} + \mathbf{p_2})$. In other words, in the framework of our Quad/Tri Subdivision, the edge point will not be moved after the linear subdivision step. If the edge has a sharpness $0 < s < 1$, the final position of the edge-point is a linear blend between the position computed using the sharp rules $\mathbf{p}_o^{sharp}$ and that computed using the smooth rules $\mathbf{p}_o^{smooth}$:

$$\mathbf{p}_o = s\mathbf{p}_o^{sharp} + (1-s)\mathbf{p}_o^{smooth} \tag{6}$$

Now consider a new vertex-point (also called even vertex) $\mathbf{p}_e$ produced from a base mesh vertex $\mathbf{p}$ with one or more incident sharp ($s > 0$) edges. If there is only one incident sharp edge, the vertex will be smoothed as usual - it is not affected by having an incident sharp edge.

If, however, there are exactly two incident sharp edges $\mathbf{pp_1}$ and $\mathbf{pp_2}$ then the new position is

$$\mathbf{p}_e^{\text{sharp}} = \frac{1}{8}(6\mathbf{p} + \mathbf{p_1} + \mathbf{p_2}) \tag{7}$$

[1]Henceforth we will use the term 'sharp' to mean both semi-sharp and infinitely sharp.

Note that if the two incident sharp edges both have sharpness $s_1, s_2 < 1$, then we use the average sharpness $s = 0.5 * (s_1 + s_2)$ to blend between the smooth and sharp positions of $\mathbf{p}_e$ as described above in equation 6.

Finally, if there are more than 2 incident sharp edges ($s > 0$) at a vertex, then the new position is the same as the old position, $\mathbf{p}_e^{\text{sharp}} = \mathbf{p}$. While subdividing from one level to the next, we must also reduce the sharpness of each edge at each level of subdivision, to conform to the intuitive notion that sharpness corresponds to the number of levels of subdivision for which the edge is kept sharp. The obvious solution is to reduce the per-edge sharpness by 1 - if an edge $\mathbf{e}$ with sharpness s splits into $\mathbf{e_1}$ and $\mathbf{e_2}$, then their sharpness is set to $s - 1$ each. [DeRose et al. 1998] however propose to blend sharpness from neighboring edges while reducing it at each level of subdivision, so that the disparity in the sharpness of neighboring edges is reduced at each subdivision step. The blending is done in a 3 : 1 ratio as indicated in Fig. 6.

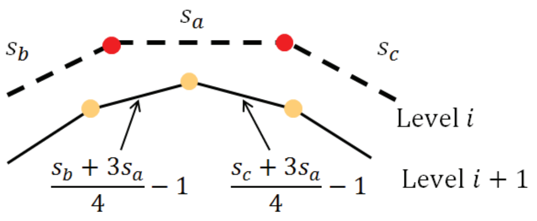


Figure 6: *Propagating edge sharpness at each subdivision step: Instead of simply reducing per-edge sharpness by 1, we blend sharpness between adjacent edges in a 3:1 ratio, to reduce the sharpness disparity between adjacent edges as we increase subdivision depth*

4 Implementation

Our algorithm combining sharp features and Quad/Tri subdivision was implemented in C++/OpenGL on a Visual Studio platform. The Mesh data structure comprises a interconnected lists of the following data structures:

```
Vertex{
   float3 position, normal;
   Face* startFace; //pointer to one Face
   int index;       //used for indexing with next level
   float s;         //sharpness, default = 0
}

Face{
   FaceType type;    //enum{Quad,Tri}
   Vertex* v[4];     //pointer to vertices
   Face* f[4];       //pointers to neighbor faces
   int index;        //used for indexing with next level
}

Edge{
   Vertex* v[2];     //pointer to end points
   float s;          //user-specified sharpness, default = 0
}
```

Note that both the edge and vertex data structures store the sharpness values. Out user-interface (discussed in the next Section) allows users to edit edge sharpness - only edges with $s > 0$ are pushed into the base mesh $\mathbf{M}_0$'s edge list when the system is 'updated'.

