Introduction

1 Overview

This project implements mesh subdivision based on half-edge data structure, including LOOP subdivision and Modified butterfly subdivision. The project also implements parsing and writing .wrl files and achives visualizing model using OpenGL. In addition, project provide normal of face, and make tests of adding crease.

2. Design

(1) Data structure design:

According to the definition of the half-edge data structure, the three main data structures: struct HalfEdge_vertex (vertex), struct HalfEdge_face (face) struct HalfEdge_halfedge (half edge) are designed.

The three data structures contain the following information (black is a member of the original data structure, and blue is the member that is added for implementing algorithm):

Data structure	Member's name	type	meaning
struct HalfEdge_vertex	x/y/z	double	Vertex coordinates (x, y, z)
	asOrigin	HalfEdge_halfedge*	Point to the halfedge of the vertex
	number	unsigned	The number of the vertex, used to write files&easy to DEBUG
	adjust	HalfEdge_vertex*	Point to the after-adjusting vertex
	asInsert	HalfEdge_halfedge*	point to the halfedge which vertex is inserted. Working in remesh
	beenOrigined	bool	Check if the vertex is already be an origin vertex of an halfedge.So don't have to assign it every time.

struct HalfEdge_face	boundary	HalfEdge_halfedge*	Point to one of the halfedge sides surrounding the face
	normal	HalfEdge_normal*	Point to the normal of the face
struct HalfEdge_halfedge	origin	HalfEdge_vertex*	Point to the original point of the halfedge
	nextEdge	HalfEdge_halfedge*	Point to the next halfedge of the halfedge
	preEdge	HalfEdge_halfedge*	Point to the pre halfedge of the halfedge
	twin	HalfEdge_halfedge*	Twin pointing to the halfedge
	incidentFace	HalfEdge_face*	Point to the face surrounded by the halfedge
	number	unsigned	The number of the halfedge, easy to DEBUG
	insert	HalfEdge_vertex*	Point to the vertices inserted in this halfedge
	isCrease	bool	Judging whether it is a crease
	CreaseChecked	bool	Determine whether it has been checked and avoid repeated inspections

In addition, in order to facilitate the parameter transfer and iteration, the data structure of struct HalfEdge_mesh is designed. In order to render the face in OpenGL and determine Crease, adds the face normal data structure struct HalfEdge_normal:

Data structure	Member's name	type	meaning
HalfEdge_normal	x/y/z	double	Vector (x, y, z)
	number	unsigned	The number of the normal, ease for DEBUG
HalfEdge_mesh	face	HalfEdge_face	The face collection of the mesh

	halfedge	HalfEdge_halfedge	The halfedge collection of the mesh
	vertex	HalfEdge_vertex	The mesh vertex collection
	normal	HalfEdge_normal	The normal collection of the mesh

(2) Algorithm design

Both subdivision algorithms can be roughly divided into 3 steps: 1)determining the coordinates of the inserted vertex.2)Determining the position of the original vertex.3)Remesh.

1)Determine where to insert the vertex: Calculate the vertex coordinates to be inserted by the formula, generate a new vertex collection, and associate it to the old halfedge (equivalent to cutting the old halfedge).

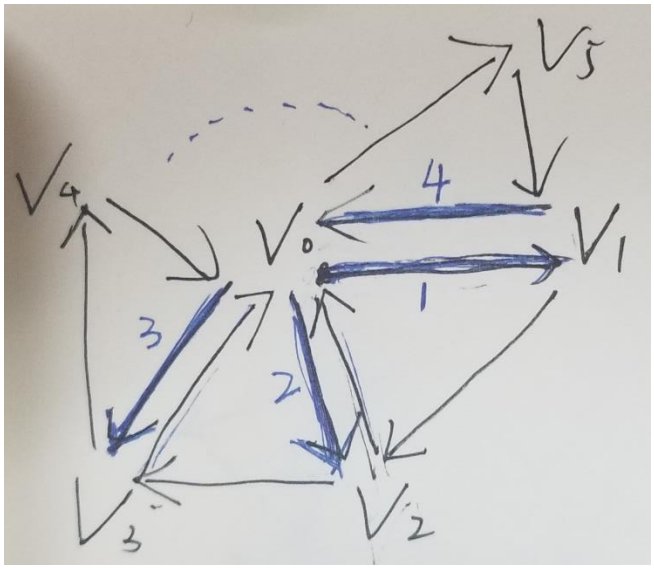
2)Determine the position of the original vertex: the loop algorithm needs to re-adjust the coordinates of the old vertex, and the butterfly algorithm copies the old vertex coordinates.

3)Remesh: Combine existing halfedge collections and vertex collections to build new faces and grids.

In addition to calculating coordinates, the two most important parts are calculating valence and remesh for the new vertex. Both the loop algorithm and the butterfly algorithm need to calculate valence.

a. Find neighbor.

In my experiment (in LoopSubdivision.cpp or ModifyButterflySubdivision.cpp), the method of calculating valence is as follows:

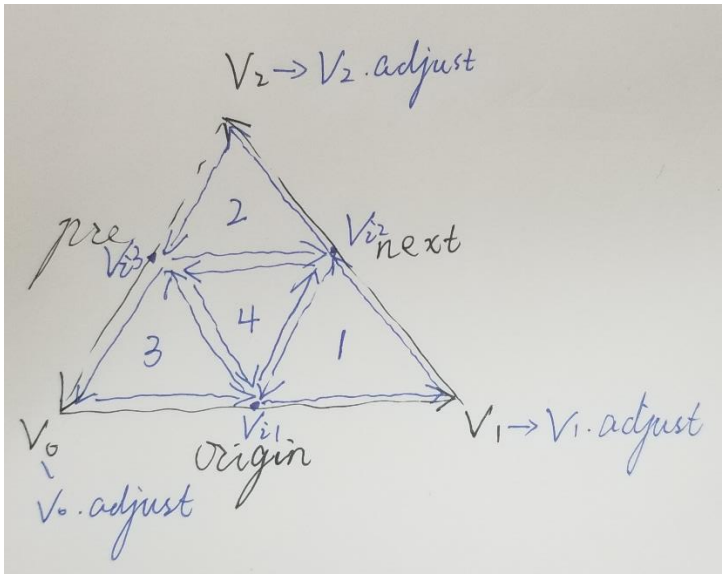


Taking halfedge(v0, v1) as an example, let HalfEdge_halfedge* find point to the halfedge, add the next origin of the halfedge pointed by find, that is, v1 to the neighbor of v0, and check if the next twin of the halfedge pointed to by find exists. Then let find

point to it, and continue to loop until it returns to halfedge (v0, v1). In this process, if the next twin of the halfedge pointed to by find does not exist, it reaches an end, and you can choose whether it is only out/in degrees. The neighbor points calculate valence. Then use a find similar to find to point to halfedge(v1, v0) and calculate from the other direction.

b. Remesh.

