# 計算機圖學 HW2
# 309553012 黃建洲

1. Explain how you implement the above reqirements.

以下說明我整體作業的程序:

(1) main.cpp:

首先，為了使c++檔案能夠讀取vertex shader檔和fragment shader檔，我首先在initShader()中使用幾項專案已經幫我們寫好的含式進行。先使用ReadShader(path)分別讀取所有的shader檔案，再使用CreateShader()將shader檔轉換格式，最後使用CreateProgram使我的程式碼可以使用該shader。範例如下:

```
char* ph_vertexShaderSource = ReadShader("Resources/shaders/phongFragment.vert");
char* ph_fragmentShaderSource = ReadShader("Resources/shaders/phongFragment.frag");
CreateShader(vertexShader, GL_VERTEX_SHADER, ph_vertexShaderSource);
CreateShader(fragmentShader, GL_FRAGMENT_SHADER, ph_fragmentShaderSource);
CreateProgram(PhongProgram, 2, vertexShader, fragmentShader);
```

第二，我需要將bunny的object file寫入opengl，專案中已經幫我們將object檔parse成可簡單取用的形式，position裡面包含所有faces的vertex position(故size為3 * faces_num的vector3)，texcoord包含貼上texture所需要的座標資訊，normals則包含vector normal資訊。因此，我建立了一個專案要求的structure格式，並將所有資料以float array的形式存取，如下圖

```
struct VertexAttribute
{
    GLfloat position[4];
    GLfloat normal[3];
    GLfloat texcoord[2];
};
```

```
b_vertices = new VertexAttribute[196890];
for (int i = 0; i < positions.size(); i++) {
    b_vertices[i].position[0] = positions[i].x;
    b_vertices[i].position[1] = positions[i].y;
    b_vertices[i].position[2] = positions[i].z;
    b_vertices[i].position[3] = 1;
    b_vertices[i].normal[0] = normals[i].x;
    b_vertices[i].normal[1] = normals[i].y;
    b_vertices[i].normal[2] = normals[i].z;
    b_vertices[i].texcoord[0] = texcoords[i].x;
    b_vertices[i].texcoord[1] = texcoords[i].y;
    //std::cout << positions[i].x << " " << positions[i].y << " " << positions[i].z << std::endl;
}
```

讀取完資料後，在BindBuffer()中將這些資料以vertexAttributePointer的方式匯入vertex shader檔，如下圖。

```
glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(VertexAttribute) * 196890, b_vertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, position)));
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, normal)));
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, texcoord)));

glBindBuffer(GL_ARRAY_BUFFER, 0);

glBindVertexArray(0);
```

最後，在RenderObject()中要將shader檔案中所需要使用的uniform格式的資料傳入。除了作業中已經幫我們寫好的M、mainTex、WorldLightPos和WorldCamPos外，我在這次作業的三個shader中仍須匯入的項目為Viewer matrix V、Projection matrix P、noise Texture、dissolve中需要的edge length、以及ramp texture。剩餘的參數我選擇在shader file裡面進行宣告。

(2) vertex shader:

在vertex shader的檔案中，首先要以layout(location = id)的方式讀取之前從main.cpp傳出的vertex position、normal和texture coordinate。如下圖

```
layout(location = 0) in vec4 pos;
layout(location = 1) in vec3 norm;
layout(location = 2) in vec2 texCoord;
```

接著以uniform的形式讀取main.cpp傳出的Model matrix M、Viewer Matrix V、Projection Matrix P。如下圖:

```
uniform mat4 M;
uniform mat4 V;
uniform mat4 P;
```

並定義我們在vertex shader檔案中需要傳給fragment shader檔案的值(out)，這次作業中我統一傳texture coordinate、normal和vertex position的xyz維度。

