CS554 Geometric Modeling

Project1 Report

Tianle Yuan (933946576)

January 22, 2021

1. (Corner) Construct the corner list for a 3D model.

The structure of the corner list can be described as below:

```
class Corner {
public:
    double c;   //corner's angle value
    int index;  //corner's index

    Corner* o;  //opposite corner
    Corner* p;  //privious corner
    Corner* n;  //next corner
    Vertex* v;  //corner's vertex
    Edge* e;    //corner's edge
    Triangle* t;

    void* other_props;
};
```

In which, **c** is the angle of corresponding corner with degree measure. **index** is the position in the corner list that the corner is saved in. **o** is the opposite corner of the original corner. **p** is the previous corner of the original corner in clock wise direction. **n** is the next corner of the original corner in clock wise direction. **v** is the vertex that the corner is locating. **e** is the opposite edge that the corner' vertex is facing. **t** is the triangle that contains the corner.

In order to check whether my code's structure is correct, I have set those equations for testing:

Test 1: For some vertex, traverse all of it's interior angle to prove the equation shows below:

$$(C(v).(p.o.p)^{ntris(v)}).c = C(v).c$$

Test 2: For some corner, test the reflexivity of operation named "next corner":

$$(C.n.n.n).c = C.c$$

Test 3: For some corner, test the reflexivity of operation named "previous corner":

$$(C.p.p.p).c = C.c$$

Test 4: For some corner, test "next corner" and "previous corner" by considering:

$$(C.n.p).c = C.c = (C.p.n).c$$

Test 5: For some corner, test the reflexivity of operation named "edge of corner":

C.e= C.o.e

Test 6: For some corner, test the reflexivity of operation named “opposite corner”:

(C.o.o).c = C.c

Taking **bunny** model as an example, we chose the 208th triangle and the 2nd corner of the triangle for testing. The testing result has been shown below for proving the correctness of my Corner structure:

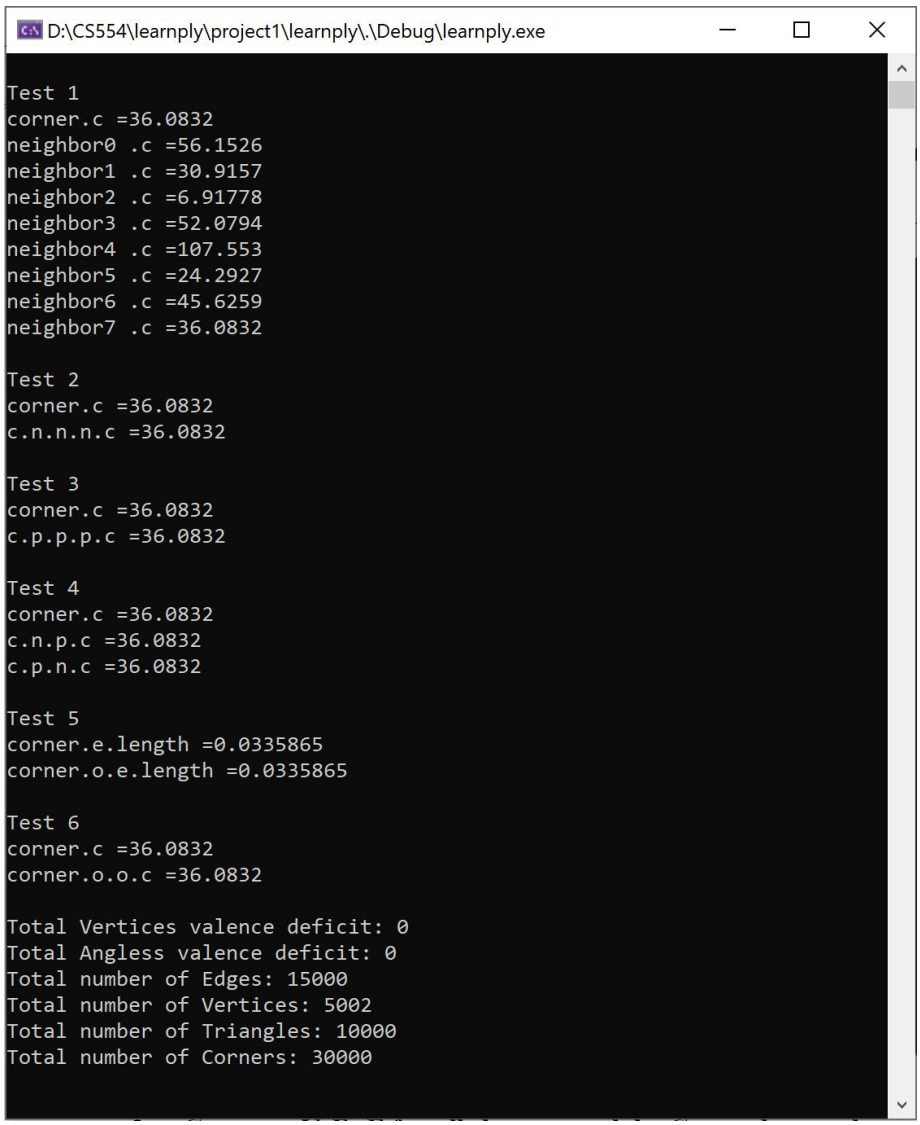


Figure 1. Test result of Bunny by testing the 208th triangle and the 2nd corner

By looking at the result of the running code, the picture proves that my **Corner** structure passed all the tests and satisfy the modeling requirement of **Corner** class.

2. Compute V-E+F for all the test models. Count the number of handles in these models. Include images that show where these handles are. A handle is also sometimes called a tunnel depending on its configuration relative to the rest of the shape. What relationship do you find? Can you provide some intuition about it?

For all the model, to get the number of vertices and faces are easy, because they are shown in the .ply file. Thus, to calculate V-E+F we only need the number of edges, which can be gotten from the member named "nedges" in Polyhedron class. The calculate result are shown below as a table:(note: Grey data means the model is non-compilable)

Model	Edges	V-E+F	Result
bunny	15000	5002-15000+10000	2
dodecahedron	/	20-?+12	/
dragon	30000	10000-30000+20000	0
feline	15000	4998-15000+10000	-2
happy	30000	9990-30000+20000	-10
hexahedron	/	8-?+6	/
icosahedron	30	12-30+20	2
octahedron	12	6-12+8	2
sphere	12288	4098-12288+8192	2
tatrahedron	/	4-?+4	/
torus	13824	4608-13824+9216	0

Table 1. Euler characteristic (V-E+F) calculation results of all models

By loading all of the ply file of models, the data collection of the number of handles has been listed below: (note: Grey data means the model is non-compilable)

Model	Number of handles
bunny	0
dodecahedron	/
dragon	1
feline	2
happy	5
hexahedron	/
icosahedron	0
octahedron	0
sphere	0
tatrahedron	/

torus	1

Table 2. Counting of handles of all models

The models' existing handles has been show below:

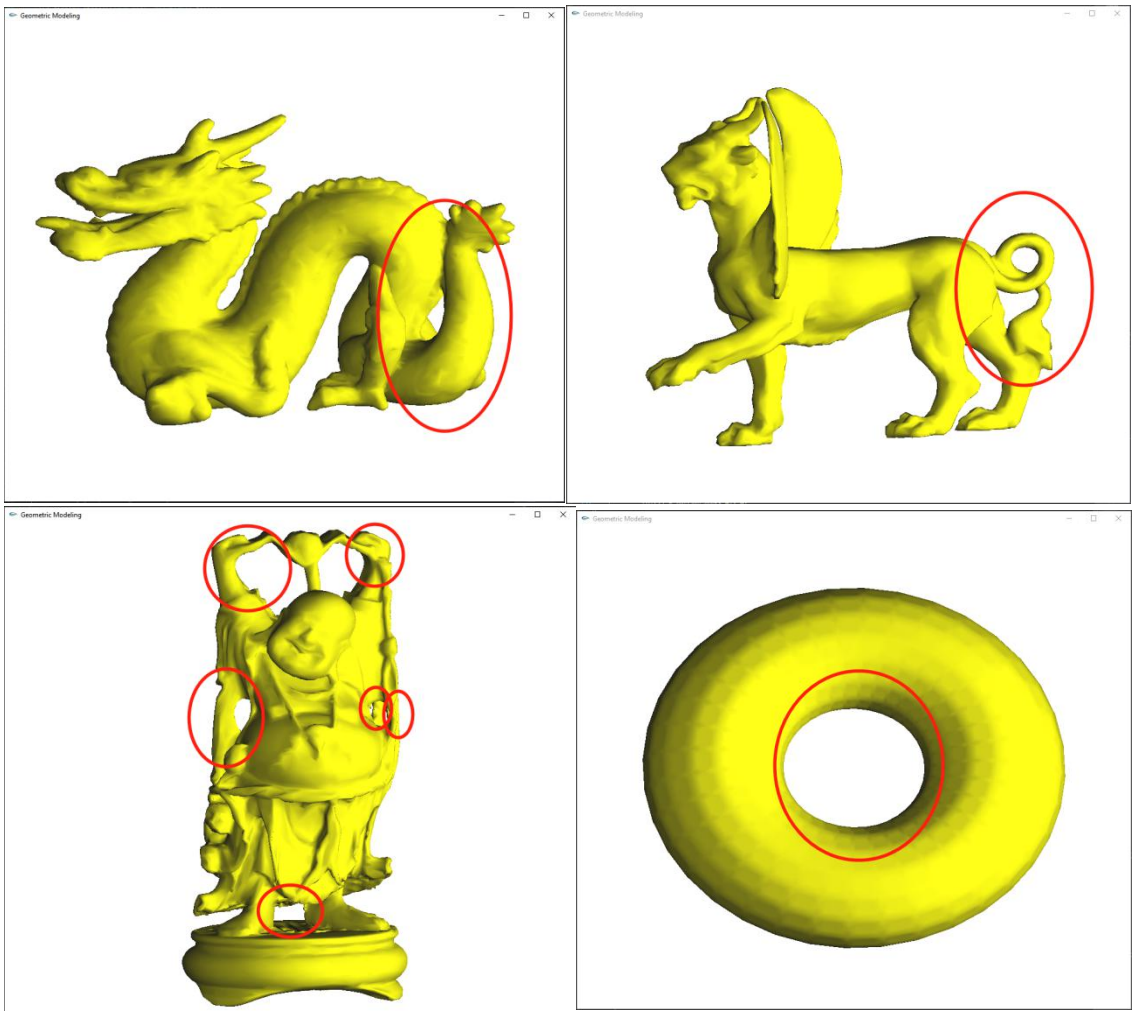


Figure 2. Models with handles

Conclusion: the relationship I find is that: the result of the "V-E+F" depends on how much "hole" (the professional word as the question mentioned is Tunnel) the shape it has. According to the result from Table1 and the handles' data from Table2, my assumption is that:

V(M)-E(M)+F(M) = 2 - 2*(Number of tunnels (M)) .

in which **M** means "model". Taking the simple shape as an example, such as **bunny**, **icosahedron**, and **octahedron**, their results are all 2 because they do not have any hole in the shape. But when we see **torus**, it has one hole so the result is 2-2*1 = 0 which is equals to the result of V-E+F. And also for **happy**, because there are 6 holes, according to my assumption the result should be 2-2*6 =10. This result also coincides with the calculated result of V-E+F.

a. Detect irregular vertices in triangle meshes. (A vertex is regular if its valence is six; otherwise it is irregular.) Color code them based on the valence deficit.