The process of generating the insertion point and moving the original vertex is associated with the inserted new vertex and the old halfedge, so it can actually be associated with the situation as shown in remesh.



The original face is then divided into four according to the order shown, and the new face and halfedge are generated in the order shown.

3. Implementation and improvement

(1)read / write files

The read and parse files learned from the IfcConvert source code used in my previous study, which reads the number of rows, reads the header, and writes model information to the data structure when parsing the file.

When parsing the .wrl file, the information is checked first. However, this project can only check the files under the #VRML V2.0 utf8 (Converted to ASCII) standard, and only for file contains point and surface information.

When parsing, an initial HalfEdge_mesh is created. By reading the vertex information, the coordinates of all vertex in the vertex set are obtained. When reading the surface information, each face[a, b, c] (a, b, c is the number of the vertex in the vertex collection) Can lead to a halfedge set {ab, bc, ca}, so you can fill in the vertex collection, halfedge collection and face collection part of the information, after the entry is completed in the entire table to find the halfedge's twin, face normal.

For write file, write the header first, then traverse the vertex collection to write the coordinates of all points, then traverse the face collection, and write the three vertex numbers in the face collection in order.

(2)Iteration

At the beginning of the implementation, I only used the three data structures of halfedge, vertex, and face, so I have to write a lot of code every time I pass the parameters. I thought about using mesh to include the three, and passing the current mesh when iterating. The address of the next iteration is meshed, and the mesh is obtained after subdivision.

(3)Visualization

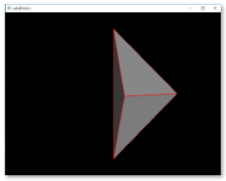
Implemented with OpenGL, the triangle vertex information is passed when the grid is drawn, and the normal information is added when the surface is drawn.

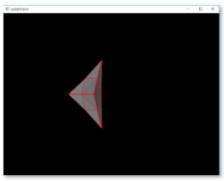
4. Experimental results and analysis

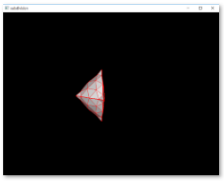
I have experimented with the experiment model multiple times, but limited by the container size, the tetrahedron and the cube can be iterated 6 times, T-Shape can iterate 5 times, igea_ascii_Simp1000 can iterate 3 times, and igea_ascii_Simp4000 can iterate 2 times. The result is as follows: (also can see under the ./result folder of the project)