The core algorithm is implemented in the **sharpSubdivide** function which takes as input a fully connected mesh $\mathbf{M}_i$, and the output is a new, fully connected, subdivided mesh $\mathbf{M}_{i+1}$. We obviously start by calling **sharpSubdivide** with input $\mathbf{M}_0$. The main steps in this function are listed below:

```
1  Copy all vertex positions $\mathbf{p}_e$ from $\mathbf{M}_i$, into $\mathbf{M}_{i+1}$
2  For each face $f$ in $\mathbf{M}_i$
     1  Insert 4 (NULL) faces in $\mathbf{M}_{i+1}$
     2  If $f$ is a QUAD, insert a face center vertex into $\mathbf{M}_{i+1}$
3  For each even vertex $\mathbf{p}_e$, search incident edges from $\mathbf{M}_i$
     1  If 2 edges with $s > 0$ found, compute and store $\mathbf{p}_e^{\text{sharp}}$,
        and store $\mathbf{p}_e.s = 0.5(s_1 + s_2)$
     2  If more than 2 edges with $s > 0$ found, set $\mathbf{p}_e.s = -1$,
        indicating marked for no smoothing
4  Create a Map<edge> edges; for each edge $\mathbf{e}$ of each Face $f$
     1  If $\mathbf{e} \notin$ edges, insert new odd vertex $\mathbf{p}_o$ at edge midpoint
     2  If $\mathbf{e}.s > 0$; insert new edges into Mesh.edges, and set
        sharpness according to Fig. 6
     3  If $0 < \mathbf{e}.s < 1$ set $\mathbf{p}_o.s = \mathbf{e}.s$; if $\mathbf{e}.s > 1$ set set
        $\mathbf{p}_o.s = -1$
5  Update new vertex->StartFace pointers
6  Update new texture coordinates by averaging from previous
   Mesh
7  Update new texture coordinates by averaging from previous
   Mesh
8  Update Face->f neighbor face pointers
9  Create list<float3> newPos; for each vertex $\mathbf{p}$ in $\mathbf{M}_{i+1}$
     1  If $\mathbf{p}.s = -1$, store $\mathbf{p}$ in newPos and continue to next
        vertex
     2  If $\mathbf{p}.s = 0$, smoothen and apply vertex correction and
        store in newPos
     3  If $0 < \mathbf{p}.s < 1$ blend between smooth and sharp posi-
        tions and store in newPos
10 Copy position from newPos to the vertex list of $\mathbf{M}_{i+1}$
11 Compute all vertex normals by adding adjacent face normals
   and normalizing
```

The algorithm roughly consists of 3 parts. In the first part, steps 1 through 4, the new mesh $\mathbf{M}_{i+1}$ is created as per the Stam-Loop linear subdivision pass, and in addition vertices on sharp edges or with incident sharp edges are processed. The sharpness values of the odd and even vertices are also set if they were subdivided with $s < 1$, to be used later for blending between sharp and smooth. Particularly interesting is step 4, where we create a C++ Map of edges, which allows us to rapidly loop though each edge of each face and check if the edge was previously encountered, thus enabling us to efficiently create new odd vertices.

The second part, steps 5 through 8, essentially establishes connectivity in $\mathbf{M}_{i+1}$ as required by the vertex, face and edge data structures described earlier in this section.

Finally, in the last part, steps 9 and 10 apply the Stam-Loop smoothing pass as in Sec.2.1 and the vertex correction step of Sec.2.2. The new positions must be stored in a separate temporary list to avoid overwriting the old positions before all positions are recomputed. At this stage, if any vertices were marked with sharpness $0 < s < 1$, the blending step must be performed. The last step finally computed all vertex normals to be used for Gourard shading of the mesh.

5 Results

In Fig 1 we demonstrate the great degree of control our sharp subdivision algorithm provides. The base mesh consists of both quads and tris, and setting different sets of sharpness values produces different results as desired.

Our algorithm also produces smooth scalar field interpolation. It is difficult to prove $\mathbf{C}^1$ continuity for our algorithm explicitly, however the two base algorithms of [Stam and Loop 2003] and [DeRose et al. 1998] have been shown to be $\mathbf{C}^1$ everywhere. We observed no issues with texture coordinate interpolation for a variety of cases and this can be taken as an indication of $\mathbf{C}^1$ continuity. One example with a smooth texture is shown in Fig 7.

Our algorithm can trivially handle meshes with only quads, as shown in Fig 8 (a) and (b), and only tris, as in Fig 8(c) and (d). The second case also has a base mesh with a large number of triangles. This can be slow to initialize but it may be possible to speed up the algorithm with appropriate multi-threaded or GPU optimization.

