

浙江大学

本科实验报告

课程名称：基于 GPU 的绘制

实验名称：Outlines

姓 名：*****

学 院：计算机学院

系：数字媒体与网络技术

专 业：数字媒体技术

学 号：315010****

指导教师：唐敏

2018 年 1 月 8 日

浙江大学实验报告

实验名称: Outlines 实验类型: 操作实验

同组学生: 无 实验地点: 外经贸楼

一、实验目的与内容

a) 实验目的

在实验过程中熟练顶点着色器的使用。

b) 实验内容

使用 GLSL 语言编写顶点着色器，为模型添加轮廓线和边缘线。

二、实验环境与配置

a) 实验环境

i. 开发环境

Windows 7 SP1 64-bit + CLion 2017.3 + MinGW-W64 5.0 + CMake 3.9.1

ii. 硬件

NVIDIA Quadro K620 (OpenGL 4.5) & NVIDIA GeForce GTX 960M (OpenGL 4.5)

iii. 第三方库

GLEW 7.0 64-bit + FreeGLUT 3.0.0 64-bit + GLM 0.9.8.5

b) 实验配置

本实验项目使用 C++ 编写，在项目中依次包含 GLEW, FreeGLUT 和 GLM 的头文件以及其他必要的系统头文件，链接 64 位版的 GLEW、FreeGLUT 库文件，使用 CMake 生成可执行文件。

三、实验方法与步骤

a) 编写窗口、事件、键鼠控制等基础模块

复用曾经编写的工程模块。

b) 编写 Shader 类封装着色器相关操作

复用曾经编写的工程模块。

c) 编写着色器以实现功能

细节将在下一节中介绍。

四、实验结果与分析

a) Overall: 实验效果

本次实验的最终效果如 Figure 1 所示。

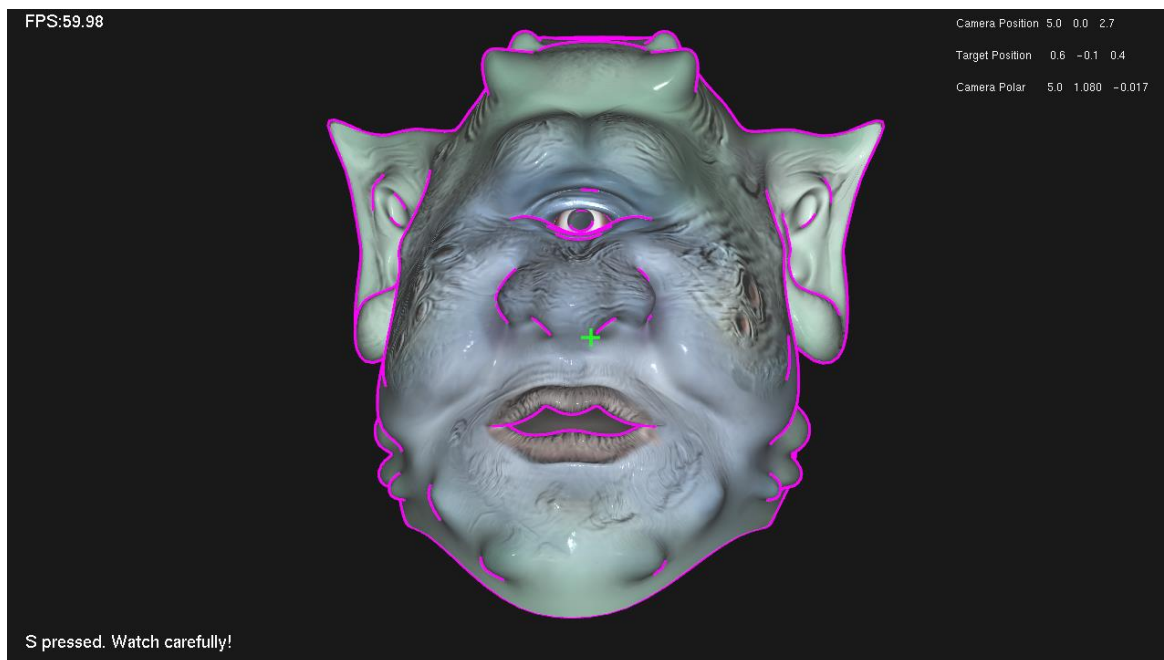


Figure 1 实验效果

b) CPU: 输入 OBJ 模型并进行预处理

从 OBJ 文件中，我们可以读取到模型的顶点位置、顶点法线、纹理坐标和面上的顶点索引信息。在这个过程中，非三角面的面将被分割为三角面（Figure 2）。

```
// If number of edges in face is greater than 3,
// decompose into triangles as a triangle fan.
if (face.size() > 3) {
    int v0 = face[0];
    int v1 = face[1];
    int v2 = face[2];
    // First face
    faces.push_back(static_cast<GLuint>(v0));
    faces.push_back(static_cast<GLuint>(v1));
    faces.push_back(static_cast<GLuint>(v2));
    for (GLuint i = 3; i < face.size(); i++) {
        v1 = v2;
        v2 = face[i];
        faces.push_back(static_cast<GLuint>(v0));
        faces.push_back(static_cast<GLuint>(v1));
        faces.push_back(static_cast<GLuint>(v2));
    }
} else {
    faces.push_back(static_cast<GLuint>(face[0]));
    faces.push_back(static_cast<GLuint>(face[1]));
    faces.push_back(static_cast<GLuint>(face[2]));
}
```