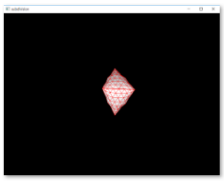
(1) tetrahedron

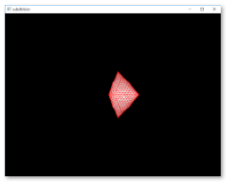
Loop subdivision

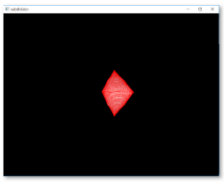

Tetrahedron_loop_0.png

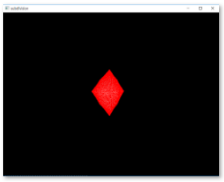

Tetrahedron_loop_1.png


Tetrahedron_loop_2.png

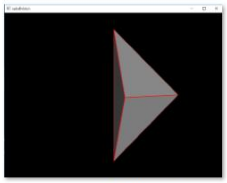

Tetrahedron_loop_3.png

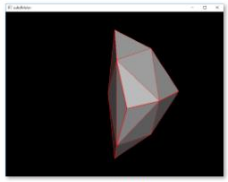

Tetrahedron_loop_4.png

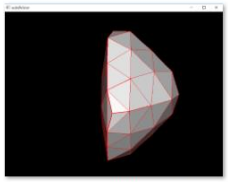

Tetrahedron_loop_5.png

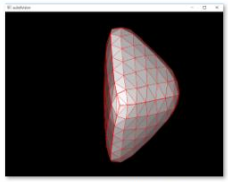

Tetrahedron_loop_6.png

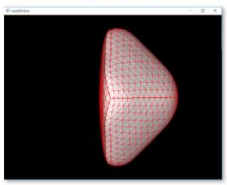
Modified-butterfly subdivision

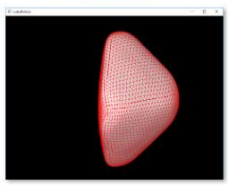

Tetrahedron_modifiedButterfly_0.png

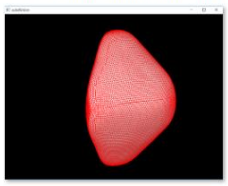

Tetrahedron_modifiedButterfly_1.png


Tetrahedron_modifiedButterfly_2.png


Tetrahedron_modifiedButterfly_3.png

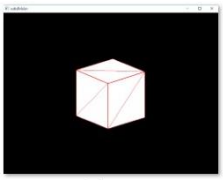

Tetrahedron_modifiedButterfly_4.png

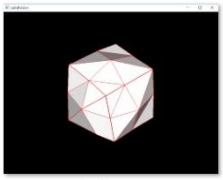

Tetrahedron_modifiedButterfly_5.png

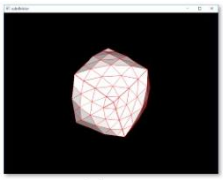

Tetrahedron_modifiedButterfly_6.png

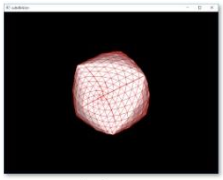
(2)Cube

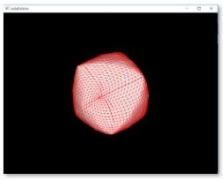
Loop subdivision

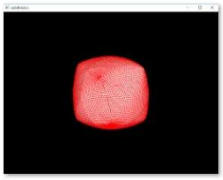

Cube_loop_0.png

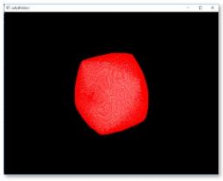

Cube_loop_1.png


Cube_loop_2.png

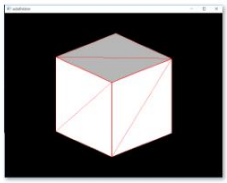

Cube_loop_3.png

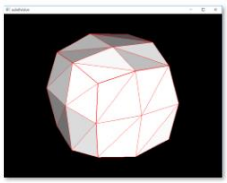

Cube_loop_4.png

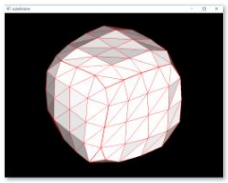

Cube_loop_5.png

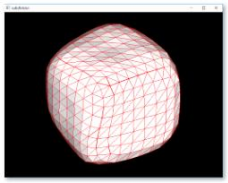

Cube_loop_6.png

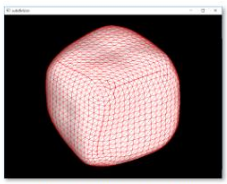
Modified-butterfly subdivision

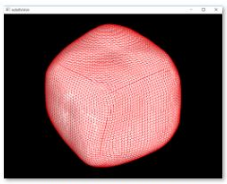

Cube_modifiedButterfly_0.png

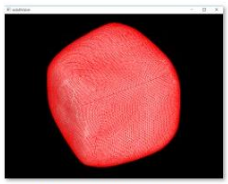

Cube_modifiedButterfly_1.png


Cube_modifiedButterfly_2.png


Cube_modifiedButterfly_3.png

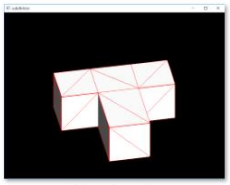

Cube_modifiedButterfly_4.png

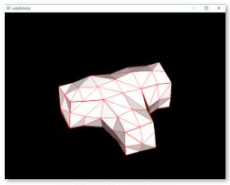

Cube_modifiedButterfly_5.png

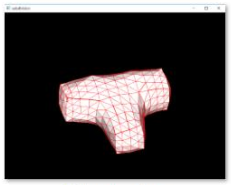

Cube_modifiedButterfly_6.png

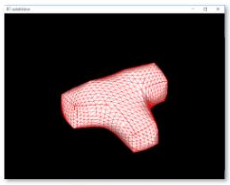
(3)T-Shape

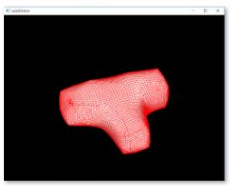
Loop subdivision

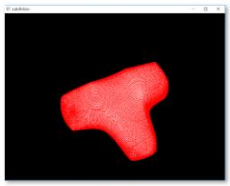

T-Shape_loop_0.png


T-Shape_loop_1.png

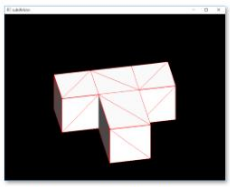

T-Shape_loop_2.png

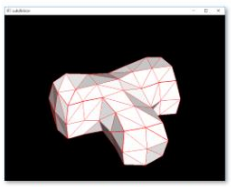

T-Shape_loop_3.png

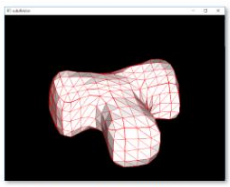

T-Shape_loop_4.png

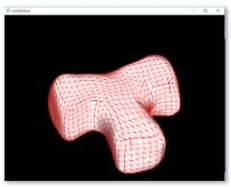

T-Shape_loop_5.png

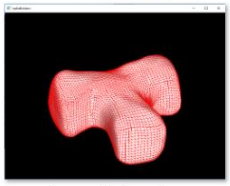
Modified-butterfly subdivision

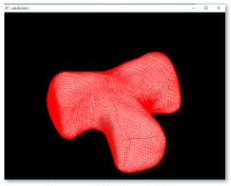

T-Shape_modifiedButterfly_0.png


T-Shape_modifiedButterfly_1.png

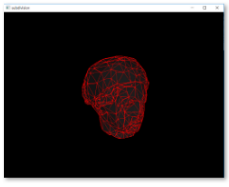

T-Shape_modifiedButterfly_2.png

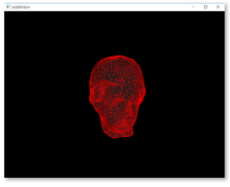

T-Shape_modifiedButterfly_3.png

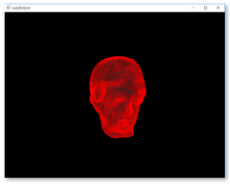

T-Shape_modifiedButterfly_4.png

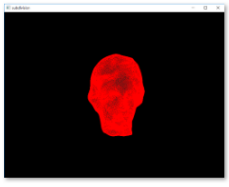

T-Shape_modifiedButterfly_5.png

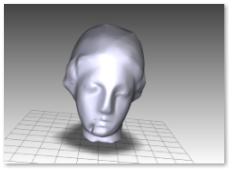
(4) igea_ascii_Simp1000
Loop subdivision