Here, to color code the valence deficit of the irregular vertices, I use: green to represent vertex with 5 valence deficit, yellow to represent vertex with 4 valence deficit, red to represent vertex with 3 valence deficit, purple to represent vertex with 2 and 1 valence deficit, and blue to represent vertex with negative valence deficit.

Taking the bunny model as an example, the result picture can be show below when draw the model with GL_POINTS and GL_TRIANGLES:

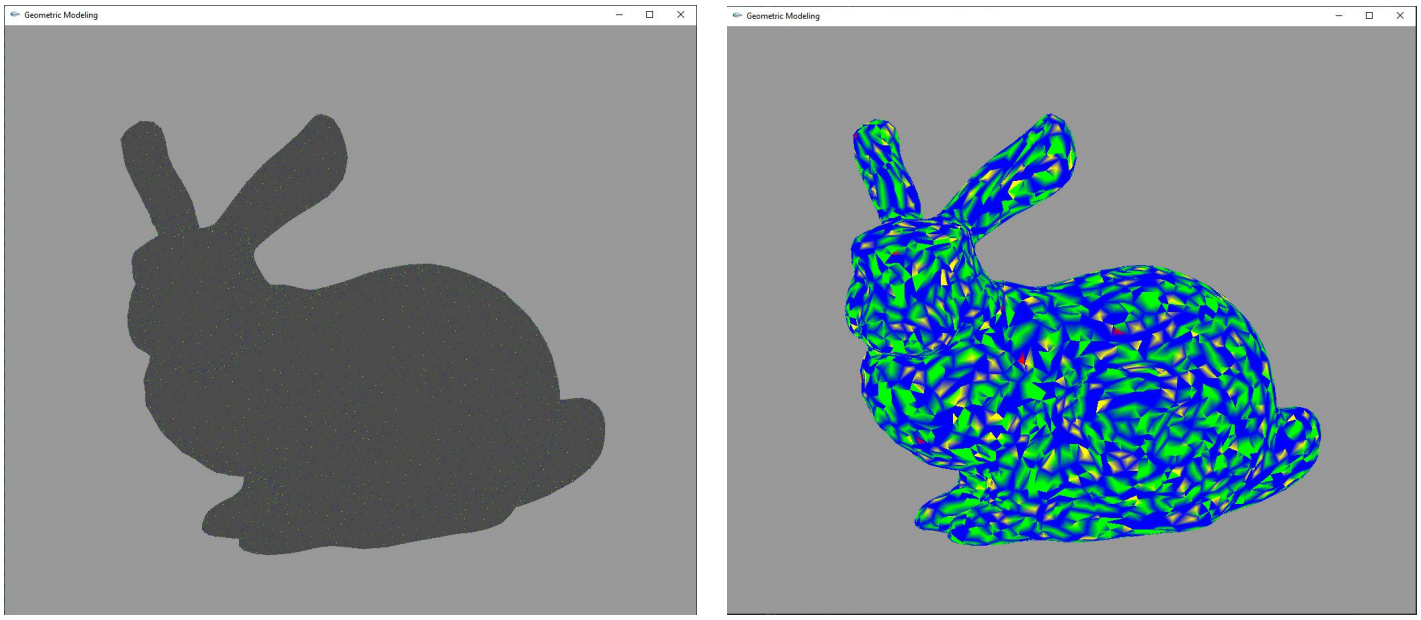


Figure 3. Irregular vertices of Bunny model

The conclusion is that: most of the irregular vertices of Bunny model have 4, 5, or negative valence deficit by observing that most of the color points are green, yellow and blue points. There are small part of vertices have valence with 3. The right picture colored model is more clear than the left vertex one showing the valence in a macro scale.

Actually, there are no vertices have valence with 2, 1and 0. The reason might because that the shape is constructed by closed mesh, even for the shape which has several tunnels like **happy** model. If a mesh has some vertices with valence of 2, 1and 0, it will change the topological connections of the model.

b. Compute the total valence deficit for each model. Is this quantity related to some other quantity that you know of? Prove this relationship between the total valence deficit and the known quantity.

The counting algorithm for the total valence deficit can be shown below:

```
for (i = 0; i < this_poly->nverts; i++) {
    int count = this_poly->vlist[i]->ntris;
    if ( count == 1) { glColor3f(1.0, 0.0, 1.0); count_valence += 5;}
    else if (count == 2) { glColor3f(1.0, 0.5, 0.0); count_valence += 4;}
    else if (count == 3) { glColor3f(1.0, 0.0, 0.0); count_valence += 3;}
    else if (count == 4) { glColor3f(0.0, 1.0, 0.0); count_valence += 2;}
    else if (count == 5) { glColor3f(0.0, 0.0, 1.0); count_valence += 1;}
    else if (count == 7) { glColor3f(0.0, 1.0, 1.0); count_valence -= 1; }
    else if (count == 8) { glColor3f(0.0, 1.0, 1.0); count_valence -= 2; }
    else if (count == 9) { glColor3f(0.0, 1.0, 1.0); count_valence -= 3; }
    else if (count == 10) { glColor3f(0.0, 1.0, 1.0); count_valence -= 4; }
    else if (count == 11) { glColor3f(0.0, 1.0, 1.0); count_valence -= 5; }
    else if (count == 12) { glColor3f(0.0, 1.0, 1.0); count_valence -= 6; }
    else if (count == 13) { glColor3f(0.0, 1.0, 1.0); count_valence -= 7; }
    else if (count == 14) { glColor3f(0.0, 1.0, 1.0); count_valence -= 8; }
    glVertex3d(this_poly->vlist[i]->x, this_poly->vlist[i]->y,
this_poly->vlist[i]->z);
    }
return count_valence;
```

By adding all the valence deficit from all the colorful color vertices, collecting all the result from different model, we can get a list shown below: (note: Grey data means the model is non-compilable)

Model	Number of vertices	Number of faces	Total valence deficit
bunny	5002	10000	12
dodecahedron	20	12	/
dragon	10000	20000	0
feline	4998	10000	-12
happy	9990	20000	-60
hexahedron	8	6	/
icosahedron	12	20	12
octahedron	6	8	12
sphere	4098	8192	12
tatrahedron	4	4	/
torus	4608	9216	0

Table 3. Total valence deficit of all models

The relationship that I have found between quantities is that: The total valence deficit of a model depends on how many **tunnels** it has. So the mathematical equation that I got from the knowing quantities is:

Total Valence deficit (M) = 12 - 12*(Number of tunnels (M)) = 6*(V(M)-E(M)+F(M)) . in which **M** means "model". Taking the Happy model as an example, the Euler characteristic number of the model is -10. According to the equation, the total valence deficit of Happy model should be 6*(-10) = -60. The result is exactly what we calculated from the table.

c. Compute the total angle deficit for each vertex.

To calculate the total angle deficit for a vertex, we can follow this way:

(1) To traverse each vertex, we are in the for loop:

```
for (i=0; i< this_poly->nverts; i++){
    ...
    Vertex *temp_v = this_poly->verts[i];
...}.
```

(2) Then for each vertex, we go to scan the member named triangle with it's angles, and sum them together. In the algorithm shown below, "`VertAngleValence`" represents the angle valence for each vertices:

```
for (i=0; i< this_poly->nverts; i++){
    for (j=0; j< this_poly->verts[i]->ntris; j++){
        for (k=0; k < 3; k++){
         if (this_poly->verts[i]->tris[j]->verts[k] == this_poly->verts[i])
            sumangle += this_poly->verts[i]->tris[j]->angle[k];
        }
    float VertAngleValence = 360.0 - sumangle;
    }
...}.
```

d. Compute the total angle deficit for all the vertices in the mesh? How is this quantity related to some known quantity? Prove this relationship.

Adding one line for the algorithm in question c. we can get:

```
for (i = 0; i < this_poly->nverts; i++) {
    float sumangle = 0;
    for (j = 0; j < this_poly->vlist[i]->ntris; j++) {
        for (int k = 0; k < 3; k++) {
         if (this_poly->vlist[i]->tris[j]->vlist[k] == this_poly->vlist[i])
                sumangle += this_poly->vlist[i]->tris[j]->angle[k];
        }
    }
    count_angle += (360 - sumangle);}
```

However, to get the angle the only way is to calculate the dot product of 2 vectors created by 3 points in a triangle. Then use acos to get the angle for the corresponding vertex. The algorithm can be shown below:

```
Vertex* nowpoint = this_poly->vlist[i];
std::vector <Vertex*> otherpoints;
if (k != 0) otherpoints.push_back(this_poly->vlist[i]->tris[j]->verts[0]);
if (k != 1) otherpoints.push_back(this_poly->vlist[i]->tris[j]->verts[1]);
if (k != 2) otherpoints.push_back(this_poly->vlist[i]->tris[j]->verts[2]);
vec3 vector1 = vec3(otherpoints[0]->x - nowpoint->x, otherpoints[0]->y - nowpoint->y,
otherpoints[0]->z - nowpoint->z);
vec3 vector2 = vec3(otherpoints[1]->x - nowpoint->x, otherpoints[1]->y - nowpoint->y,
otherpoints[1]->z - nowpoint->z);
angle = 180.0*glm::acos(glm::dot(vector1, vector2)/(glm::length(vector1)*
glm::length(vector2)))/3.1415926;
        sumangle += angle;
```

The final result has been shown in table 4:

Model	Total Angles Valence Deficit
bunny	719.973≈720
dodecahedron	/
dragon	-0.063≈0
feline	-720.032≈-720
happy	-3600.060≈-3600
hexahedron	/
icosahedron	720.000≈720
octahedron	720.000≈720
sphere	719.976≈720
tatrahedron	/
torus	-0.019≈0

Table 4. Total angle valence deficit for all models

The relationship I have found from table 4 is that: the total angles valence deficit of each model relates to how many **tunnels** it has, which means it can follow the equation shows below:

Total Angle Valence deficit (M) = 720° - 720°*(Number of tunnels (M)) = 360°*(V(M)-E(M)+F(M)) .

Taking happy model as an example, according to the equation, the total angles valence deficit of this model should be 360°*(-10) = -3600°. Then if we check the table, the result is exactly what our formula tells us.

e. You now have two measures of flatness: a vertex with a valence of six (zero valence deficit), and a vertex with a total of interior angle of 2π (zero angle deficit). Where in the mesh do you tend to see irregular vertices, and where do you see vertices with a total interior angle of much more or less than 2π? How do these two types of vertices position with respect to each other? Can you make some hypotheses and validate them? Use your program to facilitate the investigation.