Figure 2 105:17@vbmesh.cpp

利用这些数据，我们首先计算面法线：按每个面的顶点索引，找到这个每个顶点的位置，计算出两条边（向量形式），并做叉乘，就可以得到这个面的面法线（Figure 3）。

```

void VBOMesh::generateNormals(
    const vector<vec3> &points,
    const vector<GLuint> &faces,
    vector<vec3> &faceNormals) {

    for (GLuint i = 0; i < faces.size(); i += 3) {
        const vec3 &p1 = points[faces[i]];
        const vec3 &p2 = points[faces[i + 1]];
        const vec3 &p3 = points[faces[i + 2]];

        vec3 a = p2 - p1;
        vec3 b = p3 - p1;
        vec3 n = glm::normalize(glm::cross(a, b));

        faceNormals.push_back(n);
    }
}

```

Figure 3 229:1@vbomesh.cpp

然后我们需要对模型的每一条边创建一个高度为 0 的矩形（这个矩形由两个三角面拼合而成，即增加 6 个顶点）。我们称这些矩形为“候选轮廓线”。它被传入顶点着色器之后，“候选轮廓线”的一条长边上的 2 个顶点（对应到 2 个三角面上的 3 个顶点）会被设置一定的位置偏移。我们可以这样操作是基于这样的事实：高度为 0 的“候选轮廓线”不会对显示效果有任何影响。当这些面在一定的条件下，高度大于 0，则会产生显示效果，由此，我们已经论证了显示轮廓线的可行性。那么，只要再定义在何种条件下“候选轮廓线”会被显示即可。我们在“候选轮廓线”可能被偏移的顶点上，附加上一组额外的属性：这个顶点所在的边所在的一组面（一般为两个）的面法线方向。为了得到每一条边的两个邻接面，我们使用复杂度为 $O(n^2)$ 的遍历，将每个面与其他所有面做判断，其中任意两个三角面的六条边相互匹配的情况共 $3 * 3 = 9$ 种。模型的边界面很容易成为轮廓线，所以应该被标记：一个非边界面（我们称处于模型的边界的面为边界面）会与自身以外的 3 个面匹配，匹配面数不满 3 的三角面被认为是一个边界面，而模型边界边就在边界面上。我们为边界边添加一个特殊的相邻面法向量，这个特殊值这将在顶点着色器中被判断（Figure 4）。

```

vector<vec3> newNormalBeside_empty;
vector<vec3> newNormalBeside = {faceNormals[i], faceNormals[j]};
if (i == -1 && j == -1) {
    newNormalBeside = {vec3(2.0), vec3(2.0)};
}
normalsBesideToWrite[slot].push_back(newNormalBeside_empty);
normalsBesideToWrite[slot].push_back(newNormalBeside_empty);
normalsBesideToWrite[slot].push_back(newNormalBeside);
normalsBesideToWrite[slot].push_back(newNormalBeside);
normalsBesideToWrite[slot].push_back(newNormalBeside);
normalsBesideToWrite[slot].push_back(newNormalBeside_empty);

```

Figure 4 618:5@vbomesh.cpp

为了将每条边上的 6 个新增顶点与原来的模型顶点区分，我们再用一个布尔量标记这个顶点是否为新增顶点。除此之外，还有顶点的纹理坐标属性。因此，总计有顶点位置、顶点法线、相邻面法线（2 个）、顶点标记、纹理坐标共 6 条顶点属性被传送至顶点着色器中。由于顶点属性不支持布尔值，我们使用 1.0f 表示顶点标记为真，0 表示顶点标记为假。

c) GPU：偏移顶点并着色

i. 顶点着色器：计算面法线朝向，移动顶点

在顶点着色器中，我们可以得到的是：一个顶点的位置、法线、纹理坐标、新增顶点标记和相邻面的法线方向。非新增顶点和新增顶点中的不需要被偏移的点（“候选轮廓线”的另一条边上的顶点），它们的相邻面法向量被设置为 0。我们令每个相邻面法线方向乘以模视矩阵得到这个相邻面在摄像机坐标系中的法向量 n ——若 n 的 z 为正值，表示这个相邻面是背朝摄像机的；若 n 的 z 为负值，表示这个面是朝向摄像机的——那么如果一个顶点的两个相邻面一个朝向摄像机，一个背朝摄像机，那么这两个面的共享边是一条轮廓线，这个顶点就在轮廓线上，需要被偏移。当然，在这之前，如果一个顶点的相邻面法向量是一个特殊值，那么这个顶点是模型边界边上的顶点，直接标记它为需要偏移的点。对于不需要被偏移的顶点，我们正常输出它的坐标；对于需要被偏移的顶点，我们为它的坐标设置一个其自身法向量乘以轮廓线宽度因子的偏移（Figure 5）。

```
vec3 position;
vec3 FNormal_0 = NormalMatrix * FaceNormal_0;
vec3 FNormal_1 = NormalMatrix * FaceNormal_1;
if (FNormal_0.z * FNormal_1.z < 0 || FaceNormal_0.x == 2.0) {
    position = vec3(ModelViewMatrix * vec4(VertexPosition, 1.0)) + LineWidthFactor * normal;
}
else {
    position = vec3(ModelViewMatrix * vec4(VertexPosition, 1.0))
}
```