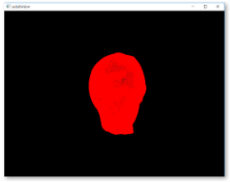

igea_ascii_Simp1000_loop_0.png

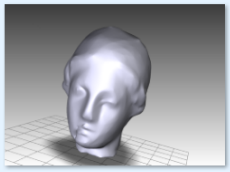

igea_ascii_Simp1000_loop_1.png


igea_ascii_Simp1000_loop_2.png

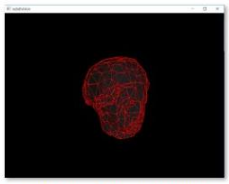

igea_ascii_Simp1000_loop_3(1).png

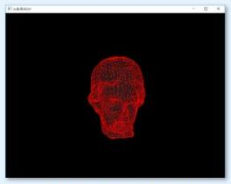

igea_ascii_Simp1000_loop_3(2).png

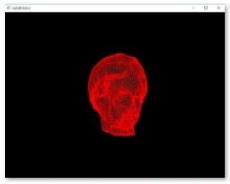

igea_ascii_Simp1000_loop_4(1).png

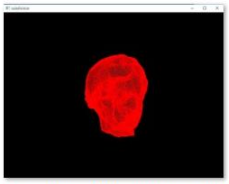

igea_ascii_Simp1000_loop_4(2).png

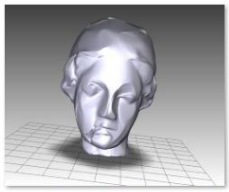
Modified-butterfly subdivision


igea_ascii_Simp1000_modifiedButterfly_0.png

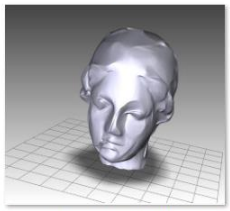

igea_ascii_Simp1000_modifiedButterfly_1.png


igea_ascii_Simp1000_modifiedButterfly_2.png

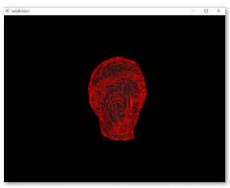

igea_ascii_Simp1000_modifiedButterfly_3(1).png

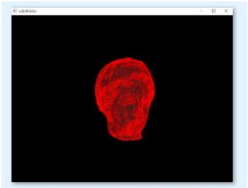

igea_ascii_Simp1000_modifiedButterfly_3(2).png

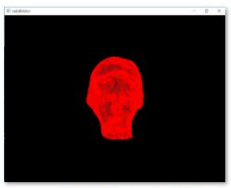

igea_ascii_Simp1000_modifiedButterfly_4(1).png

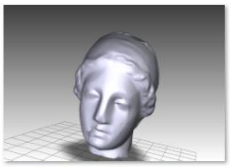

igea_ascii_Simp1000_modifiedButterfly_4(2).png

(5) igea_ascii_Simp4000
Loop subdivision

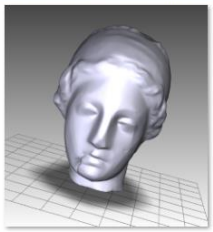

igea_ascii_Simp4000_loop_0.png


igea_ascii_Simp4000_loop_1.png

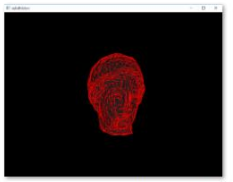

igea_ascii_Simp4000_loop_2(1).png

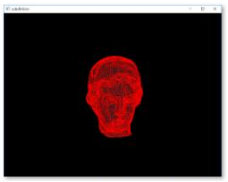

igea_ascii_Simp4000_loop_2(2).png

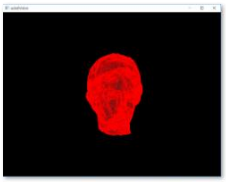

igea_ascii_Simp4000_loop_3(1).png

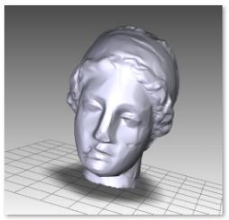

igea_ascii_Simp4000_loop_3(2).png

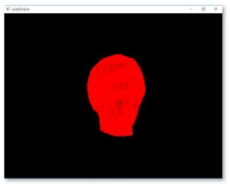
Modified-butterfly subdivision

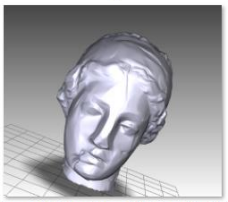

igea_ascii_Simp4000_modifiedButterfly_0.png


igea_ascii_Simp4000_modifiedButterfly_1.png


igea_ascii_Simp4000_modifiedButterfly_2(1).png


igea_ascii_Simp4000_modifiedButterfly_2(2).png


igea_ascii_Simp4000_modifiedButterfly_3(1).png


igea_ascii_Simp4000_modifiedButterfly_3(2).png

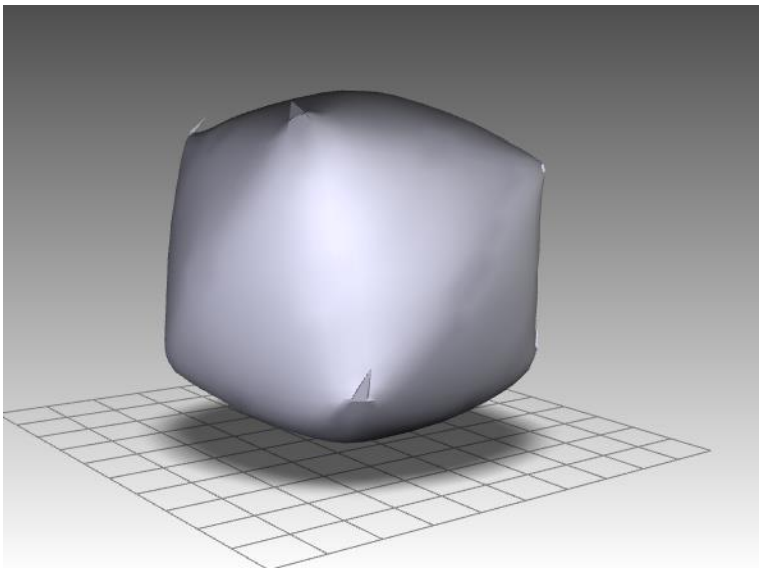
We can observed that the loop algorithm has a tendency to shrink as a whole due to the adjustment of the old and new vertex, therefore, model looks more smooth.

