ICG HW3 Report

I. Blinn-Phong shading

- Vertex Shader: 在這裡計算每個 vertex 的位置(FragPos)和法向量(Normal)，並傳到 fragment shader 中。

  1. FragPos: 先投影到世界座標，再取其中的 x, y, z 項
  → FragPos = vec3(M * vec4(aPos, 1.0))
  2. Normal: 經由轉換公式更新法向量並正規化。
  →Normal = normalize(mat3(transpose(inverse(M))) * aNormal)

- Fragment Shader: 在這裡計算每個 fragment 的顏色。須用到以下幾個向量；接著代 Phong Reflection Model 的公式算亮度；最後乘上 texture 得到最終顏色。

  1. L: 指向光源的向量，所以由光-位置而得，需正規化。
  → L = normalize(LightPos-FragPos)
  2. N: 法向量，使用 vertex shader 傳進來的 Normal。
  → N = normalize(Normal)
  3. Viewer: 指向相機的向量，所以由相機-位置而得。
  → Viewer = normalize(C-FragPos)
  4. H: 光線和視線的一半夾角，所以由 L 和 Viewer 計算而得。
  → H = normalize(L+Viewer)
  5. 代 Phong Reflection Model 公式得到 Ambient, Diffuse, Specular，而這裡的 Specular 是用半角公式計算。
  6. Ambient, Diffuse, Specular 相加得到整體亮度，乘 texture 就是最終顏色。

II. Gouraud shading

- Vertex Shader: 在這裡用 Phong Reflection Model 計算每個 vertex 的亮度(light)，並傳到 fragment shader 中。

  1. L, N, Viewer: 計算方式同 I
  2. R: 完美鏡反射的向量，由 L 與 N 的關係式計算而得。
  → R = normalize(2*dot(L, N)*N-L)
  3. 代 Phong Reflection Model 公式得到 Ambient, Diffuse, Specular。這裡的 Specular 是用原始公式計算的。
  4. light = Ambient + Diffuse + Specular

- Fragment Shader: 在這裡計算每個 fragment 的最終顏色。
  1. vertex shader 傳進來的 light 和 texture 相乘就是最終顏色。

III. Flat shading

- Vertex Shader: 在這裡計算每個 vertex 的位置(Pos)並傳到 geometry shader。
  1. Pos: 計算方式同 I-FragPos

- Geometry Shader: 在這裡計算每個 polygon 的法向量(polyNormal)，並連同 vertex shader 傳進來的 Pos 和 texture 一起存到 triangle strip 中，傳到 fragment shader。
  1. polyNormal: 先從 vertex shader 傳進來的 Pos 得到 polygon 三個頂點的位置；接著做兩點相減得到兩個向量，定義此 polygon 的法向量為這兩個向量做外積並正規化。
  → p0 = pos_in[0].Pos
  p1 = pos_in[1].Pos
  p2 = pos_in[2].Pos
  polyNormal = normalize(cross(p1-p0, p2-p0))

- Fragment Shader: 在這裡計算每個 fragment 的顏色。須用到以下幾個向量；接著代 Phong Reflection Model 的公式算亮度；最後乘上 texture 得到最終顏色。
  1. L, N, Viewer, R: 計算方式同II
  2. 代 Phong Reflection Model 公式得到 Ambient, Diffuse, Specular，三者相加得到整體亮度，再乘 texture 就是最終顏色。

IV. Toon shading

- Vertex Shader: 在這裡計算每個 vertex 的位置(FragPos)和法向量(Normal)，並傳到 fragment shader 中。
  1. FragPos, Normal: 計算方式同 I

- Fragment Shader: 在這裡根據條件判斷，賦予每個 fragment 最終的顏色。
  1. L, N, Viewer, R: 計算方式同II
  2. 為得知 L 和 N 的夾角，計算兩者內積 dot(L, N)
  3. 為得知 Viewer 和 R 的夾角，計算兩者內積 dot(Viewer, R)
  4. L, N 的角度大於 90°時，賦予低亮度。即 L, N 內積小於 0 時，顏色深
  → if (dot(L, N) < 0) FragColor = vec4(0.1, 0.1, 0.1, 1.0)

5. Specular 強時，賦予高亮度。因為 Specular 公式中 Ls, Ks, α 是固定的，所以考慮 Viewer 和 R 的夾角越小 Specular 越強。我設的 thresholds 是 dot(Viewer, R) > 0.8，大概是±37°的範圍

→ if (dot(Viewer, R) > 0.8) FragColor = vec4(0.9, 0.8, 0.8, 1.0)

6. 其餘賦予中亮度

→ FragColor = vec4(0.5, 0.3, 0.2, 1.0)

V. Border effect

- Vertex Shader: 在這裡計算每個 vertex 的位置(FragPos)和法向量(Normal)，並傳到 fragment shader 中。

  1. FragPos, Normal: 計算方式同 I

- Fragment Shader: 在這裡計算每個 fragment 的顏色。為了做出邊界發光效果，原始的 light 以及 color 都做了加強版的亮度(stronglight)與顏色(white)；再依據邊界的判斷式決定加強的程度，得到最終的顏色。

  1. L, N, Viewer, R: 計算方式同II
  2. light: 原始版本的亮度，即一般的 Phong Reflection Model
  3. stronglight: 加強版的亮度，調高 Diffuse, Specular 至最強，即公式中的內積值代 1

  → stronglight = La*Ka + Ld*Kd + Ls*Ks

  4. newlight: 添加效果後的亮度，混合了 light 以及 stronglight。為了做出邊界效果，所以混合比例與邊界的判斷式有關，即 dot(N, Viewer)=0 代表垂直(在邊界)

  → newlight = mix(light, stronglight, 1-dot(N, Viewer))

  5. newcolor: 添加效果後的顏色，混合了 texture(color)以及白色(white)。同樣，混合比例與邊界的判斷式有關

  → newcolor = mix(color, white, (1-dot(N, Viewer))*0.7)

  6. newlight 和 newcolor 相乘就是最終顏色

VI. Dissolve effect

- Vertex Shader: 在這裡傳入一個隨時間變動的浮點數(dissolveFactor)，並根據原始位置的 x(aPos.x)判斷該 vertex 要不要顯示。其餘計算位置(FragPos)和法向量(Normal)。

  1. 以dissolveFactor為基準做比較，如果aPos.x比dissolveFactor小，該點就移出顯示範圍

  → if (aPos.x < dissolveFactor) worldPos = vec4(0.0);

else worldPos = M * vec4(aPos, 1.0);
2. FragPos, Normal: 計算方式同 I

- Fragment Shader: 在這裡計算每個 fragment 的顏色。須用到以下幾個向量；接著代 Phong Reflection Model 的公式算亮度；最後乘上 texture 得到最終顏色。

1. L, N, Viewer, R: 計算方式同II
2. 代 Phong Reflection Model 公式得到 Ambient, Diffuse, Specular，三者相加得到整體亮度，再乘 texture 就是最終顏色。

Problems & Solution

1. 向量計算錯誤

在計算 L 以及 Viewer 向量的時候，應該要減掉世界座標 FragPos，但我一開始寫成減掉模型座標 aPos，結果就變成小黑鹿了。經由重新檢視程式碼，訂正錯誤後就沒問題了。

2. 原始 polygon 不明顯

一開始定義每個 polygon 的法向量時，我用的方法是計算三個頂點分別到三角形中心的距離，並根據各點的距離倒數乘上各自的法向量加總。但這個方式做出來的每塊 polygon 看起來不明顯。後來請教朋友，用現在兩向量外積的方式算法向量，看起來就清楚許多了。