Figure 5 37:5@outlines.vert

最后，我们将顶点标记和纹理坐标按原样传送到片段着色器。

ii. 片段着色器：判断片元位置，分别着色

在片段着色器中，我们得到的数据是顶点标记（模型上的片元，标记全为 0，轮廓线上的片元，标记全为 1.0f）、顶点位置和顶点法线等。对于处于模型上的片元，我们可以简单地使用一个 ADS 光照模型为模型添加环境色、漫反射和镜面反射颜色；对于处于轮廓线上的片元，我们使用外部定义的轮廓线颜色来着色（Figure 6）。

```
FragColor = isEdge == 0.0 ? vec4(phongModel(normal.xyz, texColor.rgb), 1.0) : LineColor;
```

Figure 6 46:5@outlines.frag

在所有的候选轮廓线中，只有顶点被设置了位置偏移的面才会被显示。至此，轮廓线的显示流程全部结束。

d) Interaction: 观察与调整

本次实验依旧使用第一人称观察系统（也可以按 C 键切换到摄像机控制），使用 WSAD 和鼠标可以控制观察角度。轮廓线宽度的调整由小键盘“-”/“+”键控制，其效果如 Figure 7 和 Figure 8 所示。按空格键可以暂停/恢复模型旋转。

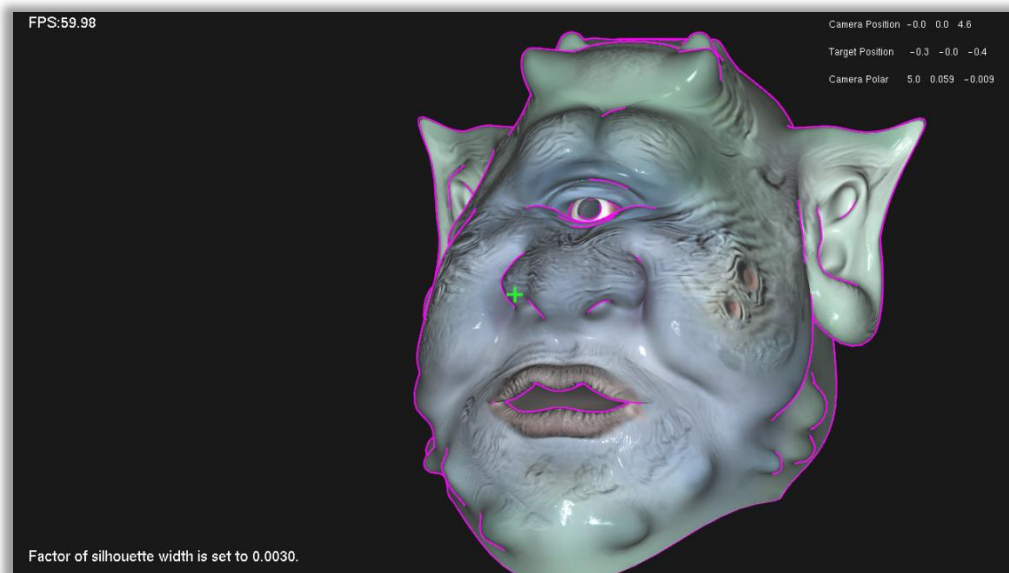


Figure 7 按“-”键减小轮廓线宽度



Figure 8 按“+”键增加轮廓线宽度

五、 实验探索与改进

a) 对深度纹理进行边缘检测以提取轮廓

在完成本次试验的探索过程中，我曾经尝试在深度空间中进行边缘检测来获得物体的轮廓，并且取得了一定的成效（Figure 9 和 Figure 10）。但是因为难以在这种方法下调整物体的轮廓线宽度而放弃，改用了现在的方法。