最後使用gl_Position定義shader應該要工作的絕對座標。整體如下。

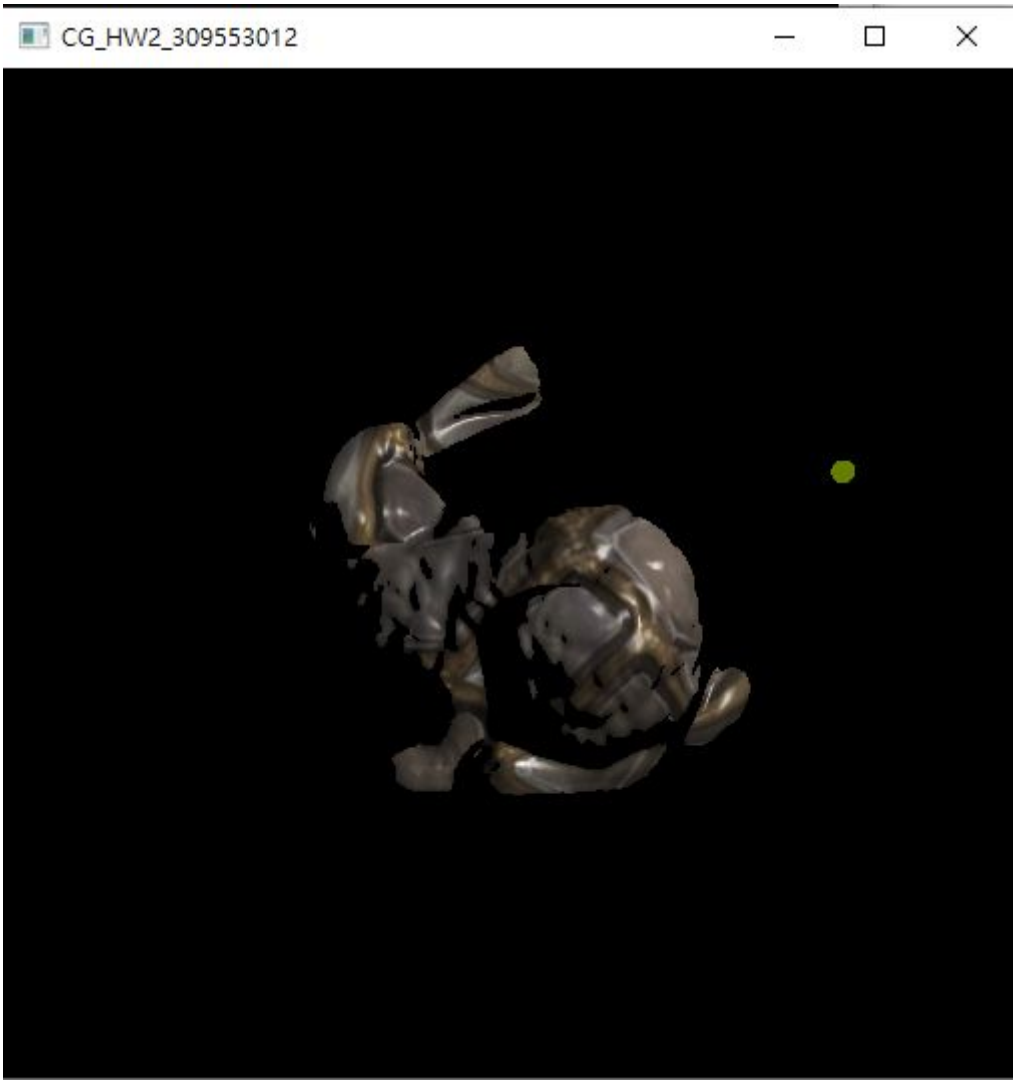
```
out vec2 uv;

out vec3 self_pos;
out vec3 normal;
void main() {
    gl_Position = P * V * M * pos;
    uv = texCoord;
    normal = norm;
    self_pos = vec3(pos.x,pos.y,pos.z);
}
```

