COMP5411 Project Report: Catmull-Clark Subdivision

HUANG Junkai, 20676520, jhuangce@connect.ust.hk
Hong Kong University of Science and Technology

1 Introduction

In this project, the Catmull-Clark subdivision algorithm with creases of integer sharpness and the corresponding user interface are implemented. The implementation is built upon the given MeshViewer framework written in C++. Several features of Catmull-Clark subdivision are discussed.

2 Methodology

Catmull-Clark Subdivision

The Catmull-Clark subdivision begins with an arbitrary polyhedron called the control mesh M^0, which may contain any polygonal faces. The control mesh is subdivided to produce the mesh M^1 by splitting each face into a collection of quadrilateral faces. The vertices of M^1 are computed using certain weighted rules, which will be discussed later. Applying the same subdivision procedure on M^1 will produce M^2, then M^3, M^4, ...

One observation is that, each vertex of M^{i+1} can be associated with either a face, an edge, or a vertex of M^i, which are called "face points", "edge points" and "vertex points" correspondingly. With such observation, the Catmull-Clark subdivision procedure, constructing M^{i+1} from M^i, can be decomposed into 4 steps:

1. Create Face Points:
   A face point in M^{i+1} is always positioned at the centroid of its corresponding face in M^i. In my implementation, function catmullclark::get_face_points() will calculate centroids of all the surfaces of source mesh M^i and save all the face points into the vertex list of the new mesh M^{i+1}.
2. Create Edge Points:
   An edge point e^{i+1} in M^{i+1} is computed as
   $$e^{i+1} = \frac{v_1^i + v_2^i + f_1^{i+1} + f_2^{i+1}}{4} \tag{1}$$
   where v_1^i, v_2^i are two end points of the edge e in M^i. f_1^{i+1}, f_2^{i+1} are two adjacent face points in M^{i+1}, which has been calculated in the previous step. (This is the smooth rule. Crease rule will be introduced later.) In my implementation, function catmullclark::get_edge_points() will create all edge points in the new mesh.
3. Create Vertex Points:
   A vertex point v^{i+1} in M^{i+1} is computed as
   $$v^{i+1} = \frac{(n-2)v^i + \frac{1}{n}\sum_j e_j^i + \frac{1}{n}\sum_j f_j^{i+1}}{n} \tag{2}$$
   where v^i is the original vertex in M^i. n is the number of edges incident to vertex v^i. $\frac{1}{n}\sum_j e_j^i$ is the average position of all edge points around vertex v^i, $\frac{1}{n}\sum_j f_j^{i+1}$ is the average position of all face points around v^{i+1}. (This is only the smooth rule. Crease rule will be introduced later.) In my implementation, function catmullclark::get_vertex_points() will create all vertex points in the new mesh.
4. Create half edges and faces:
   Up to now, all the vertices in the new mesh M^{i+1} has been obtained. This step creates half-edges and faces of M^{i+1}. It is worth mentioning that for now the original mesh M^i is still well preserved for reference to help distinguish different kinds of vertices in the M^{i+1}. We can construct all the faces by traversing all the face points in M^{i+1} and add faces around each of the points. One should carefully merge two adjacent faces as they share the same edge. Please refer to catmullclark::get_faces() for detailed implementation.

After 4 steps above, the complete mesh M^{i+1} has been created. One can replace mesh M^i with mesh M^{i+1}.

Creases of Integer Sharpness

To sharpen some edges, an integer attribute "sharpness" is attached to each edge.

Face point calculation are not be affected by sharpness.

When calculating edge point e^{i+1}, suppose sharpness of edge e in mesh M^i is s. If $s = 0$, the smooth rule (1) will be applied. If $s > 0$, then

$$e^{i+1} = \frac{v_1^i + v_2^i}{2} \tag{3}$$

and two subedges created each will have sharpness $s - 1$.

When calculating vertex point v^{i+1}, suppose there are n sharp edges (with $s > 0$) incident to v^i. If $n \le 1$, the smooth rule (2) will be applied. If $n = 2$, suppose the two sharp edges are $v^i v_1^i$, $v^i v_2^i$, then

$$v^{i+1} = \frac{v_1^i + 6v^i + v_2^i}{8} \tag{4}$$

If $n \ge 3$, then

$$v^{i+1} = v^i \tag{5}$$

User Interface

A user interface is implemented to easily assign sharpness to each edge for Catmull Clark subdivision. Here are the steps to use the UI:

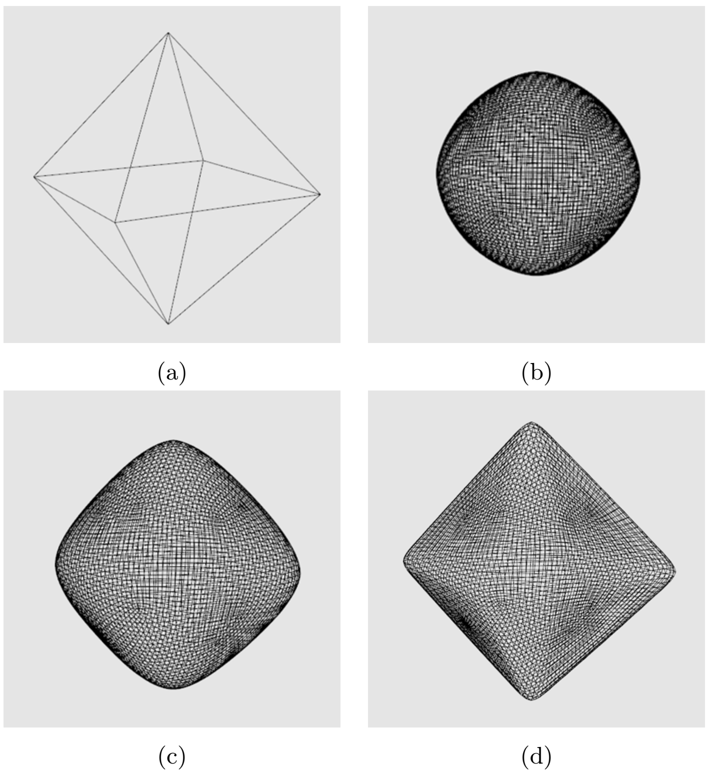

Figure 1: Catmull-Clark subdivision results. (a) is the original octahedron mesh. (b), (c), (d) are results of applying Catmull-Clark subdivision 5 iterations on (a), with all edges having sharpness 0, 1, 3.

1. Open the executable, change "Shading" to "None", tick the "Wireframe" checkbox. (The provided framework doesn't well support to shade a quadrilateral mesh.)
2. Load model octahedron.obj (or other closed mesh).
3. To select an edge, please press down and hold the mouse middle button, cross ONE edge with your mouse pointer, then release the middle button. Then the crossed edge will be selected.
4. To assign sharpness to the selected edge, please input your target sharpness into the "Target Sharpness" textbox. Then click "Set Sharpness" button. (The sharpness of all the edges are initialized to be zero.)
5. Click "CatmullClark" button to run the Catmull-Clark subdivision. One click applys one iteration.

3 Results

We test Catmull-Clark subdivision by applying it iteratively on an octahedron with all edges set to sharpness 0, 1, 3, as shown in Figure 1. One can observe that with sharpness $s = 0$, the mesh quickly converges to a ball after 5 iterations. With $s = 3$, the geometry of output mesh is influenced significantly, which tends to be much more squarish. The sharpness should be less than the number of iterations, in order to avoid extremely sharp edges in the output mesh.

Iterations	Vertices	Edges	Faces	Time(ms)
0	6	12	8	0
1	26	48	24	2
2	98	182	96	5
3	386	768	384	8
4	1538	3072	1536	20
5	6146	12288	6144	70
6	24578	49152	24576	280
7	98306	196608	98304	1106
8	393218	786432	393216	4410
9	1572866	3145728	1572864	17679

Table 1: Output Mesh Statistics and Running Time

4 Discussions

Catmull-Clark subdivision is a robust algorithm that can be applied to any closed mesh. If the original face is an n-gon, the algorithm will divide it into n quadrilateral faces in one iteration. And all the following meshes are quadrilateral meshes. The calculation of this algorithm is simple and efficient, since the position of each vertex can be calculated locally. However, an obvious drawback of this algorithm is that the output mesh is always quadrilateral, which cannot be efficiently rendered and displayed. To render the output mesh, one may divide each quadrilateral face into two triangular faces. But that will change the geometry property of the mesh. So, we should not do this if we wants to do Catmull-Clark subdivision further on the mesh.

Time Analysis

We test Catmull-Clark subdivision by applying it iteratively on an octahedron, and collect execution time along with the number of vertices, edges and faces after each iteration. From Table 1, one can observe that the time t^{i+1} it takes for iteration $i + 1$ is approximately equal to $4 \times t^i$. One intuitive interpretation to this is that, as i goes large, the mesh is always a quadrilateral mesh. Catmull-Clark subdivision algorithm will divide each quadrilateral face into 4 subfaces. The complexity of the new mesh is 4 times of the original one. This also implies that Catmull-Clark subdivision is an linear-time algorithm.

5 Conclusion

Catmull-Clark subdivision takes a coarse mesh (the control mesh) as input and generates as output a new refined mesh containing more faces, edges, and vertices. A smooth limit surface can be obtained by repeating this process on the sequence of new meshes. The shape of the limit surface can be significantly influenced by the sharpness set on the original edges, which can lead to a highly flexible control to the geometry of output model. This efficient and powerful algorithm can be a good tool to create various fine-shaped models.