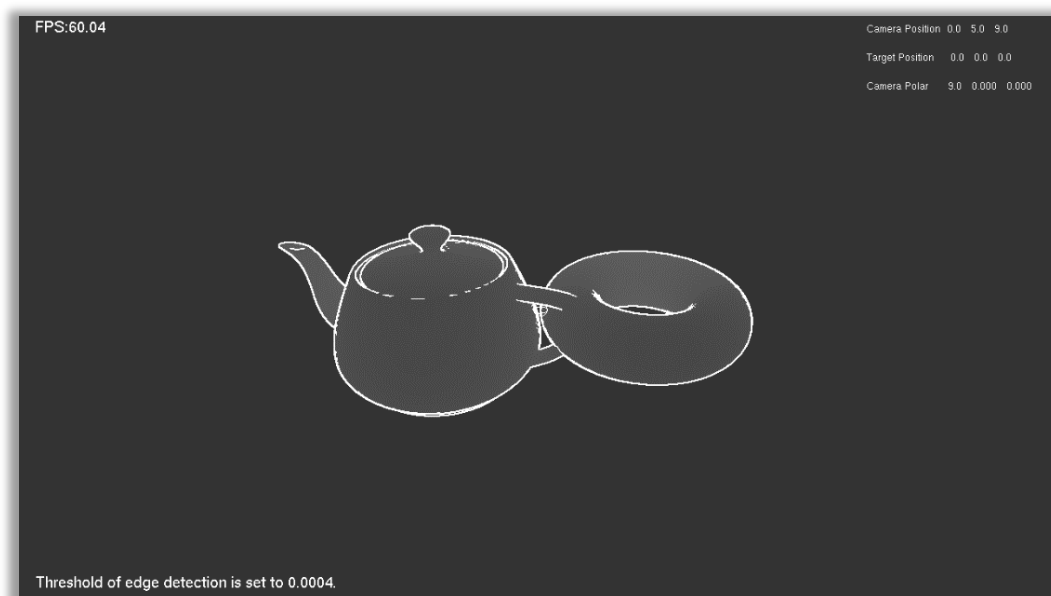


Figure 9 深度检测（一）



Figure 10 深度检测（二）

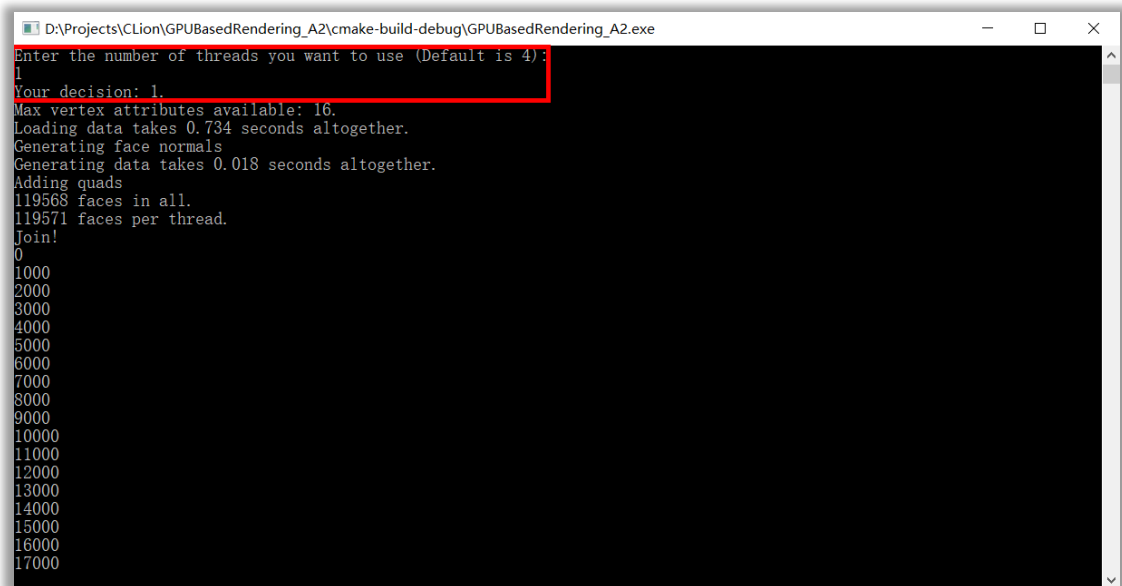
b) 启用多线程

原本的系统使用单线程顺序执行，不论是读取 OBJ 文件，还是为模型添加“候选轮廓线”，都会为场景的加载带来极长时间的阻塞。为此我将模型的面、边和顶点的分析过程安排在不同的线程，操作系统会自动将这些线程任务安排到不同的 CPU 核心上，为计算任务提供一定的并行性（Figure 11）。

```
// Create quads under multiple threads
auto facesInAll = static_cast<int>(faces.size());
cout << facesInAll << " faces in all." << endl;
int facesPerThread = (facesInAll / numOfThreads) + (3 - (facesInAll / numOfThreads) % 3);
cout << facesPerThread << " faces per thread." << endl;
for (int i = 0; i < numOfThreads - 1; i++) {
    threads.emplace_back(create, i * facesPerThread, (i + 1) * facesPerThread, i);
}
threads.emplace_back(create, (numOfThreads - 1) * facesPerThread, facesInAll, numOfThreads - 1);
```