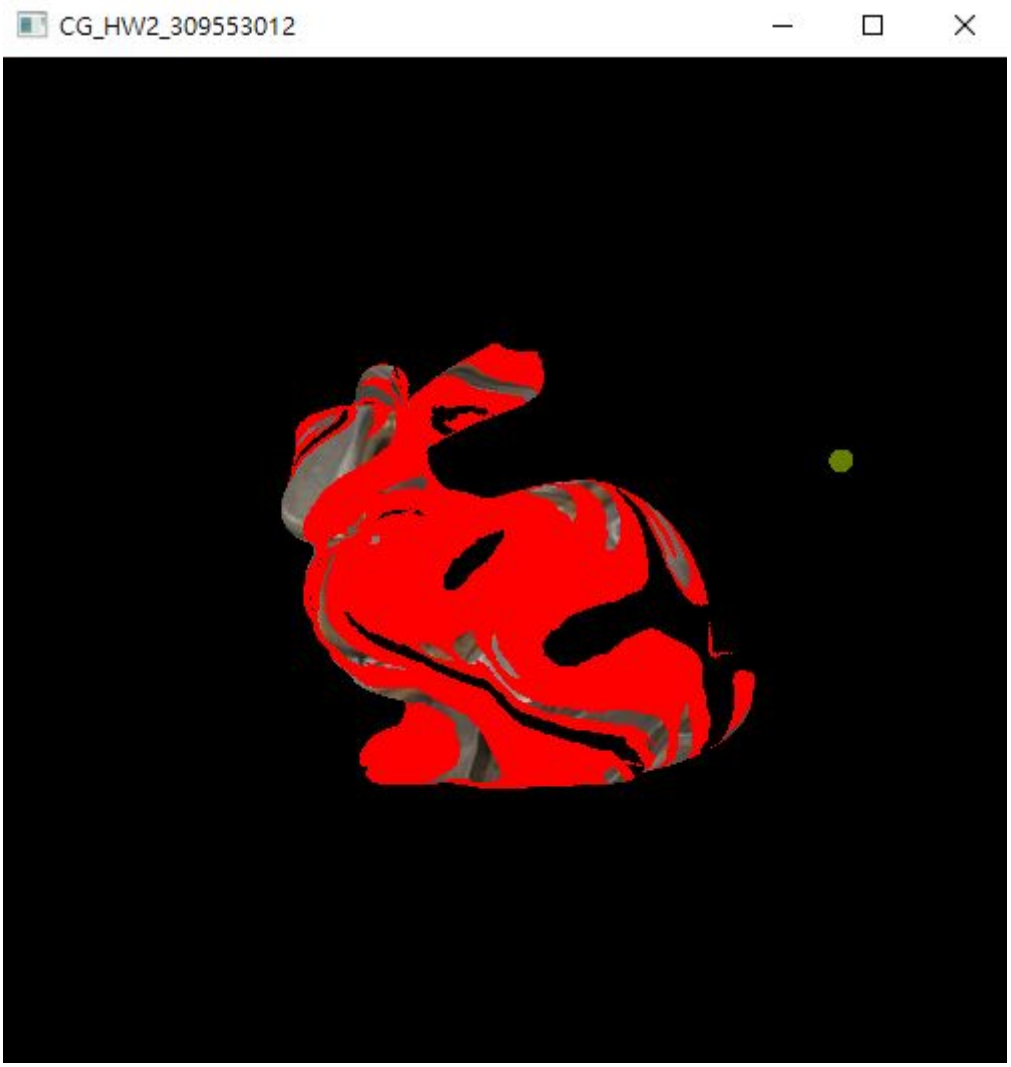
(3) Fragment shader:

在這個部分中，我依照hint中所給的虛擬碼進行，並分別做出phong model shader、dissolve shader和ramp effect shader。
其中於Phong Model比較特別的參數:
L: 光源相對於物件的normal方向
R: 光源完美反射後的向量
N: normal vector
V: Camera位置相對於物件的方向
H: Half Way，依照課本進行定義

Phong shader 結果:




dissolve shader結果(edge length = 0.4, edge color為紅色):




ramp effect shader:


## 2. Describe the problems you met and how you solved them.

在這次作業中我遇到的第一個問題是不知從何開始做起。由於只看main.cpp比較難了解他整體的架構，因此感到十分困惑。後來我決定從畫出兔子開始，故先關掉shader完成了BindBuffer的部分，可以得到一隻綠色的兔子。

接著開始進行shader的時候我遇到了第二個困難，雖然資料傳入皆依照hint的作法，但由於寫shader難以debug的特性，往往難以得知錯誤出現在哪裡。最後我決定使用一份能運作的最基本的架構(能強制把兔子變成藍色的架構)，每加入一行程式碼就跑一次，確定每個環節都是對的才繼續進行。

## 3. Illustrate extra features of your design. (optional)

在extra features中，我實作了toon shader，以dot(normal, lightDir)表示為強度，根據強度將各個區塊分層，每一層使用不同的光度，營造一種"畫出來"的感覺。

Fragment Code如下

```
void main() {
    vec3 L = normalize(WorldLightPos - self_pos);
    vec3 N = normalize(normal);
    rampCoord = dot(N,L) * 0.5 + 0.5;
    diffuse = texture(rampTex, vec2(rampCoord, rampCoord));
    albedo = texture2D(mainTex, uv);

    float intensity = dot(L,N);
    vec4 color2;
    if(intensity > 0.95) color2 = vec4(1,1,1,1);
    else if(intensity > 0.75) color2 = vec4(0.8,0.8,0.8,1);
    else if(intensity > 0.50) color2 = vec4(0.6,0.6,0.6,1);
    else if(intensity > 0.25) color2 = vec4(0.4,0.4,0.4,1);
    else color2 = vec4(0.2,0.2,0.2,1);

    outColor = albedo * color2;
}
```

結果如下