The modified butterfly algorithm expands outward as the whole, because vertex position is retained, and the overall performance is relatively large at the end. Inside is depressed.

In addition, I also tried to specify the crease in the experiment. Boundary specification is very easy: a halfedge has no twin. For crease, I initially specify the crease of the model in each iteration.

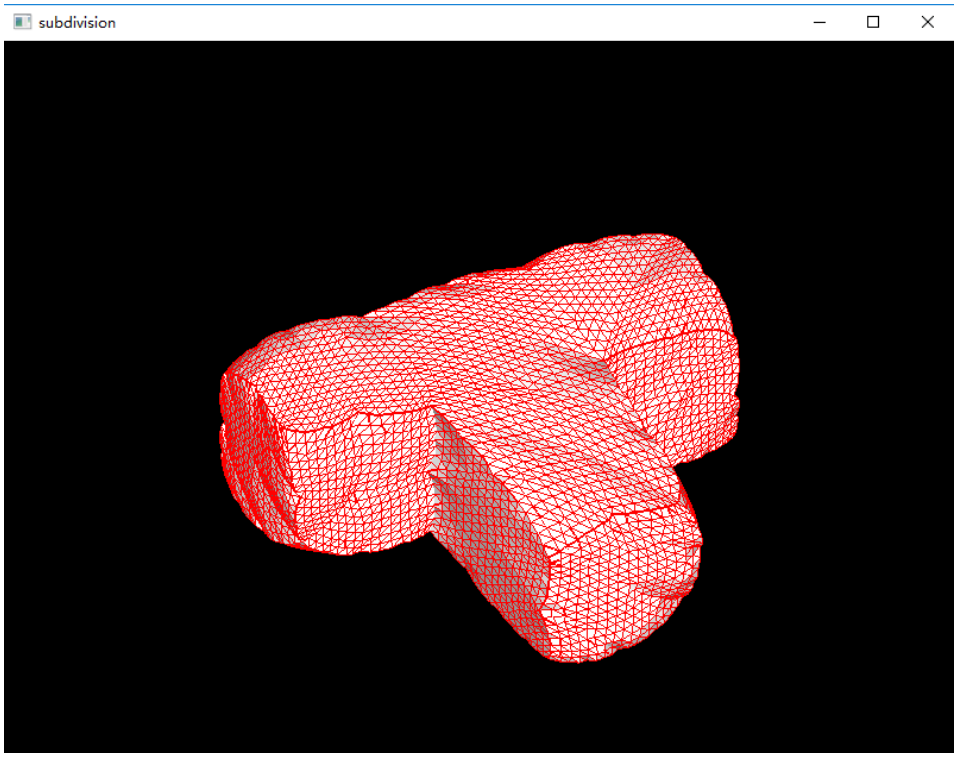
```
//assign edge/cease for tetrahedron,Cube,T-Shape
/*for tetrahedron
if (inputFileNumber == 1) {
    for (auto it = (iterationMesh[0]).halfedge.begin(); it != (iterationMesh[0]).halfedge.end(); it++)
        (*it).isCrease = true;
}
/*for cube
if (inputFileNumber == 2) {
    int i = 0;
    for(auto it = (iterationMesh[0]).halfedge.begin(); it != (iterationMesh[0]).halfedge.end(); it++){
        if (i % 3 != 2) (*it).isCrease = true;
    }
    }
/*for T-Shape, too hard to assign*/
```

if I assign all halfedge as crease, it went like this(Loop subdivision):

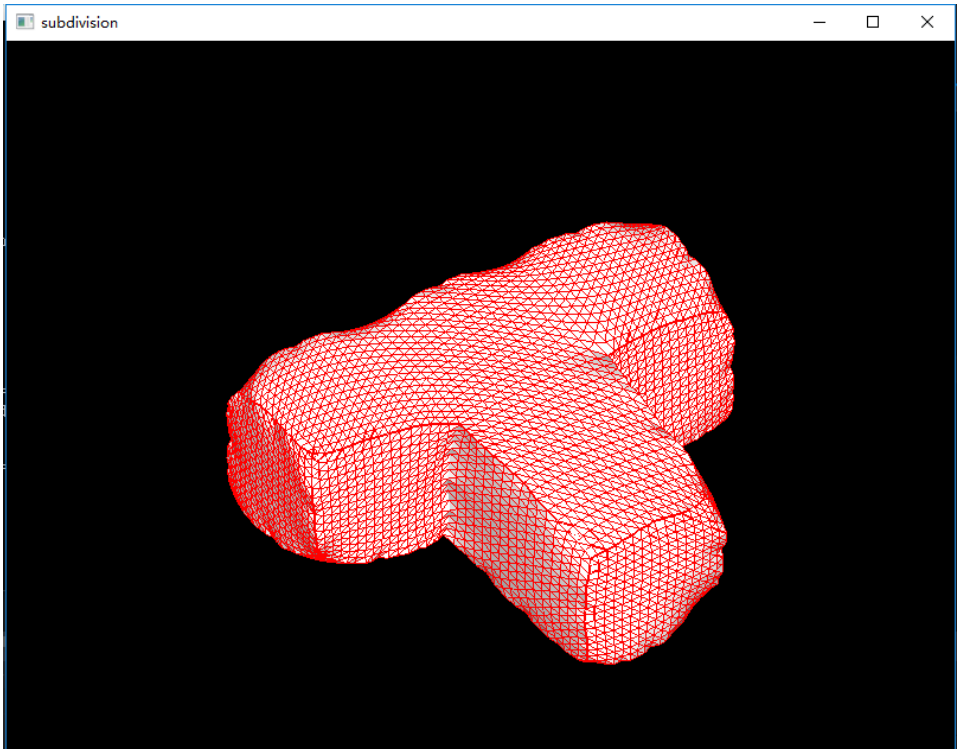


It will produce very sharp vertices, and there is no way to specify boundaries every time as the model is complex.