The irregular vetices and irregular total interior angle (more or less than 2π) always shows when the surfaces has the changes of the tangent of a point (here we can think it as the mesh's vertex) of surface.

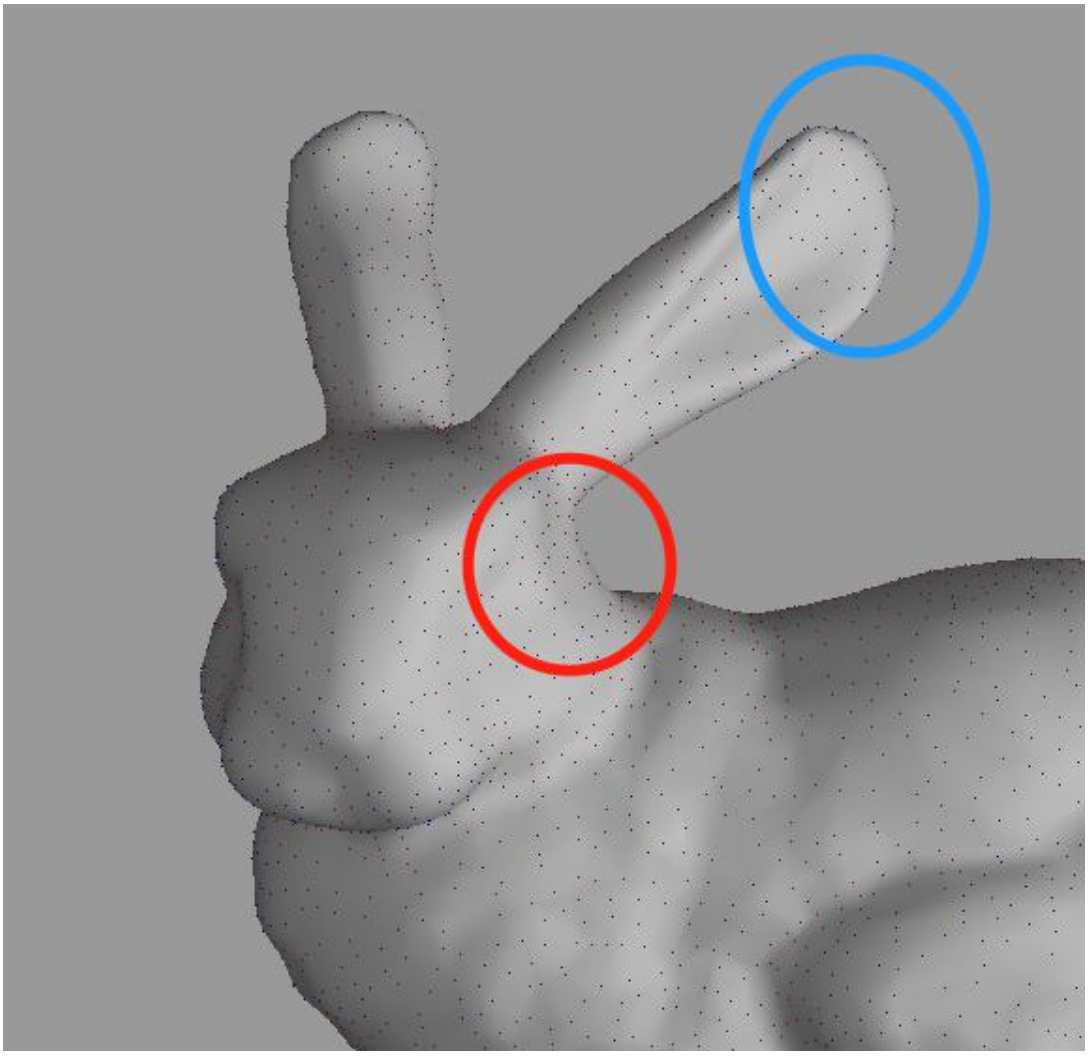


Figure 5. Positive (Blut) and Negative (Red) angle valence deficit

The picture shown above tells us that, when the surface normal has the reverse tangent increase direction of the surface, the angle valence deficit should be positive as the blue circle region shows. However, when the surface normal has the same tangent increase direction of the surface , the angle valence deficit should be negative as the red circle region shows.

3. (3D Checkerboard texture) Given a surface S and any point p=(x, y, z), the following functions will determine the color for p: R = f([x / L]), G = f([y / L]), B = f([z / L]),

where L is a positive real number and $f(n)=\begin{cases} 0, & \text{n is odd} \\ 1, & \text{n is even} \end{cases}$ **.**

a. Draw on paper the result of applying this solid texture to the unit disk for L=1. Since a disk is 2D, you can draw the Blue channel and use only Red and Green channels. You can assume that B is always 1.

To solve the problem I used the MATLAB for showing the math calculating. Setting that right now my paper, which is parallel to xy plan, has 3*3 area and Z-intercept with 1. The result has been shown below, and the square with purple outline is the unit disk:

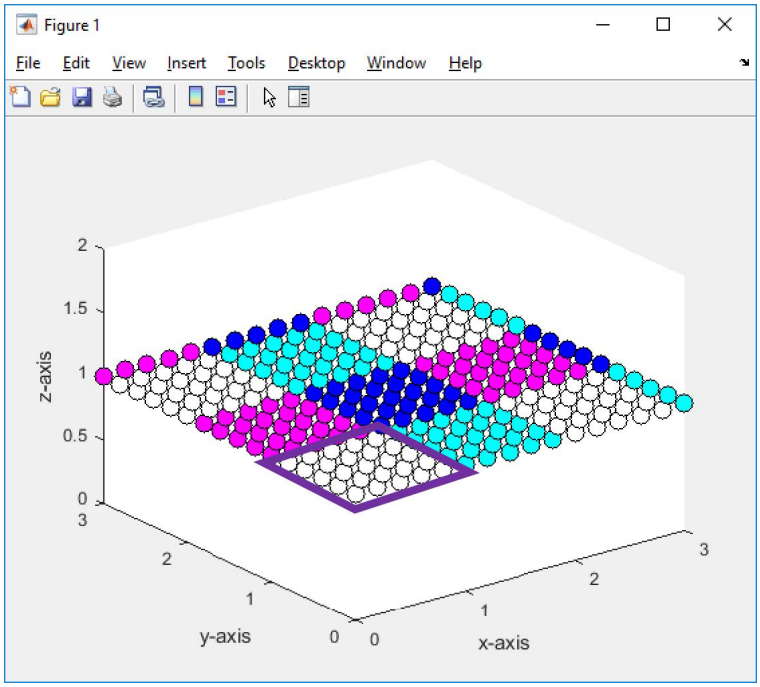


Figure 4. 2D checkerboard texture with L = 1

b. Now draw the results when you change L to 0.5 and to 2.0.

Change L to 0.5, the 2D paper will be drawn as below, the square with purple outline is the unit disk:

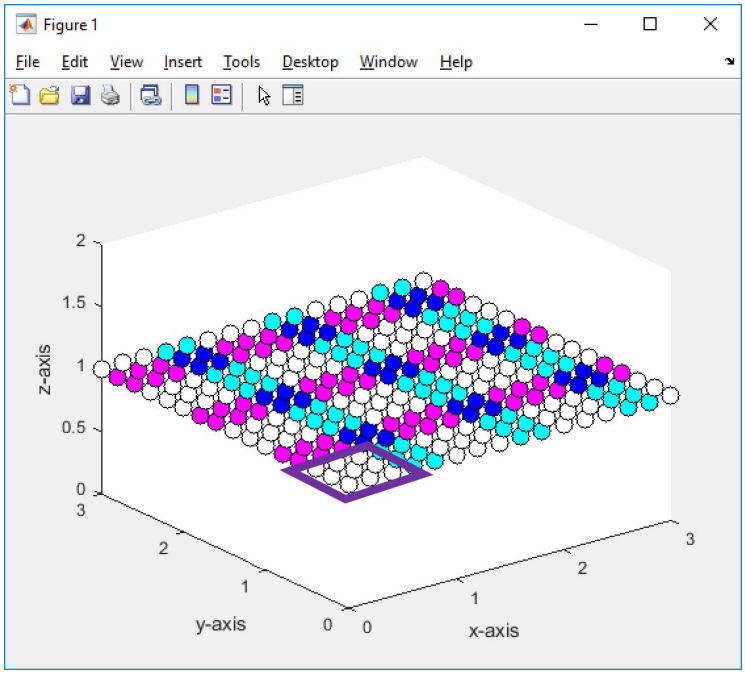


Figure 6. 2D checkerboard texture with L = 0.5

Change L to 2, the 2D paper will be drawn as below, the square with purple outline is the unit disk:

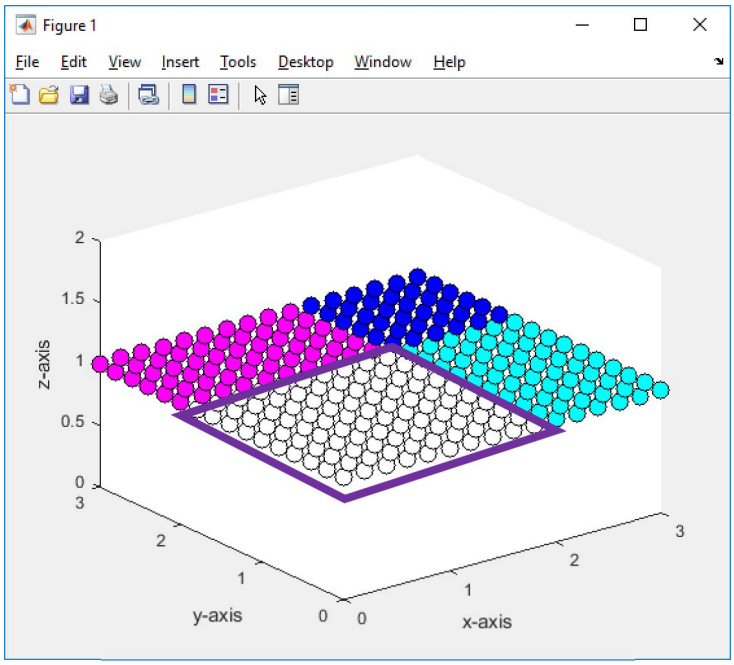
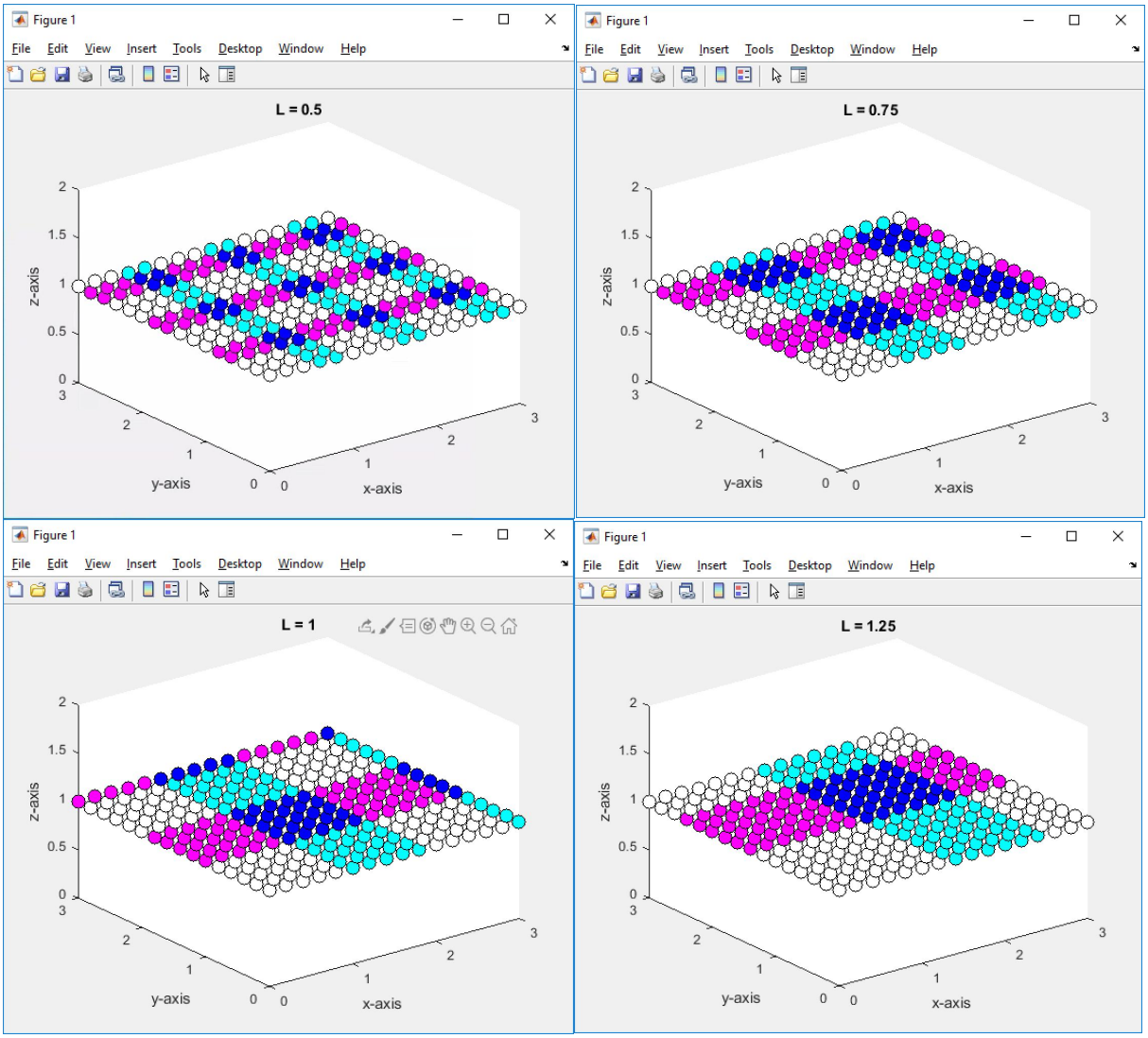


Figure 7. 2D checkerboard texture with L = 2

c. Include the three drawings in the write-up and explain what happens when L decreases and when L increases.

According to the results of L = 0.5, 1, 2 (Figure3,4, and 5), we can find a rule that L will change the area size of checkerboard unit. The smaller the value of L, the more frequently the colors appear in a staggered and regular arrangement. We can see this rule more clear if we add more experiments shown below.



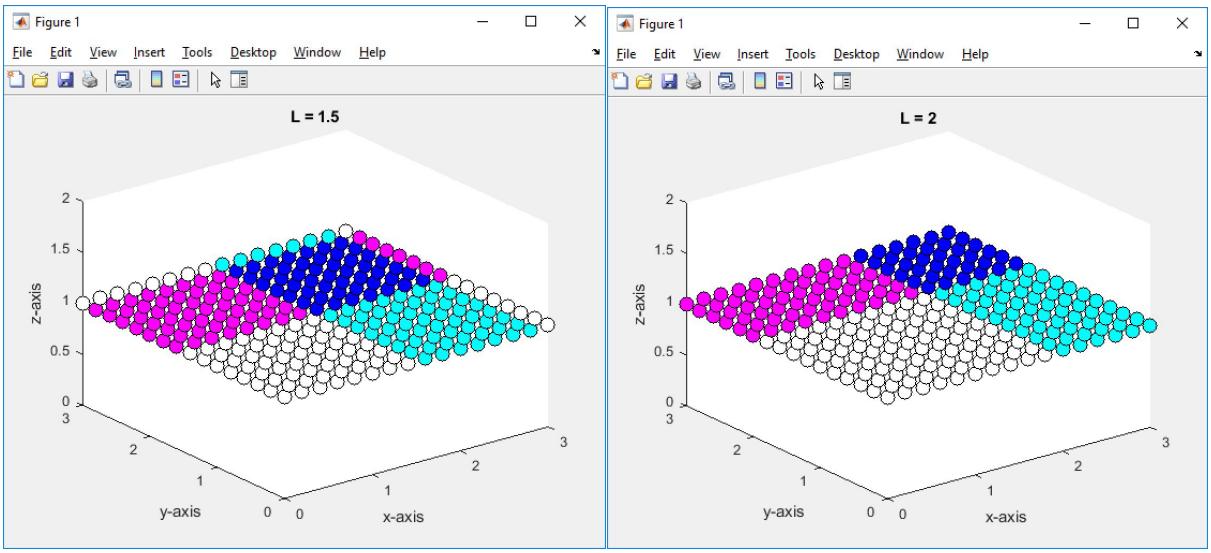


Figure 8. 2D checkerboard texture with increasing value of L

d. Now you will program the above solid textures in Learnply as follows: assign a color to each vertex based on its 3D coordinates as follows:

$R = f([V_x / L]), G = f([V_y / L]), B = f([V_z / L]),$

Taking the **Bunny** model as an example, setting L is 0.5 and removing the light, the textured result can be shown as below:

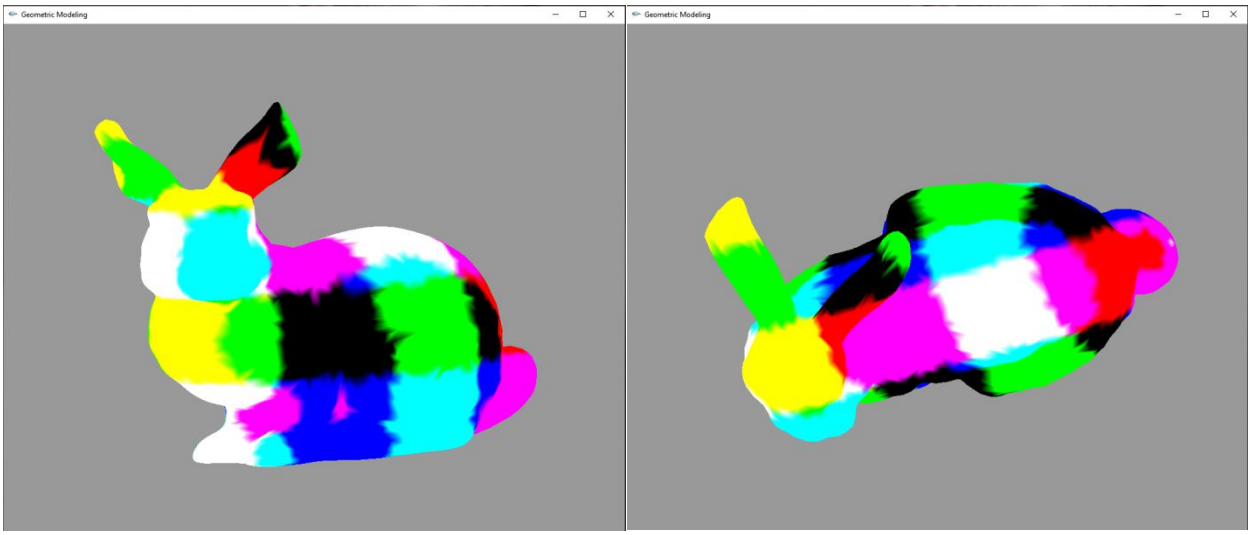


Figure 9. 3D checkerboard texture on Bunny with L = 0.5

e. Is the result perfect given an L?

Actually, the result is not perfect, because when you see the edges of the checkerboard unit, they are really coarse even we use “GL_SMOOTH” to blur the edges.

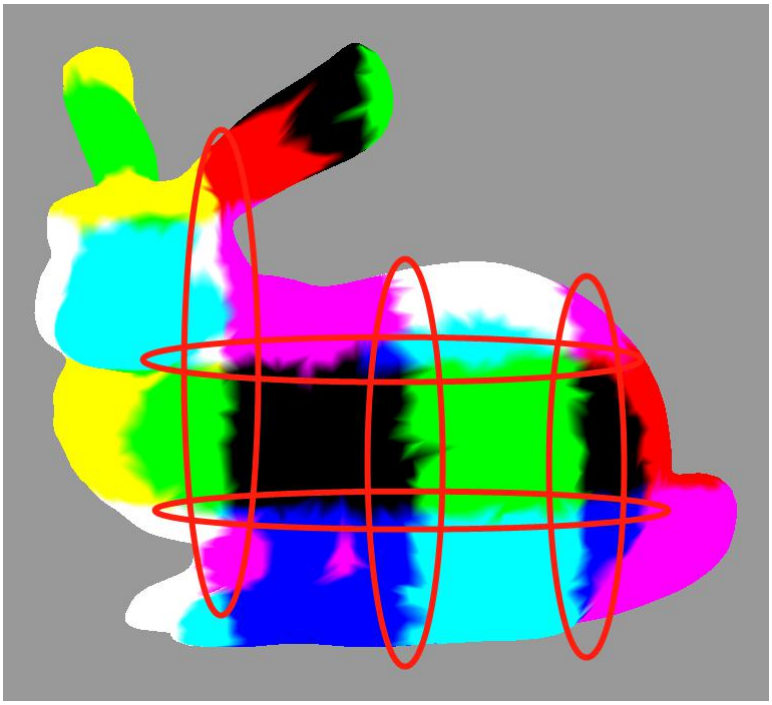


Figure 10. 3D checkerboard texture edges

f. When you increase L, does the artifact increase or decrease?

Visually, the artifact increase, but actually there are no changes of the degree of texture edges' roughness.