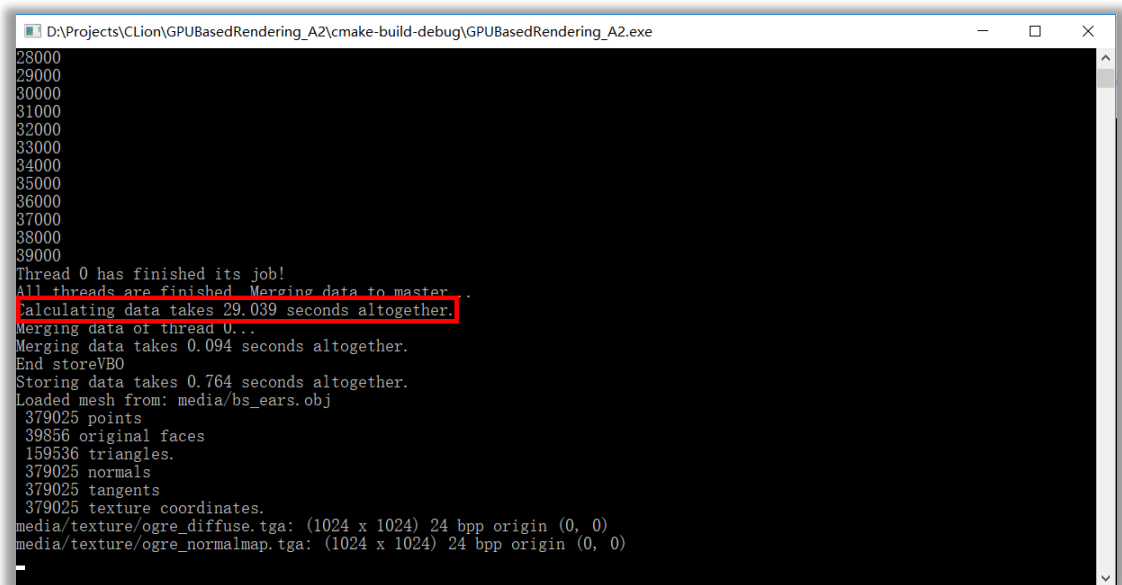
Figure 11 451:5@vbomesh.cpp

本次试验使用的模型需要处理的面数约 4 万个。当选择单线程执行计算任务时 (Figure 12), 计算过程的耗时约为 29 秒 (Figure 13); 使用 4 线程计算时 (Figure 14), 每个线程被分配到约 1 万个面, 计算耗时约 7.2 秒 (Figure 15); 使用 16 线程计算时 (Figure 16), 每个线程被分配到约 2400 个面, 计算耗时约 2.4 秒 (Figure 17); 使用 32 线程 (16 核心超线程得到 32 线程) 计算时 (Figure 18), 每个线程被分配到约 1200 个面, 计算耗时约 2.0 秒 (Figure 19)。



```
D:\Projects\CLion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
Enter the number of threads you want to use (Default is 4):
1
Your decision: 1.
Max vertex attributes available: 16.
Loading data takes 0.734 seconds altogether.
Generating face normals
Generating data takes 0.018 seconds altogether.
Adding quads
119568 faces in all.
119571 faces per thread.
Join!
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
```

Figure 12 线程数: 1



```
D:\Projects\CLion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
Thread 0 has finished its job!
All threads are finished. Merging data to master...
Calculating data takes 29.039 seconds altogether.
Merging data of thread 0...
Merging data takes 0.094 seconds altogether.
End storeVBO
Storing data takes 0.764 seconds altogether.
Loaded mesh from: media/bs_ears.obj
379025 points
39856 original faces
159536 triangles.
379025 normals
379025 tangents
379025 texture coordinates.
media/texture/ogre_diffuse.tga: (1024 x 1024) 24 bpp origin (0, 0)
media/texture/ogre_normalmap.tga: (1024 x 1024) 24 bpp origin (0, 0)
```

Figure 13 线程数为 1, 耗时 29.039 秒


```
D:\Projects\CLion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
Enter the number of threads you want to use (Default is 4):
4
Your decision: 4.
Max vertex attributes available: 16.
Loading data takes 0.757 seconds altogether.
Generating face normals
Generating data takes 0.019 seconds altogether.
Adding quads
119568 faces in all.
29895 faces per thread.
Join!

10000
20000
30000
1000
11000
21000
31000
2000
12000
22000
32000
3000
13000
23000
33000
```

Figure 14 线程数：4

```
D:\Projects\CLion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
9000
19000
29000
39000
Thread Thread 13 has finished its job! has finished its job!
Thread 2 has finished its job!
Thread 0 has finished its job!
Join!
Join!
Join!
All threads are finished. Merging data to master...
Calculating data takes 7.216 seconds altogether.
Merging data of thread 0...
Merging data of thread 1...
Merging data of thread 2...
Merging data of thread 3...
Merging data takes 0.163 seconds altogether.
End storeVBO
Storing data takes 0.117 seconds altogether.
Loaded mesh from: media/bs_ears.obj
379025 points
39856 original faces
159536 triangles.
379025 normals
379025 tangents
379025 texture coordinates.
media/texture/ogre_diffuse.tga: (1024 x 1024) 24 bpp origin (0, 0)
media/texture/ogre_normalmap.tga: (1024 x 1024) 24 bpp origin (0, 0)
```

Figure 15 线程数为 4，耗时 7.216 秒

```
E:\Projects\CLion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
Enter the number of threads you want to use (Default is 4):
16
Your decision: 16.
Max vertex attributes available: 16.
Loading data takes 0.856 seconds altogether.
Generating face normals
Generating data takes 0.020 seconds altogether.
Adding quads
119568 faces in all.
7476 faces per thread.
0
Join!
5000
10000
15000
20000
25000
30000
35000
40000
13000
23000
30000
18000
33000
```

Figure 16 线程数：16

```
E:\Projects\Clion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
Join!
Join!
Join!
Join!
Thread 13 has finished its job!
Join!
Thread 14 has finished its job!
Join!
All threads are finished. Merging data to master...
Calculating data takes 2.432 seconds altogether.
Merging data of thread 0...
Merging data of thread 1...
Merging data of thread 2...
Merging data of thread 3...
Merging data of thread 4...
Merging data of thread 5...
Merging data of thread 6...
Merging data of thread 7...
Merging data of thread 8...
Merging data of thread 9...
Merging data of thread 10...
Merging data of thread 11...
Merging data of thread 12...
Merging data of thread 13...
Merging data of thread 14...
Merging data of thread 15...
Merging data takes 0.206 seconds altogether.
End storeUBO
Storing data takes 0.361 seconds altogether.
Loaded mesh from: media/bs_ears.obj
379025 points
39856 original faces
159536 triangles.
379025 normals
379025 tangents
379025 texture coordinates.
media/texture/ogre_diffuse.tga: (1024 x 1024) 24 bpp origin (0, 0)
media/texture/ogre_normalmap.tga: (1024 x 1024) 24 bpp origin (0, 0)
```

Figure 17 线程数为 16，耗时 2.432 秒

```
E:\Projects\Clion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
Enter the number of threads you want to use (Default is 4):
32
Your decision: 32.
Max vertex attributes available: 16.
Loading data takes 0.853 seconds altogether.
Generating face normals
Generating data takes 0.019 seconds altogether.
Adding quads
119568 faces in all.
3738 faces per thread.
0
5000
10000
15000
20000
25000
```