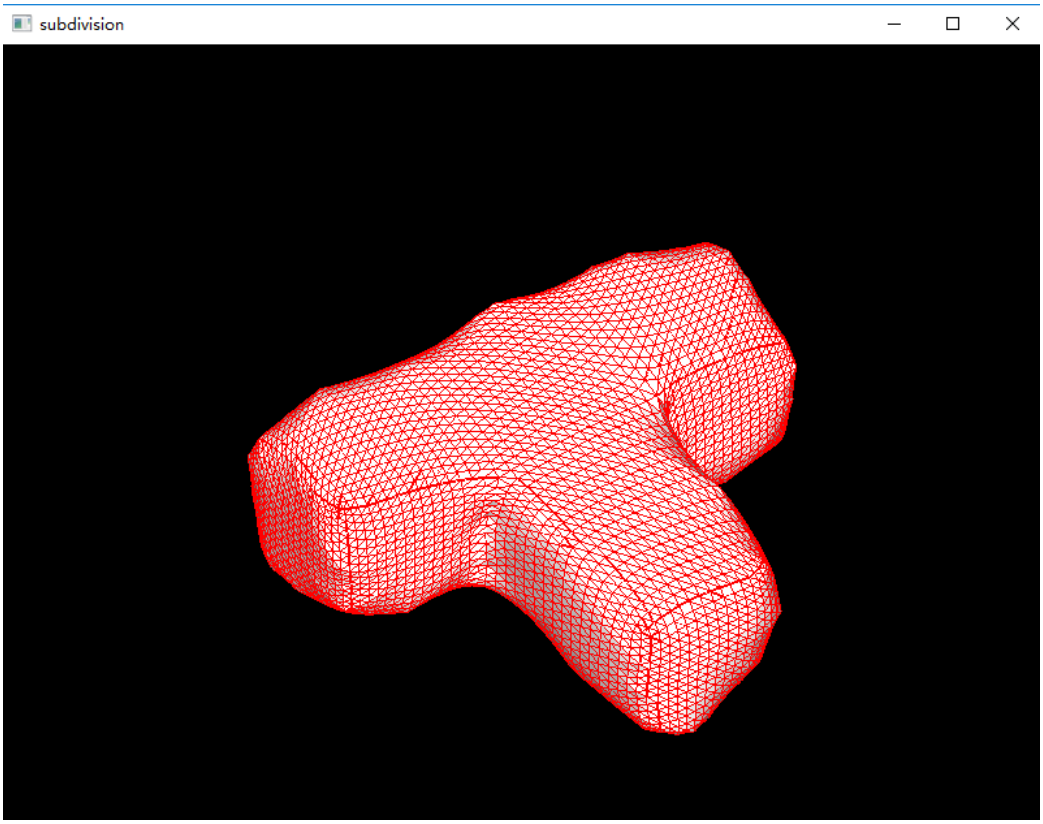
After that, I tried to add a normal calculation to the project, and calculate the angle by the angle between the two sides of the normal. The artificially specified angle is greater than a certain angle θ is crease. The following are some attempts:
(T-Shape as an example, loop algorithm, 4 iterations)
θ>5:



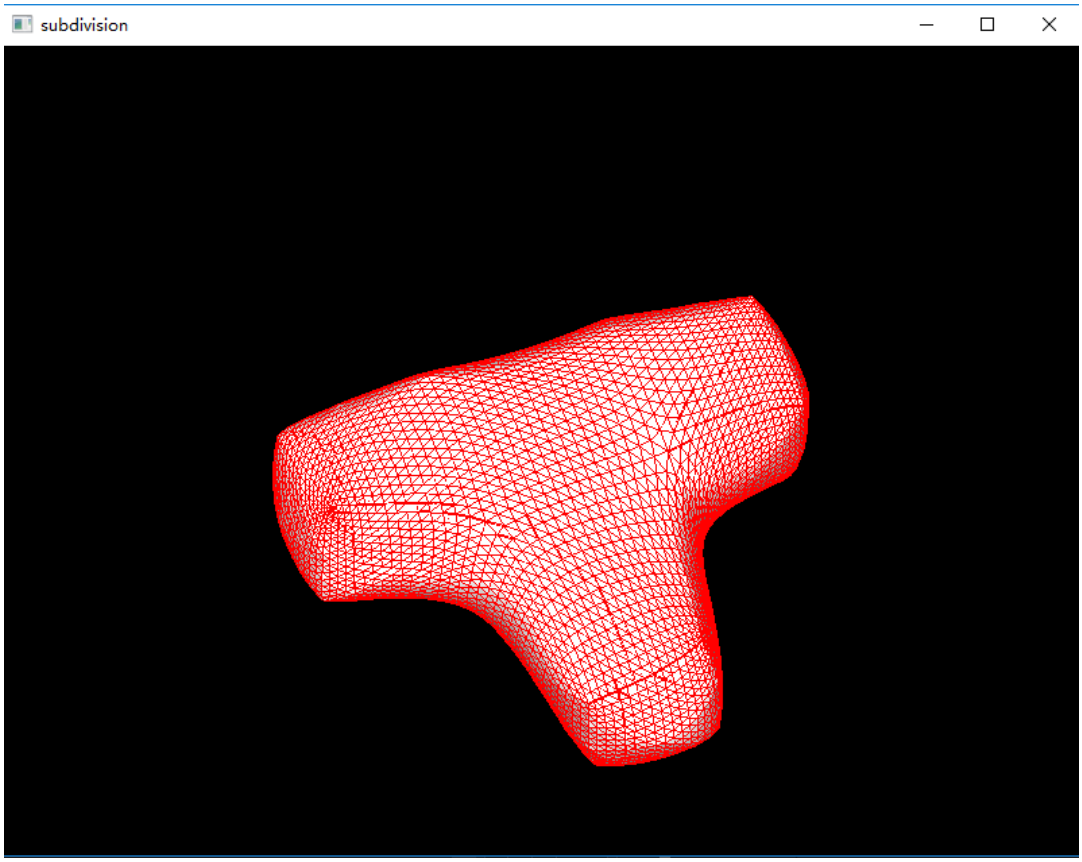
θ>60



θ>90

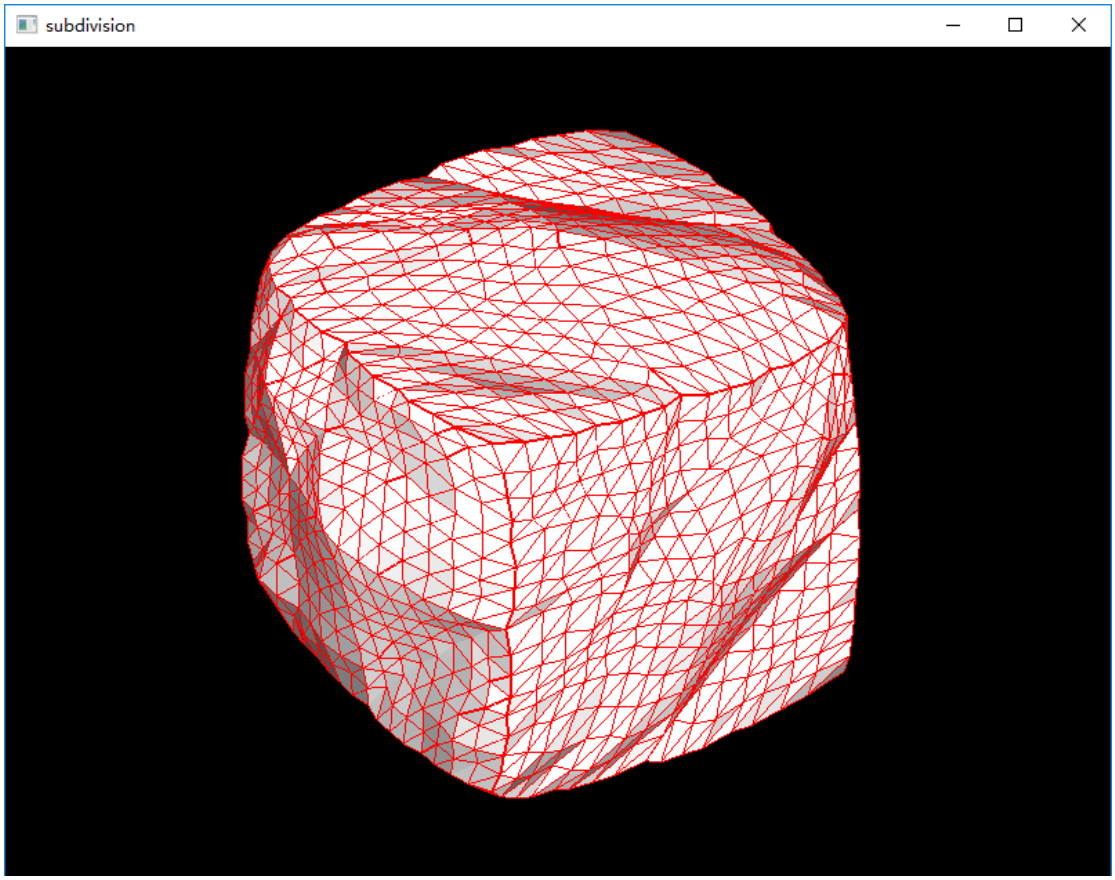


θ>150

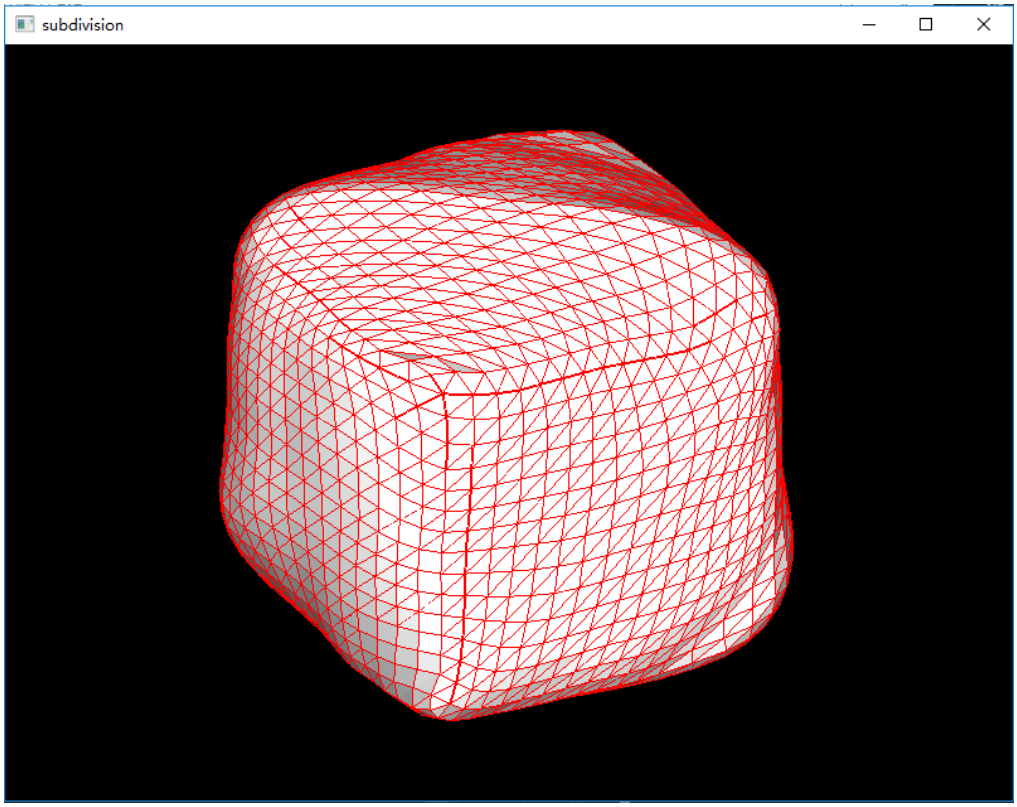


(Cube as an example, Modified butterfly algorithm, 4 iterations)