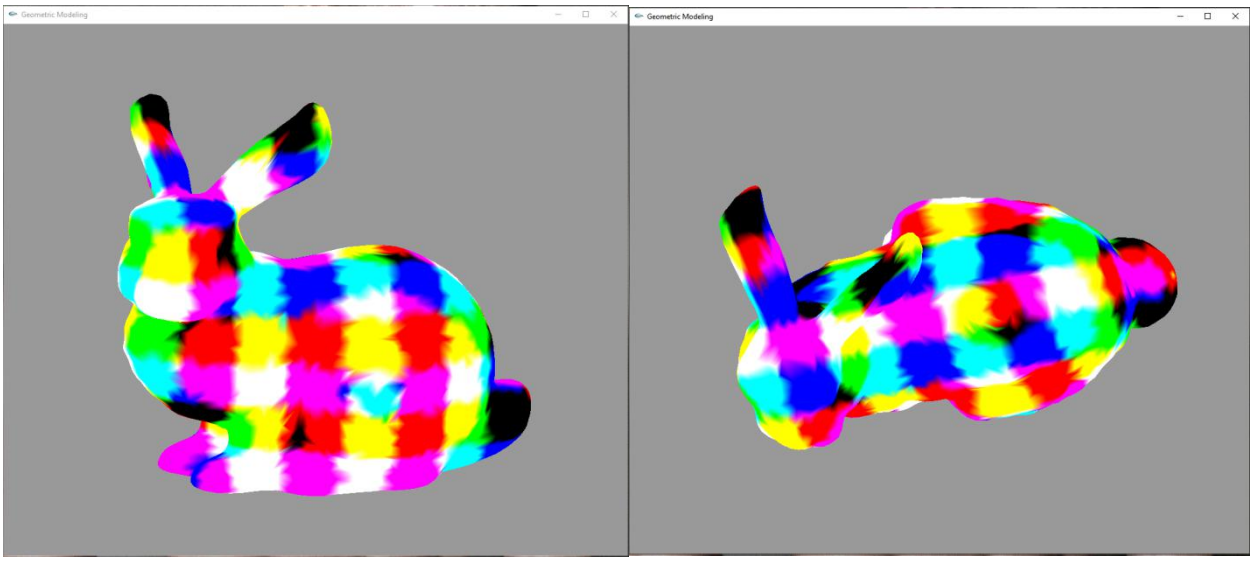


Figure 11. 3D checkerboard texture on Bunny with L = 0.25

g. When you decrease L, does the artifact increase or decrease?

Visually, the artifact decrease, but actually there are no changes of the degree of texture edges' roughness.

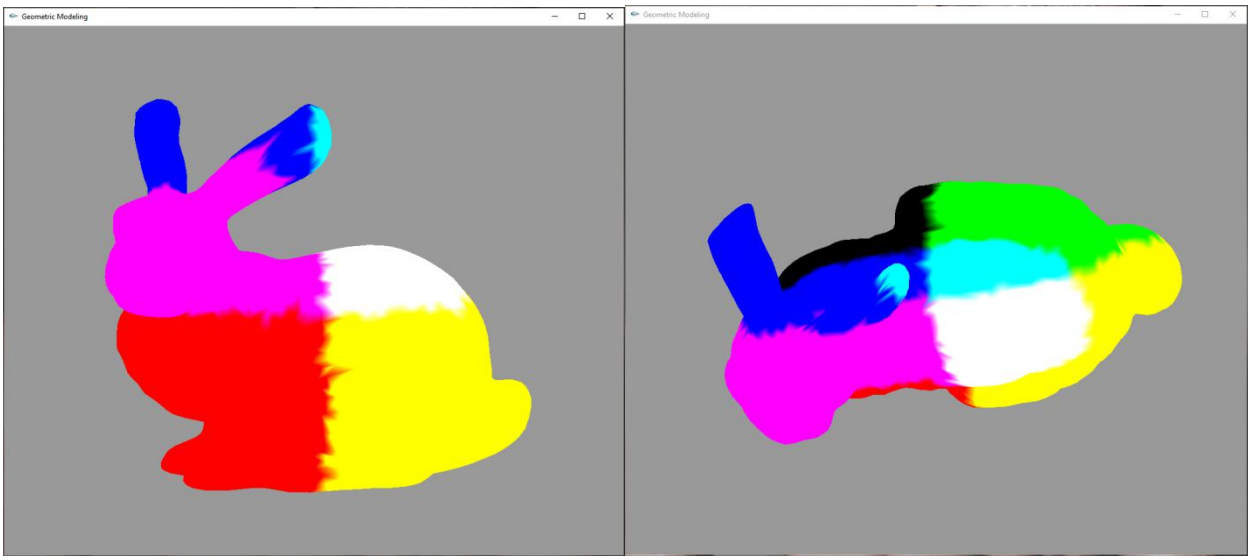


Figure 12. 3D checkerboard texture on Bunny with L = 1

h. Explain why such artifacts exist.

In my opinion, the checkerboard texture it self is perfect, however, the triangle meshes of the models mainly affects the final results. Because of the 3D mesh with some irregular 3D locations, it is hard for the texture to map on the coarse mesh surface which is supposed to have good smoothness. Thus, for the 3D model with low resolution of triangle meshes, even for surfaces that have good derivative continuity on the geometric level (such as torus model, which still exists jagged edges) it is still hard to have perfect checkerboard texture mapping.

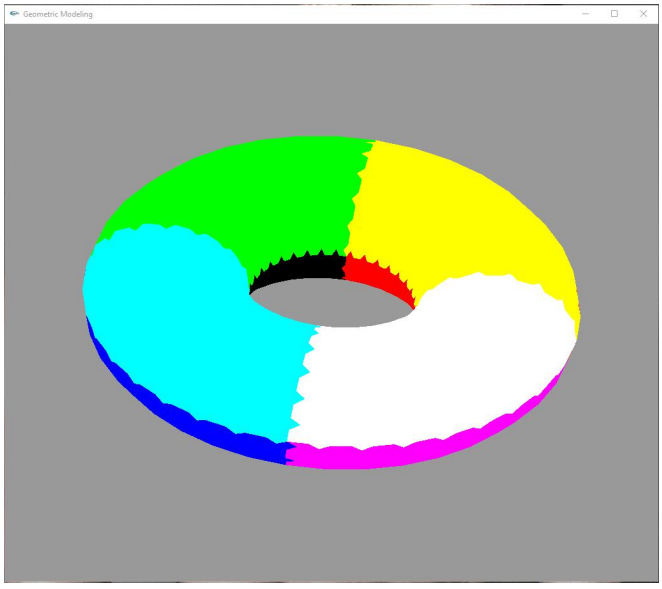


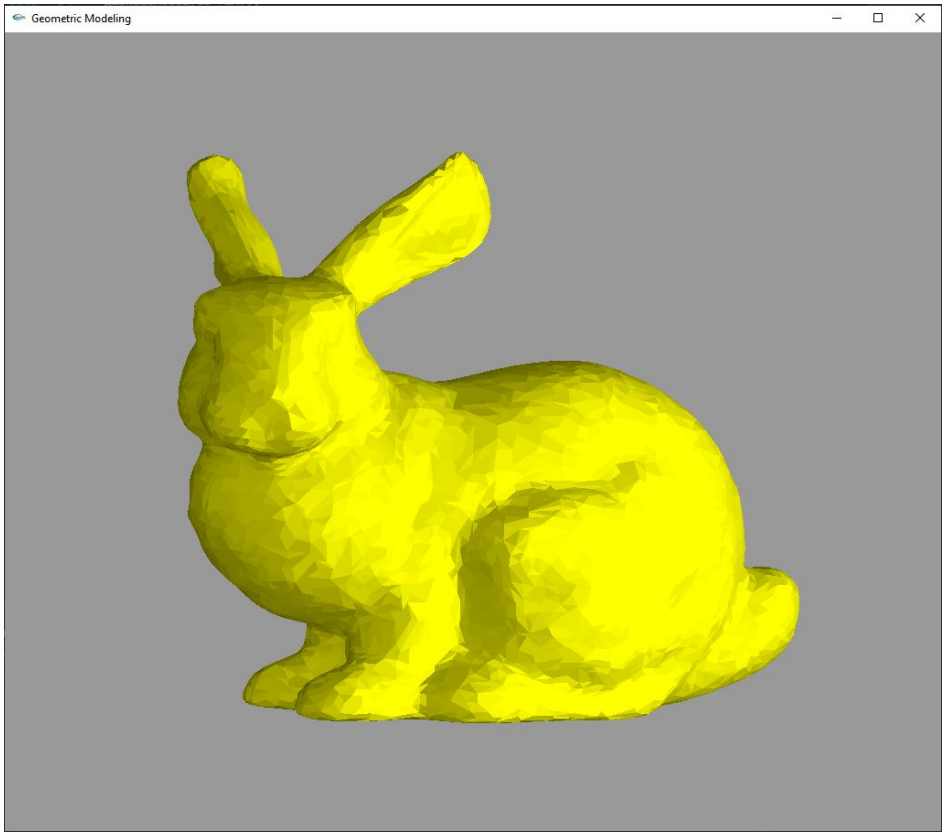
Figure 13. 3D checkerboard texture on torus with L = 1

4. (Regular subdivision) Implement the Loop subdivision scheme for a triangular mesh. Prove that the Euler characteristic of the surface is not changed by regular subdivision. Finally, iteratively subdivide your meshes until the 3D checkerboard texture looks acceptable.

Basically, the thought of subdivision is try to create new vertices on each edge, and update the value of vertices of original mesh. Then rebuild the topology property of new sub-triangles. The correct edges connection shows the topological sequence of new mesh.

Take the **bunny** model as an example, let's see the result with different iterate numbers of subdivision:

Let's firstly use **GL_FLAT** so that we can see more clear about the changes of mesh. In Figure 13, 14, 15, left side pictures shows mesh's outside looking, and the right side pictures shows the geometry data outputs.