Figure 18 线程数: 32

```
E:\Projects\CLion\GPUBasedRendering_A2\cmake-build-debug\GPUBasedRendering_A2.exe
Join!
Join!
Join!
All threads are finished. Merging data to master...
Calculating data takes 2.028 seconds altogether.
Merging data of thread 0...
Merging data of thread 1...
Merging data of thread 2...
Merging data of thread 3...
Merging data of thread 4...
Merging data of thread 5...
Merging data of thread 6...
Merging data of thread 7...
Merging data of thread 8...
Merging data of thread 9...
Merging data of thread 10...
Merging data of thread 11...
Merging data of thread 12...
Merging data of thread 13...
Merging data of thread 14...
Merging data of thread 15...
Merging data of thread 16...
Merging data of thread 17...
Merging data of thread 18...
Merging data of thread 19...
Merging data of thread 20...
Merging data of thread 21...
Merging data of thread 22...
Merging data of thread 23...
Merging data of thread 24...
Merging data of thread 25...
Merging data of thread 26...
Merging data of thread 27...
Merging data of thread 28...
Merging data of thread 29...
Merging data of thread 30...
Merging data of thread 31...
Merging data takes 0.207 seconds altogether.
End storeUBO
Storing data takes 0.354 seconds altogether.
Loaded mesh from: media/bs_ears.obj
379025 points
39856 original faces
159536 triangles.
379025 normals
379025 tangents
379025 texture coordinates.
media/texture/ogre_diffuse.tga: (1024 x 1024) 24 bpp origin (0, 0)
media/texture/ogre_normalmap.tga: (1024 x 1024) 24 bpp origin (0, 0)
Space pressed. Model starts/stops rotating.

Focus is on this window.
Mouse moves to (287, 126)
Focus is on other window.
-
```