5.1 Graphical User Interface (GUI)

An important contribution of this project is the user interface or GUI which allows the user to interactively edit edge sharpness and test the results on arbitrary manifold and water-tight meshes. It was designed on top of the open-source OpenGL User Interface Library GLUI [Rademacher 2008]. The interface, as shown in Fig 9, has a main display area which shows the base mesh $\mathbf{M}_0$ in wireframe only, as a cage around the level-5 subdivided mesh $\mathbf{M}_5$, which is Gourard shaded and can be textured. The user can hide either mesh to focus on the other. All the function are provided in two panels at the bottom and right of the main display area. In brief, these functions are:

- Loading meshes in standard OBJ format
- Clicking on the base mesh edges and editing sharpness
- Updating the subdivided mesh edges and editing sharpness
- Loading .RAW texture files if the base mesh was provided with texture coordinates
- Camera rotation, translation and zoom
- Adjustable lighting: both direction and color
- Saving the current view of the display area as a .RAW image

6 Key Challenges

One of the key challenges of this project was handling quads and tris separately for each face operation. The issue is non-trivial be-

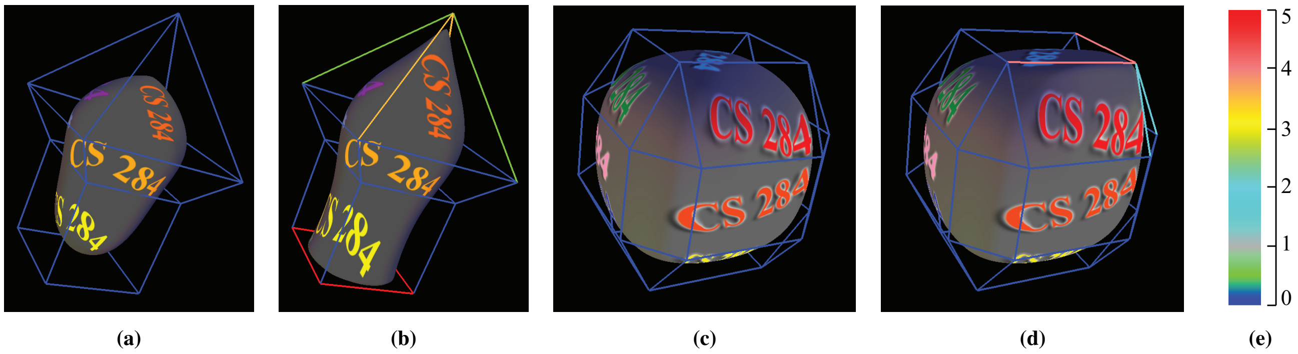
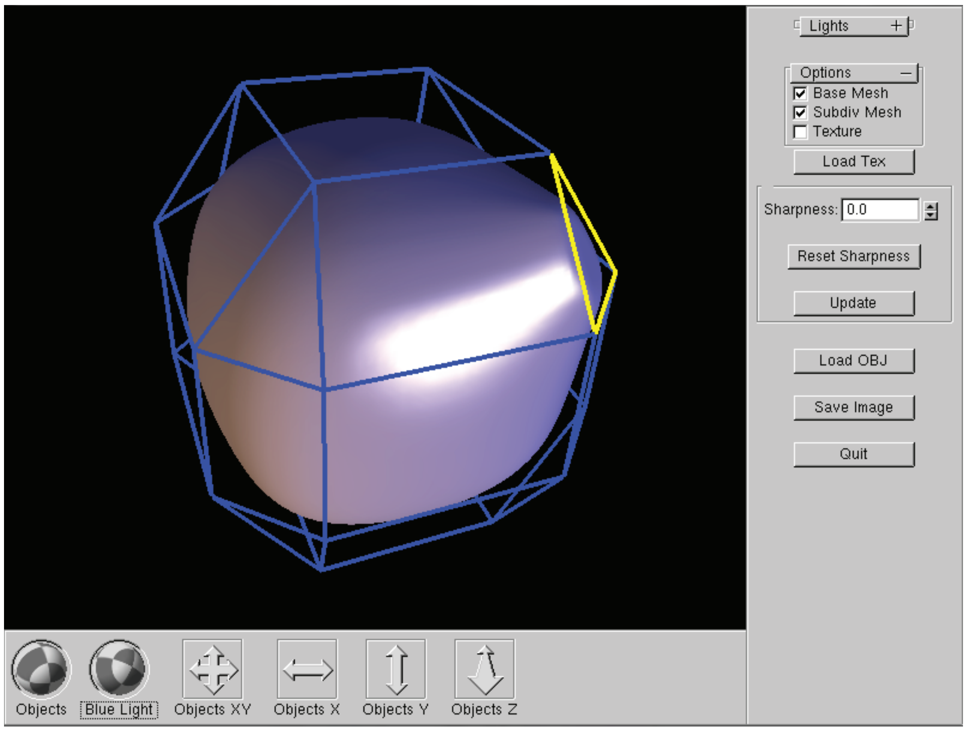


Figure 7: *(a) An example of a simple mesh with a textured subdivision surface (b) With sharp edges our alogrithm still keeps the textures smooth across edges, (c) and (d) show another example of a textured subdivision surface. In (e) we show the color code used for indicating edge sharpness in all figures in this report and in our User Interface*