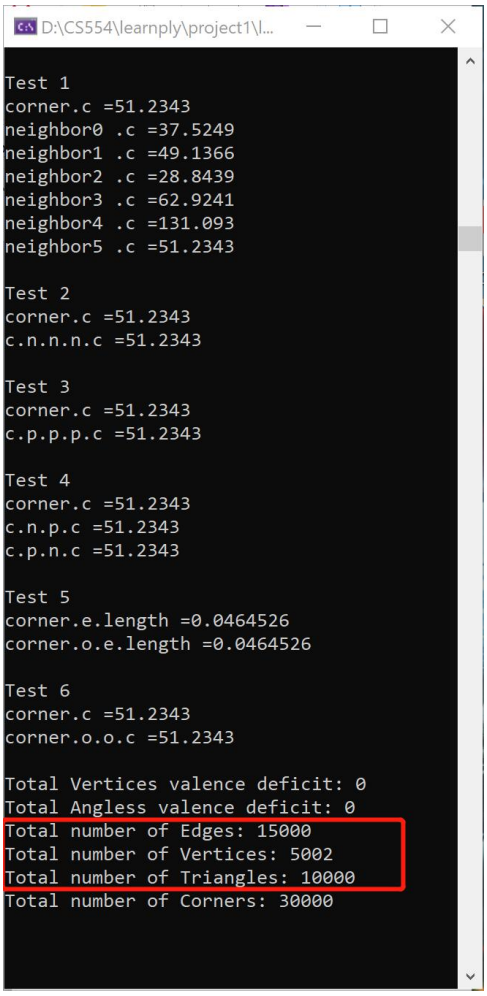
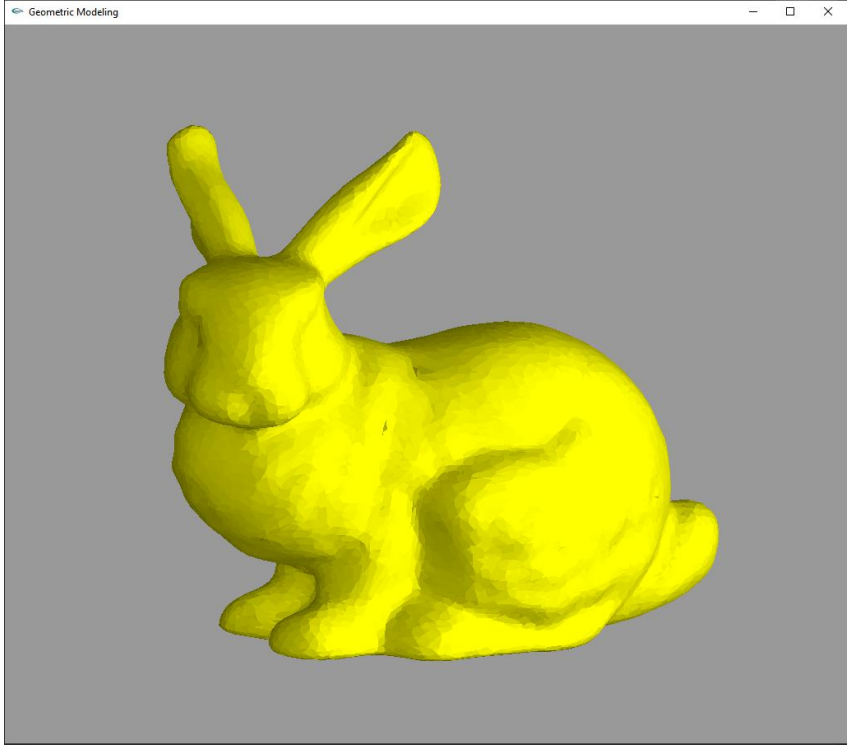


Figure 14. Bunny mesh with no Regular subdivision



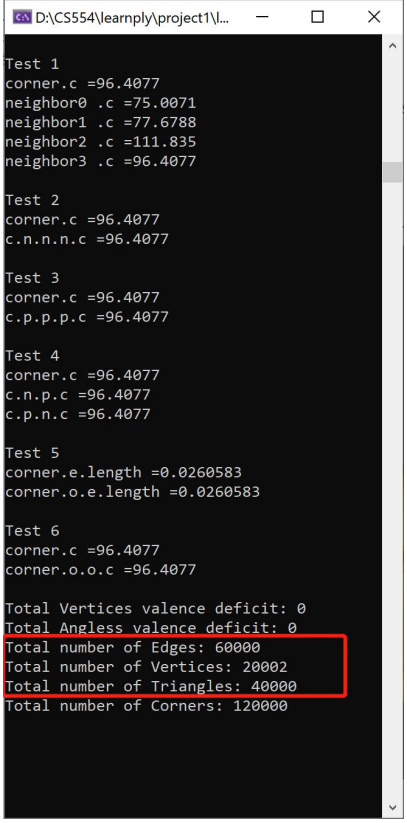
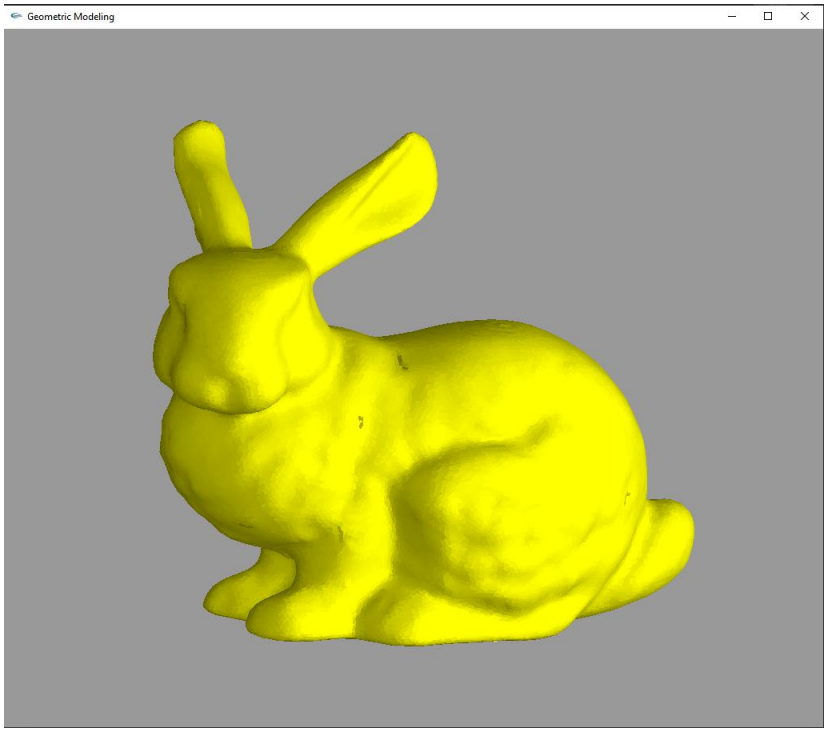


Figure 15. Bunny mesh with 1 time Regular subdivision



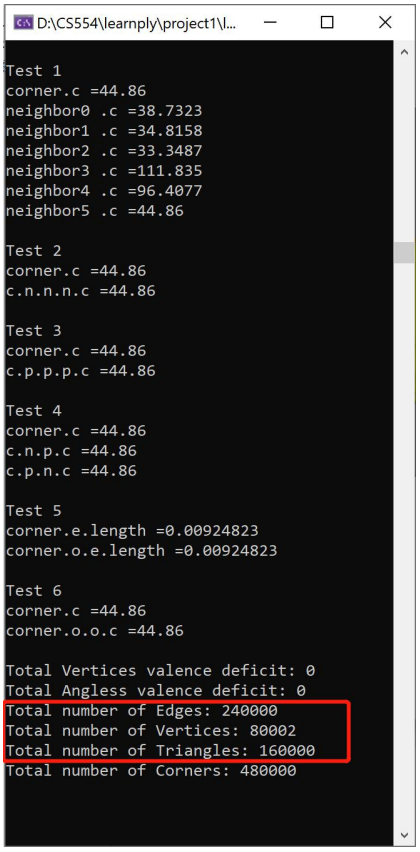


Figure 16. Bunny mesh with 2 times Regular subdivision

From the graph shows above, we can calculate the Euler characteristic of the surface is not changed by regular subdivision. If we focus on the red frames on the right side graphs, for the original mesh (Figure 13): V-E+F = 5002-15000+10000 = 2; for the mesh with one time iteration of subdivision (Figure 14): V-E+F = 20002-60000+40000 = 2; for the mesh with two times iteration of subdivision (Figure 15): V-E+F = 80002-240000+160000 = 2. The calculation tells us that the regular subdivision will not change the Euler characteristic of the surface.

Then, let's see meshes' 3D checkerboard texture with different time iteration with subdivision. Here we change **glShadeModel** back into **GL_SMOOTH** and choose the L to be 0.25. The result can be seen in Figure 17, 18 and 19:

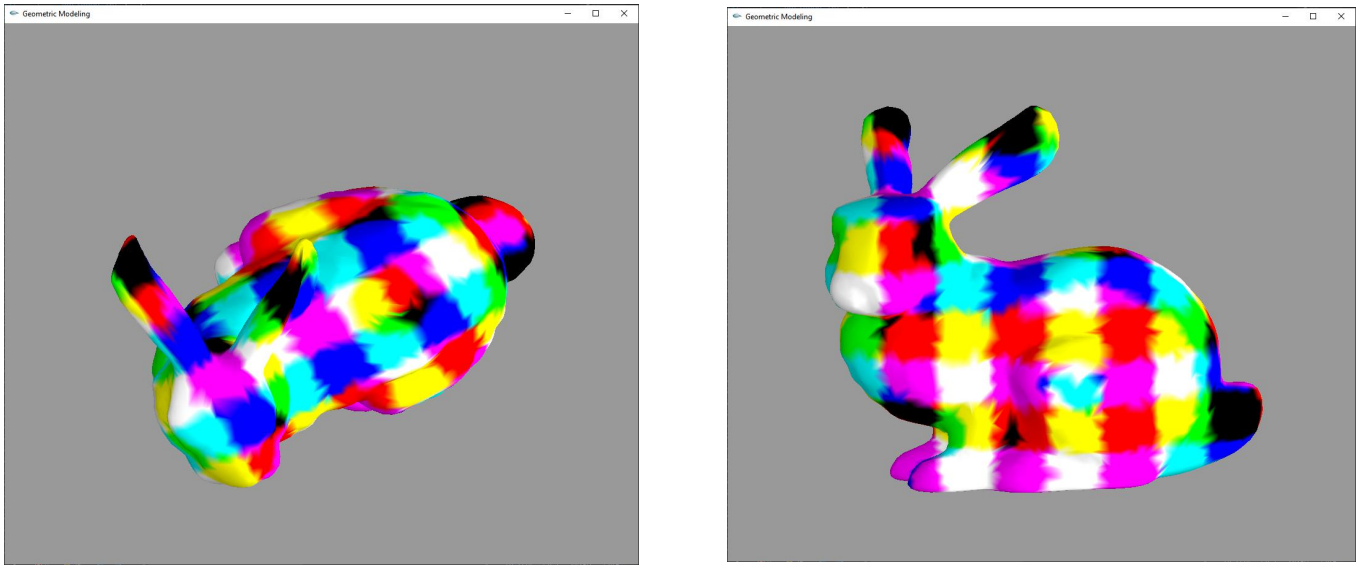


Figure 17. 3D checkerboard on Bunny with no subdivision

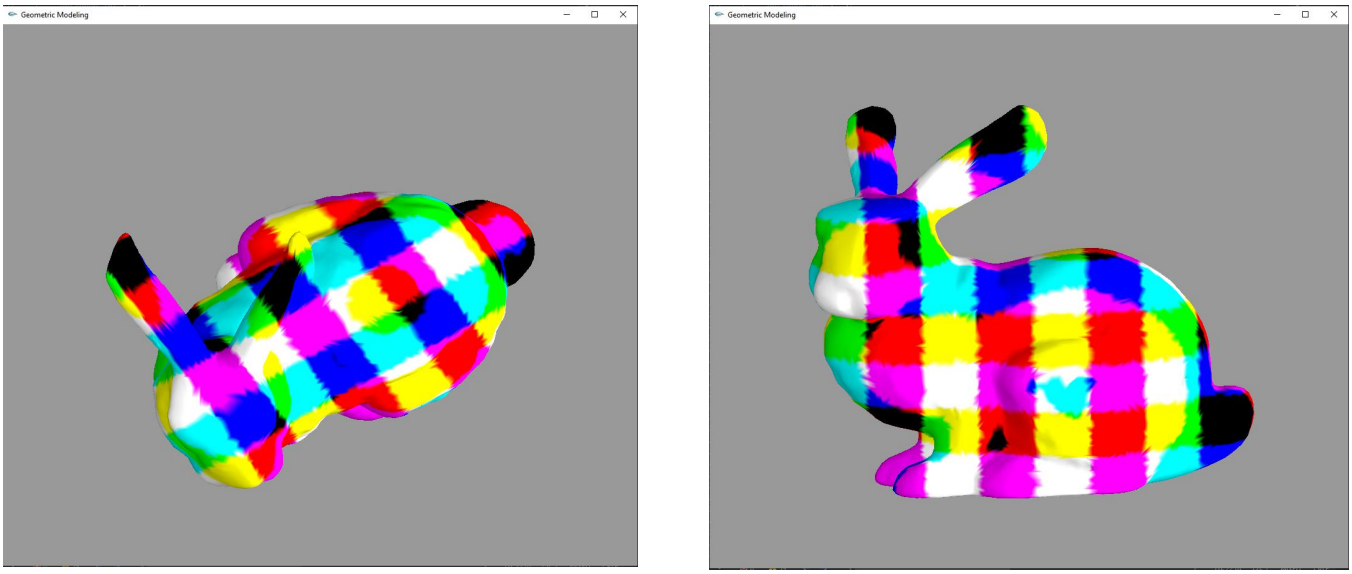


Figure 18. 3D checkerboard on Bunny with 1 time regular subdivision

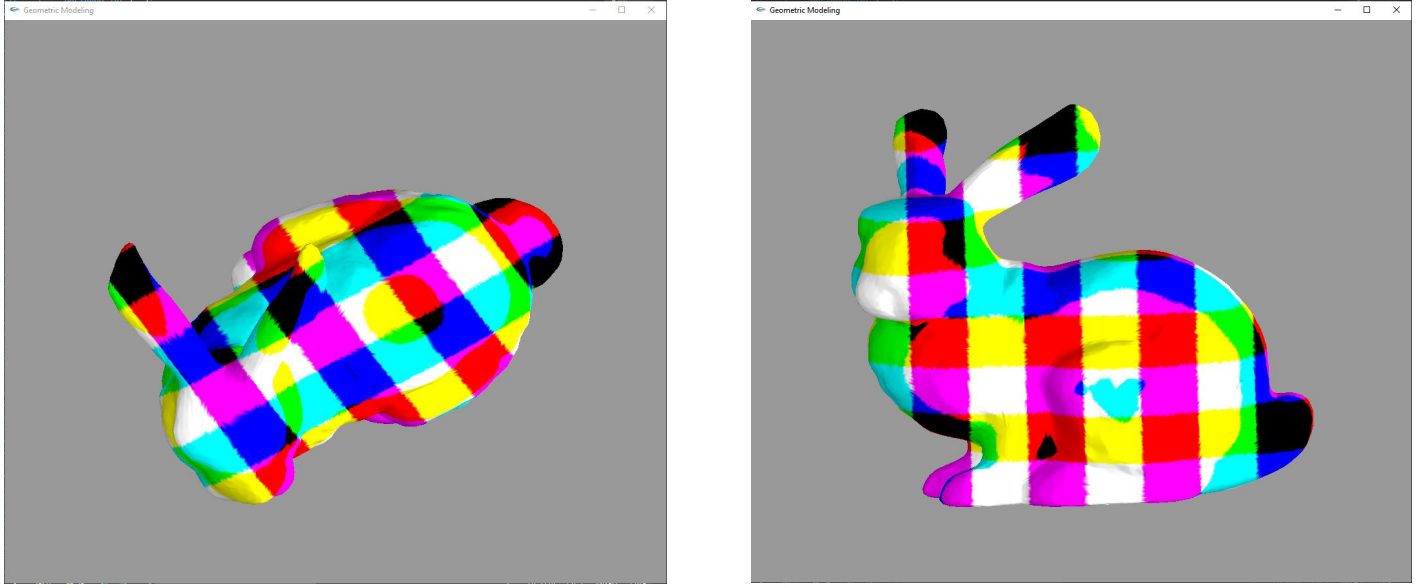
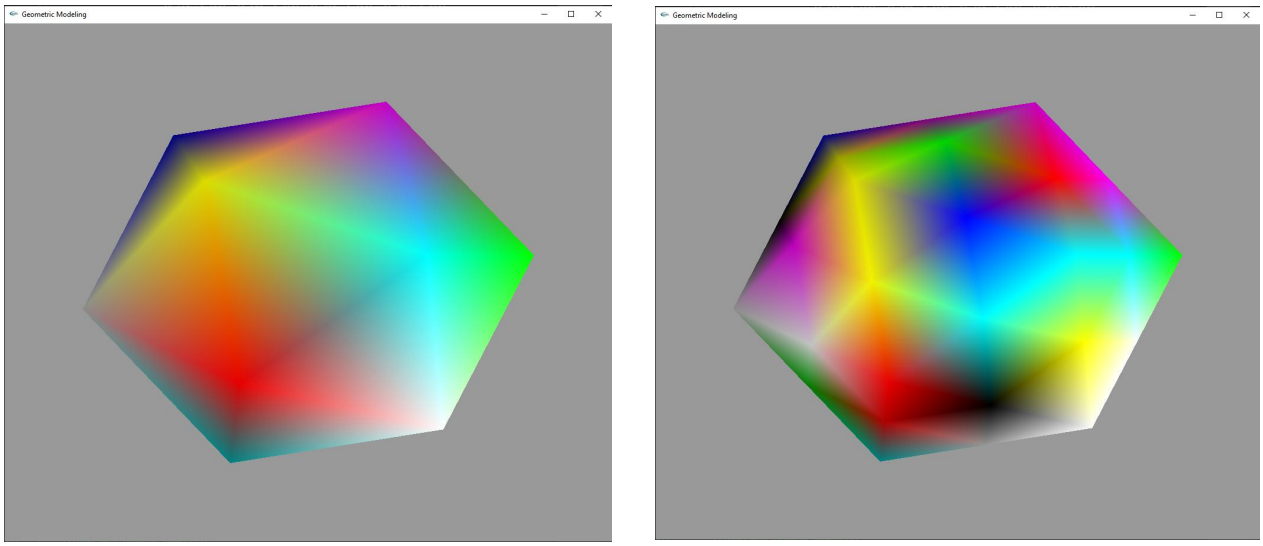


Figure 19. 3D checkerboard on Bunny with 2 times regular subdivision

For the regular subdivision, we can get a reasonable 3D checkerboard texture after we making the 2^{th} iteration of the subdivision. But it is not valuable for doing more iterations calculation under the algorithm of regular subdivision. Actually, for the complex model such as **bunny** and **happy**, the second iteration costs almost 4~5 minutes for calculation, so the time increment almost increase in exponential order.

But for the simple model such as **octahedron**, to fine the 3D checkerboard texture, we can do the iteration at least 4 times. The picture has been shown below:



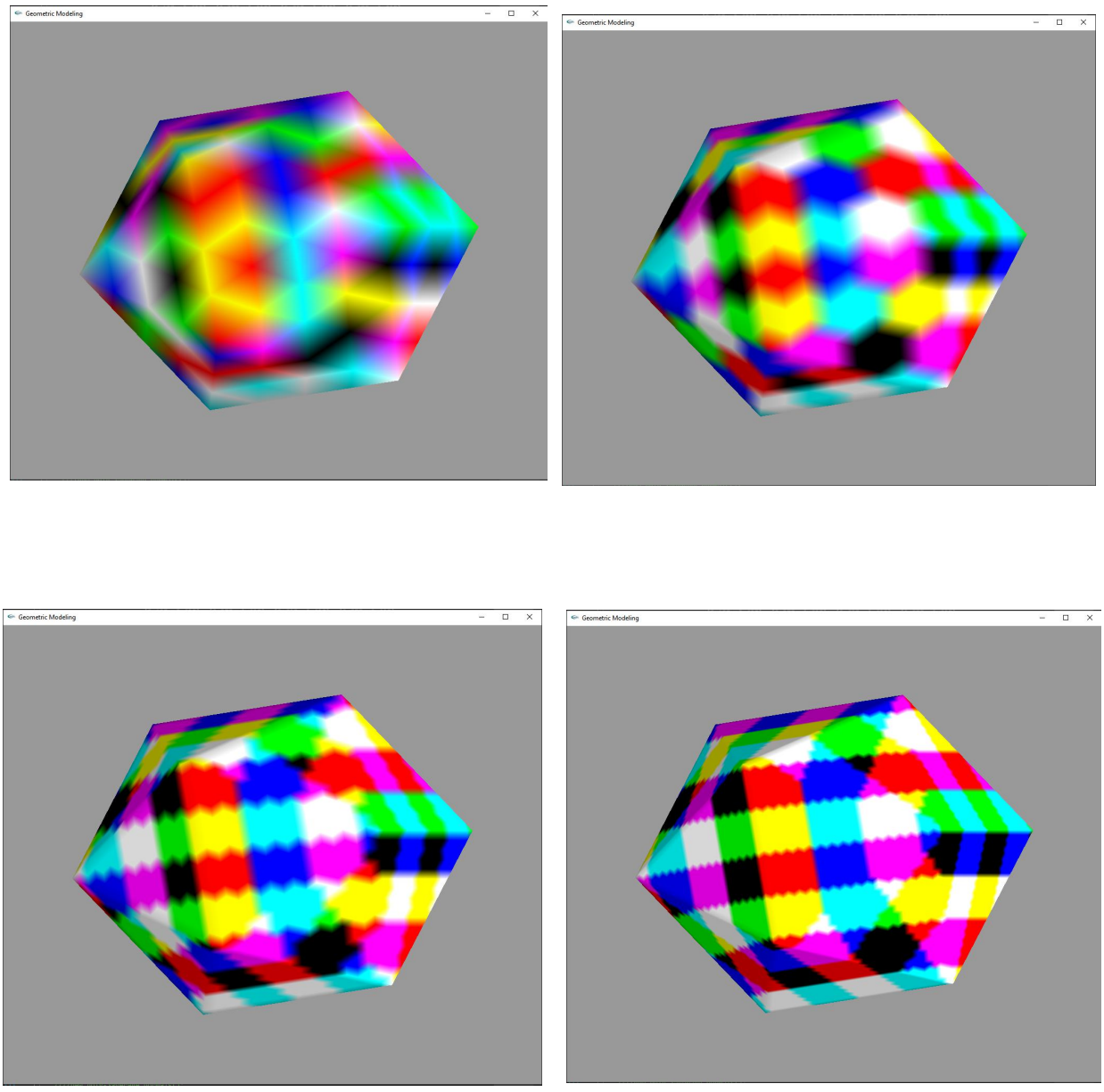
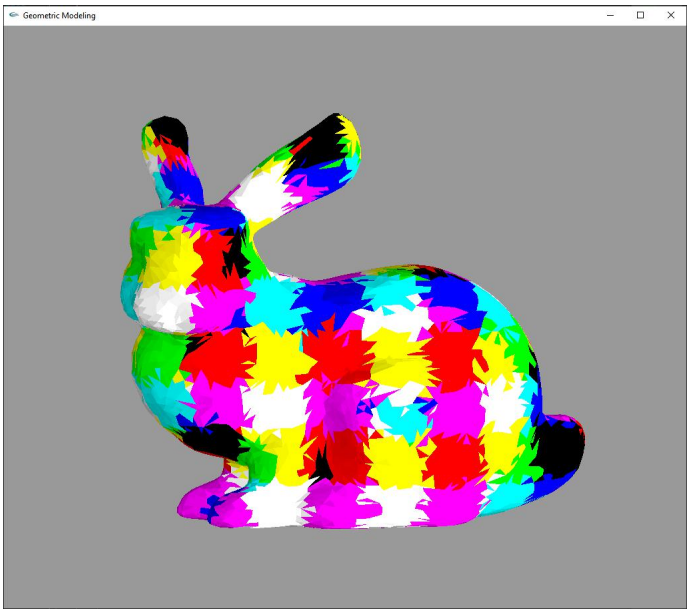