Figure 19 线程数为 32，耗时 2.028 秒

c) 使用法线贴图

原来的素模太丑，于是我为模型加上了纹理贴图和法线贴图，不仅需要多准备一些数据，而且更丑了。

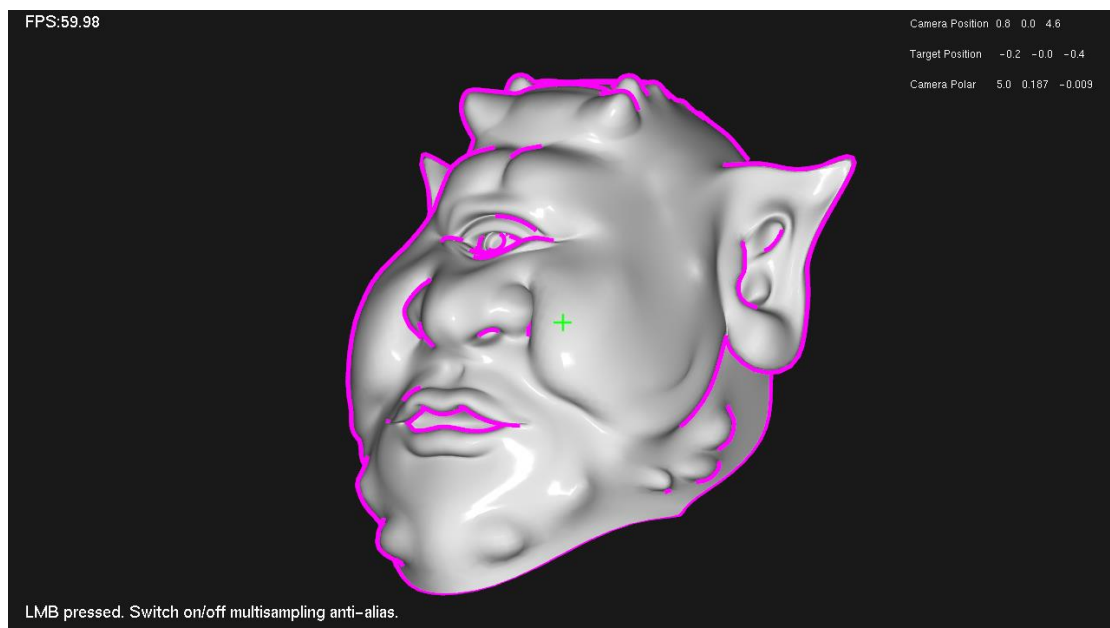


Figure 20 未使用纹理贴图和法线贴图

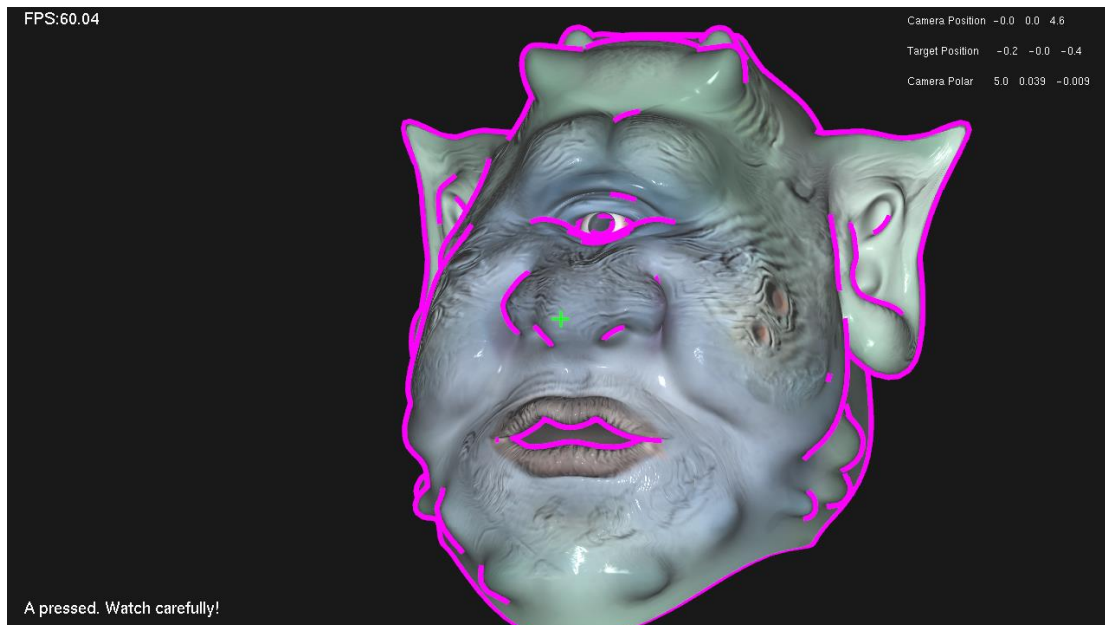


Figure 21 使用纹理贴图和法线贴图

六、 实验总结与心得

a) 由实验产生的思考

i. 思路

本次实验的要求是使用顶点着色器来实现顶点偏移。由于顶点着色器本身只能做到根据输入的顶点属性来输出顶点坐标和其他属性，并且在计算过程中对顶点附近的顶点、边、面的信息一无所知，所以想要真正地得到联结信息，只有通过顶点属性来输入。而顶点着色器作为 OpenGL 渲染管线的第一步，在不使用 Deferred shading 或多次渲染的条件下，必须经由 CPU 端将所需数据装配好。另一种思路是在模型空间中计算顶点法线的朝向来粗略判断顶点是否处在边缘线附近。它利用了这样的事实：一般如果一个顶点的法向量与摄像机视线接近垂直，那么这个顶点很有可能处在边缘线附近。但是这种方法也有缺点，即它能应对的模型应该具有光滑的表面（轮廓线附近有足够的顶点）。若将条件扩大至使用几何着色器以外的着色器来实现轮廓线的绘制，我们还可以利用深度纹理，在深度空间寻找深度变化较大的部分，将它标记为轮廓线所在区域。