☑ Base Mesh
☑ Subdiv Mesh
☐ Texture
Sharpness: 0.0

Figure 9: *Our user-interface affords great interactivity with the algorithm. The horizontal panel on the bottom controls the camera and light directions, the vertical panel on the right provides buttons to load a new Mesh, edit sharpness of selected edges, update the mesh with the new sharp edges, load texture files. and save images.*

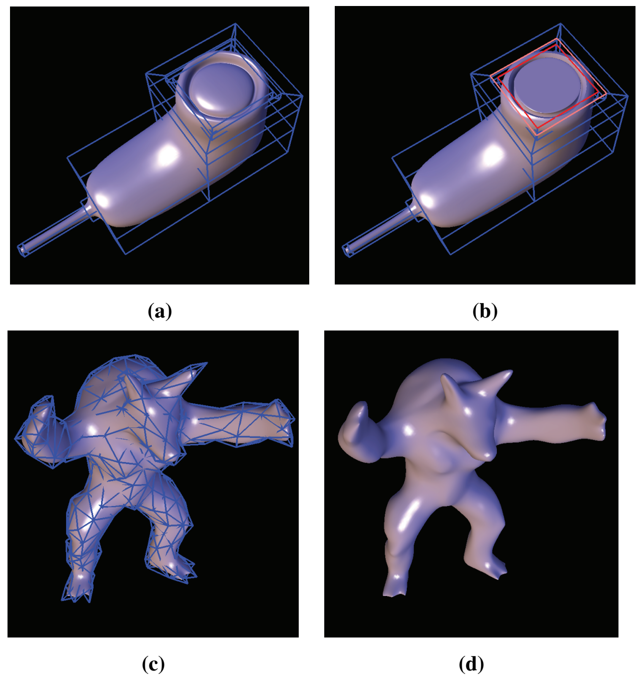


Figure 8: *Our method can handle a quad-only mesh as shown in (a) and produces clean, pleasing subdivision surfaces, also including sharpness as in (b). Similarly, (c) shows the subdivision of a coarse but still complex triangle-only mesh of the armadillo, and (d) shows the same mesh as (c) but with the base mesh wireframe hidden*

cuase a quad on subdivision introduces a new vertex at the face center while a triangle does not. Another challenge was streamling a 1-pass sharp Catmull-Clark subdivision algorithm with a 2-pass quad/tri subdivision algotihm. This was described in detail in Section 4.

7 Conclusion and Future Work

We mention a few future developments that this work could benefit from. We would like to do a curvature analysis of our subdivision surfaces to verify if sharpness affects curvature continuity of the neighboring mesh. However, computing accurate curvatures from polygonal meshes is a challenging problem in itself. We would also have liked to handle meshes with boundaries, that is, meshes that are not closed or water-tight. This is a relatively straightforward extension.

In conclusion, we demonstrated an algorithm to subdivide quad/tri meshes with sharp features. We also designed a user-friendly GUI for interactively editing edge sharpness. The source code is available online fro free non-commerical use [Mehta 2012] and we hope the tool to have practical utility to artists, hobbyists and professional designers or modelers.

8 Acknowledgements

The author would like to thank the course instructor Prof. Carlo Sequin for his valuable guidance and feedback.

References

Catmull, E., and Clark, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6, 350 – 355.

DeRose, T., Kass, M., and Truong, T. 1998. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 85–94.

Doo, D., and Sabin, M. 1978. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design 10*, 6, 356 – 360.

Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., and Stuetzle, W. 1994. Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '94, 295–302.

Jiang, Q., Li, B., and Zhu, W. 2009. Interpolatory quad/triangle subdivision schemes for surface design. *Comput. Aided Geom. Des. 26*, 8 (Nov.), 904–922.

Loop, C. 1987. *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis, University of Utah.

Mehta, S. U., 2012. Quad/tri subdivision with semi-sharp features. `http://www.eecs.berkeley.edu/~sohamum/QTS_Subdiv`. [Online; accessed Dec-2012].

Rademacher, P., 2008. Glui user interface library. `http://www.cs.unc.edu/~rademach/glui`. [Online; accessed Dec-2012].

Reif, U. 1995. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design 12*, 2, 153 – 174.

Schaefer, S., and Warren, J. 2005. On c2 triangle/quad subdivision. *ACM Trans. Graph. 24*, 1 (Jan.), 28–36.

Stam, J., and Loop, C. 2003. Quad/triangle subdivision. *Computer Graphics Forum 22*, 1, 79–85.