Figure 20. Octahedron 3D checkerboard texture with 4 times iteration

5. (Irregular subdivision: graduate students only) Implement the irregular subdivision scheme for a triangular mesh. Prove that the Euler characteristic of the surface is not changed by this subdivision scheme. Finally, iteratively subdivide your meshes until the 3D checkerboard texture looks acceptable. Compare the efficiency of the irregular subdivision to that of the Loop subdivision. You can introduce your own definition of the efficiency but need to justify it.

Take **bunny** model as the example. In question 4 we get a conclusion that, the most efficient time of iteration for the complex mesh model is 2. During the operation, last iteration cost 4~5 minutes. To test the efficiency of the irregular subdivision, we use the GL_FLAT shading mode, L=0.25, and stop the iteration when the last subdivision cost the same 4~5 minutes. The testing experiment process has been shown below:



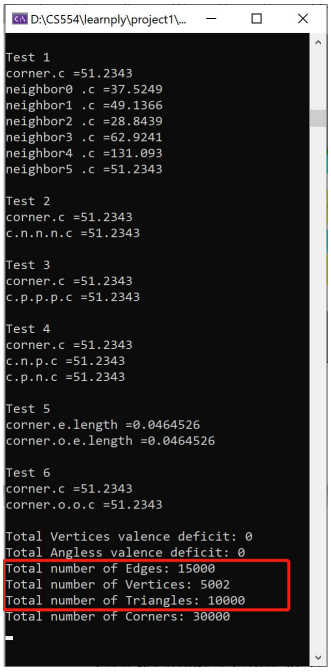
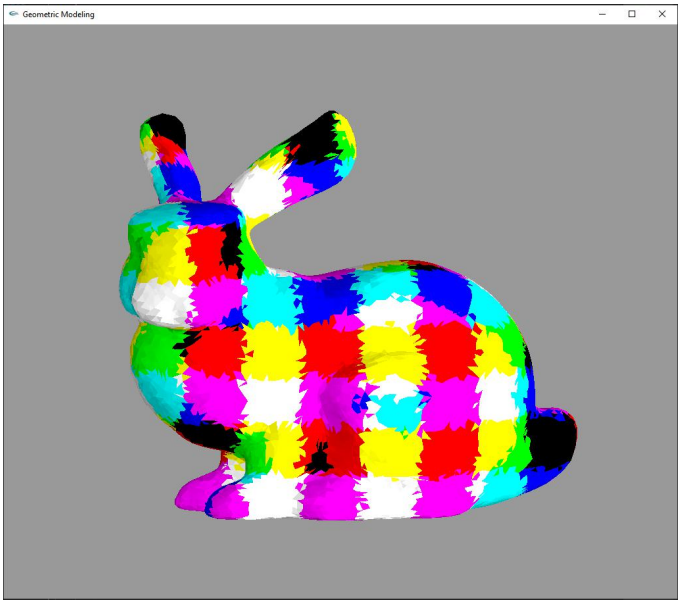


Figure 21. 3D checkerboard on Bunny with no subdivision



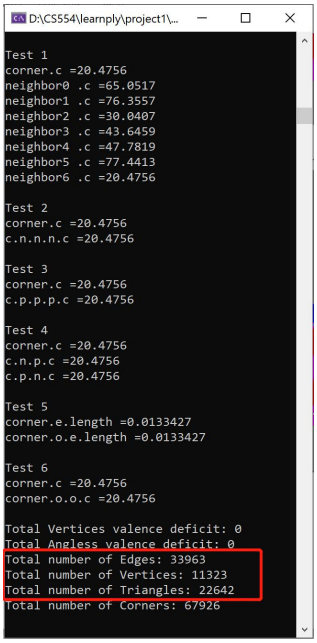
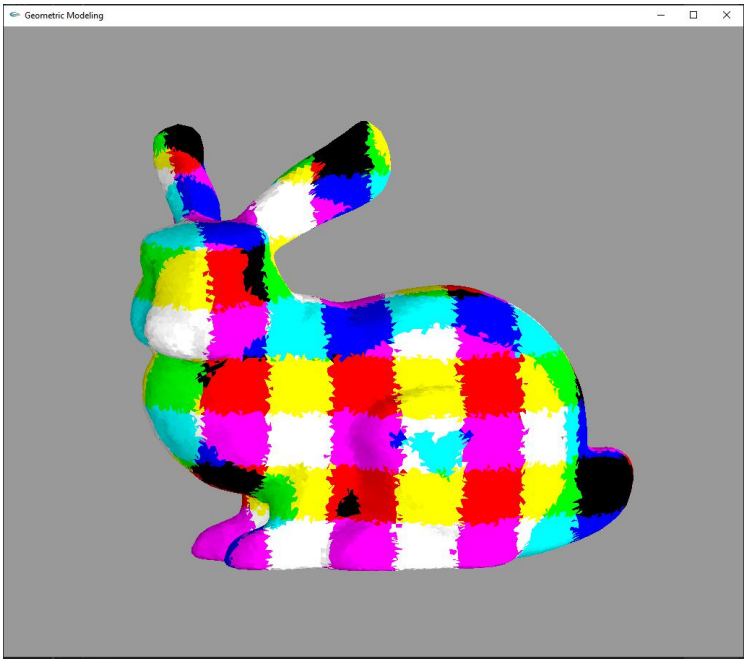


Figure 22. 3D checkerboard on Bunny with 1 time subdivision



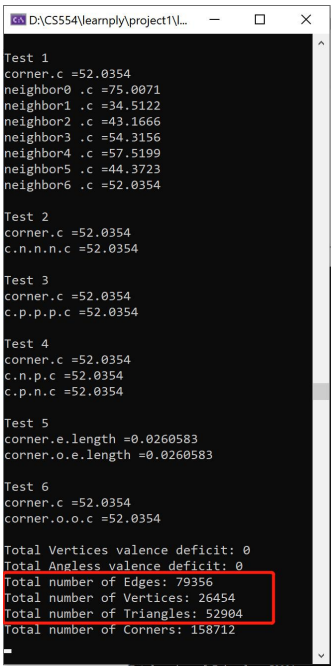
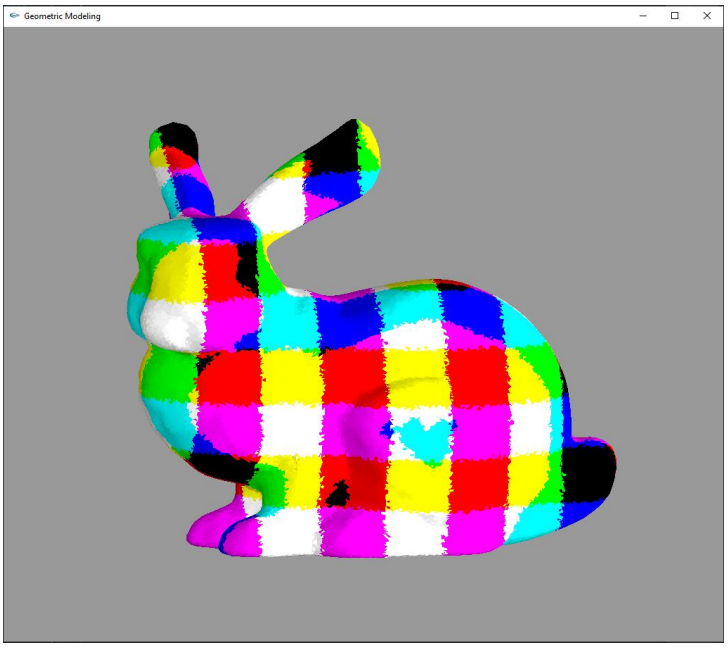
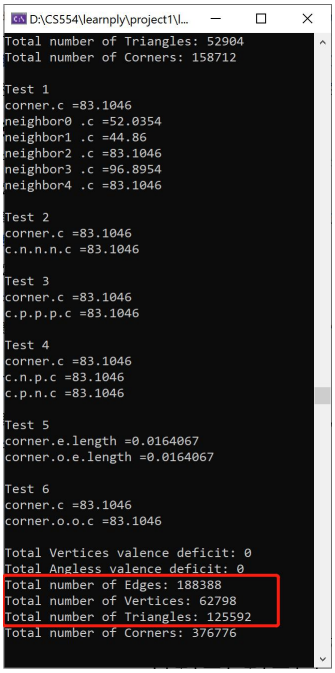


Figure 23. 3D checkerboard on Bunny with 2 times subdivision

Figure 24. 3D checkerboard on Bunny with 3 times subdivision





From the results in Figure 20-23, and comparing Figure 22 and Figure 19, we can see that even we use the **GL_FLAT**, in almost same time, irregular subdivision can get the same good result faster and do one more time iteration of subdivision. All the results have proved that the irregular subdivision is more efficient than the Loop subdivision.

Finally, let's calculate the Euler characteristic: for the mesh with no iteration of subdivision (Figure 20): V-E+F = 5002-15000+10000 = 2. for the mesh with the first iteration of subdivision (Figure 21): V-E+F = 11323-33963+22642 = 2. for the mesh with the second iteration of subdivision (Figure 22): V-E+F = 26454-79356+52904= 2. for the mesh with the third iteration of subdivision (Figure 22): V-E+F = 62798-188388+125592= 2.

The calculation tells us that the irregular subdivision will also not change the Euler characteristic of the surface.