ii. 投影

当着色器与视线方向有关时，使用透视投影还是平行投影来观察很有可能影响到显示效果。在本次试验中，我不得不将透视投影修改为平行投影，在取得了不错的效果。

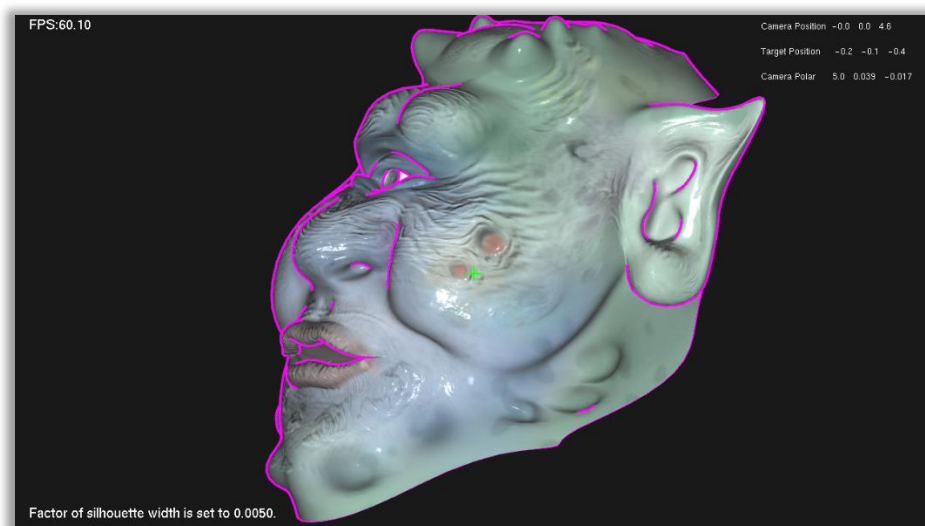


Figure 22 侧面观察（一）

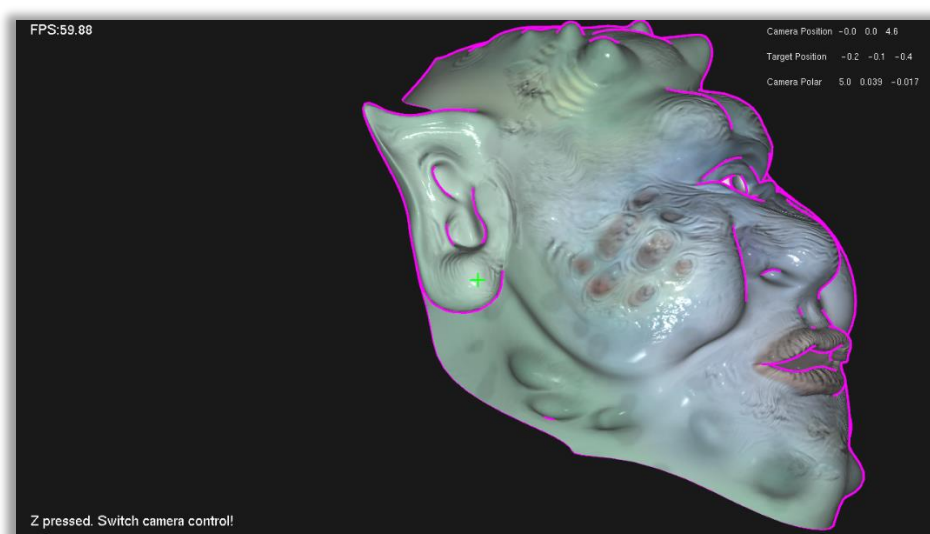


Figure 23 侧面观察（二）

b) 现在的方法的缺点

i. 计算耗时

单线程条件下加载场景的总耗时超过 30 秒。使用多线程才有所改观。

ii. 遮挡

由于深度测试的启用，新增的“候选轮廓线”作为普通的三角面图元，在一定条件下可能被模型本身所遮挡，导致轮廓线断开，如 Figure 1 模型的鼻翼处。

iii. 顶点数目激增

顶点数由原本的 2 万个增加到 38 万个，面数由原本的 12 万个增加到 38 万个，这将为显卡增加负担：显示一个普通的具有法线贴图的模型，对实验所用 GTX 960M 的负荷为 24% (Figure 24)；在此基础之上增加轮廓线的模块，负荷为 51% (Figure 25)。显卡的工作量增加了一倍。但是在 K620 上却几乎没有增加负荷，原因不明。

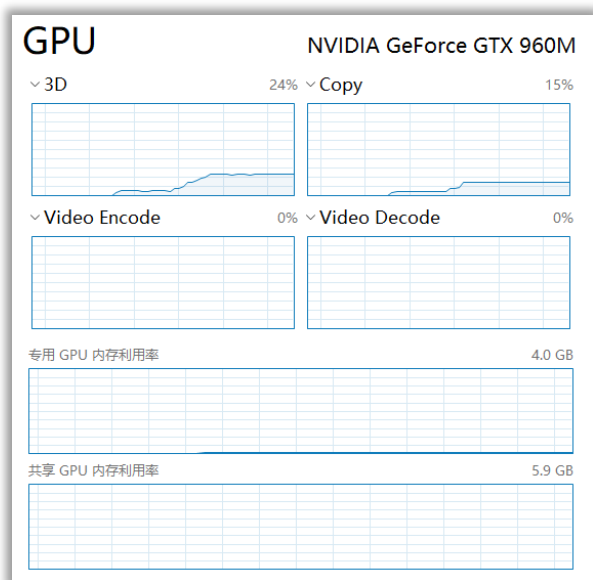


Figure 24 使用普通模型时 GTX 960M 负载为 24%

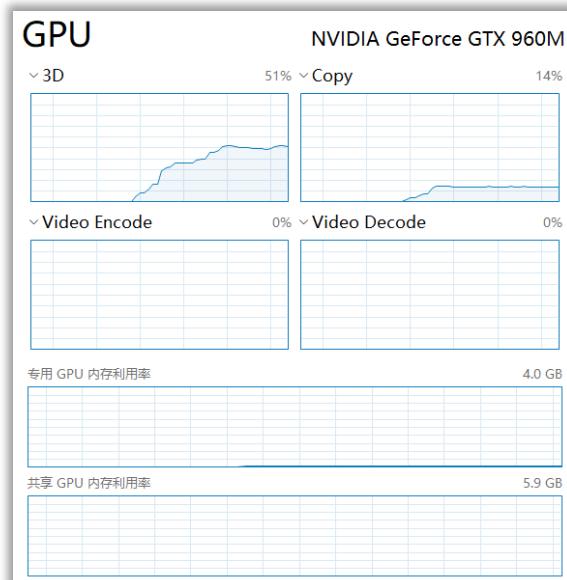


Figure 25 显示轮廓线时 GTX 960M 负载为 51%

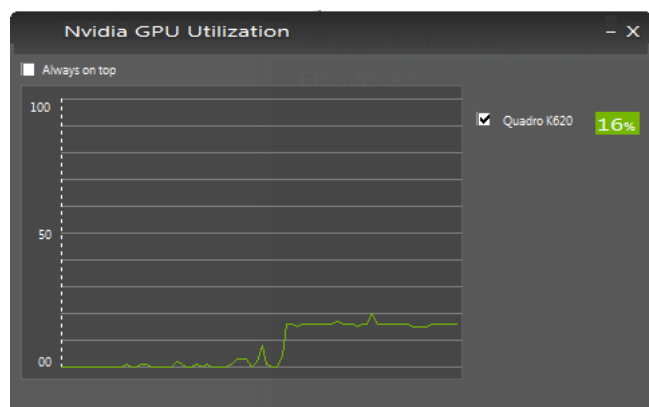


Figure 26 使用普通模型时 K620 负载为 16%

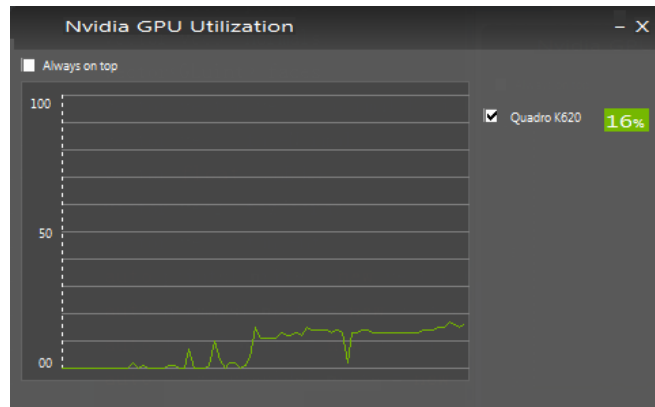


Figure 27 显示轮廓线时 K620 负载为 16%

c) 实验体会

经过本次实验，我对模型预处理和顶点着色器有了基本的了解，学会了如何用它们来创建新的顶点和添加顶点属性。为了完成本次实验，花了很长时间，失败了很多次，收获很大。