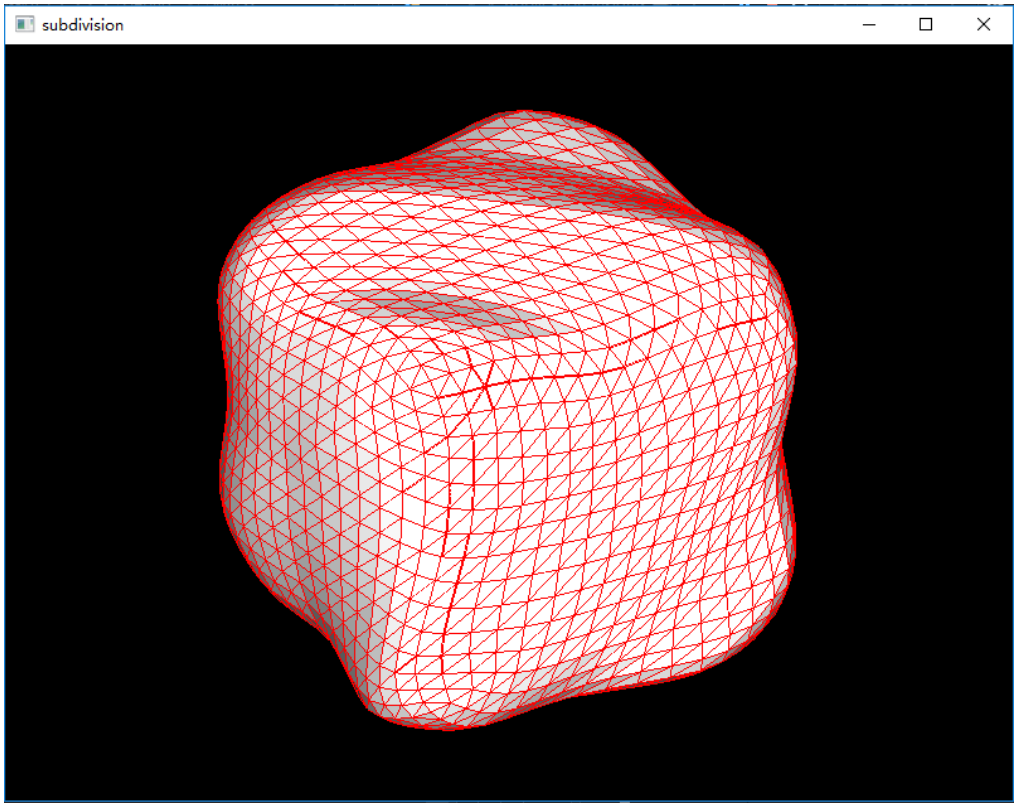
θ>5:



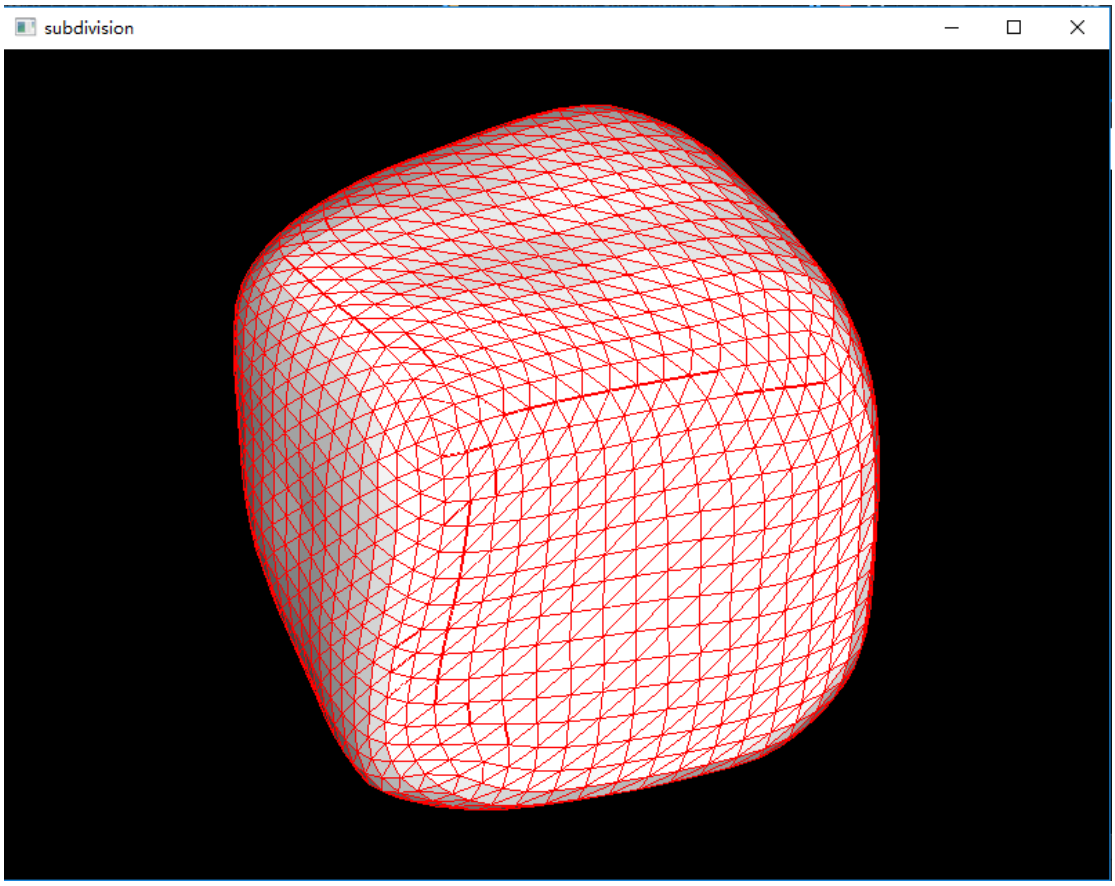
θ>60:



θ>90:



θ>150:



5. Project configuration

The angle of the crease can be adjusted in Halfedge_structure.cpp.

The OpenGL related libraries and header files are in the project directory, where GL.h is the system's own header file.