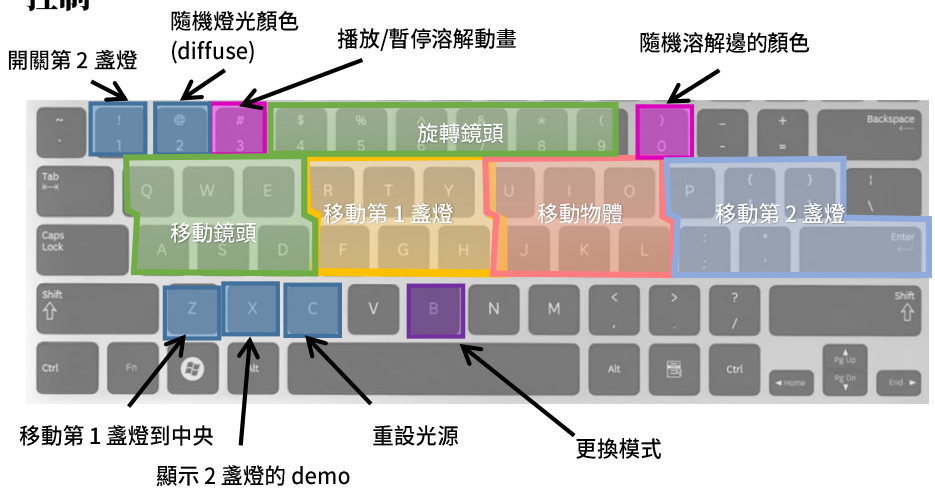
計算機圖學 作業 2 Report　　0756078 陳劭傑

簡介

本次作業為完成三個不同的 shader 效果，Phone lighting、Dissovling 以及 Ramp，在實作我使用單個 shader program 來完成，在裡面切換不同模式來完成繪製。

控制



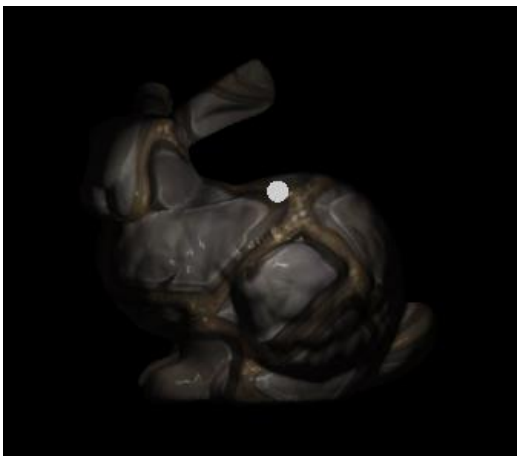
傳資料給 shader

首先用 memcpy 將 glm 載入的 model 重組成如下的結構

Vertex Data			Normal Data			Tex Coord		Vertex Data ...
vx	vy	vz	nx	ny	nz	tu	tv	vx ...

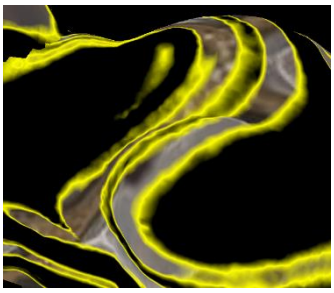
再使用 VBO 搭配 offset 傳資料給 shader

Phone shading



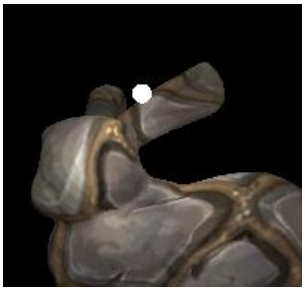
首先，在 vertex shader 中
在 shader 中用 struct 來紀錄 light 的屬性，再用 uniform 的方式在主程式傳入。
為了效率的考量，我把 shader 中變數的 location 先記下，使重繪時不用再使用 glGetUniformLocation 得到 location。

Dissolving



我設定了一個按鍵 3 切換溶解動畫的播放，以及另一個按鍵 0 隨機變換 edge 的顏色，edge 的部份我使用了線性疊加的柔邊，使邊緣看起來不那麼銳利。

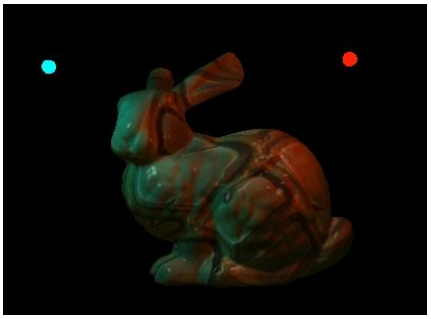
Ramp



首先使用頂點到光源的向量，和 normal 做內積得到基本的亮度，考慮到光的衰減，乘上 1/sqrt(distance)做為調整。
在 Ramp 的對應時我有注意到助教的提示，亮度要對在(0,1)區間內，否則會生成不尋常的色塊。

Bonus

兩盞燈



在 Phone shading 的部份，我實作了第二盞燈，可以使用 1 切換開啟，按 2 可以隨機變換兩盞燈的顏色，按下 X 可以看兩盞等的一個 demo。

Shader Optimization

為了減少 shader 重複計算矩陣，我在 shader 外先計算好一些矩陣的乘法，如 Normal Matrix 及 ModelView-Projection Matrix，如下：

計算 MVP 矩陣

```
void calcMVP(float * mvp, float * mv, float * p)
{
     unsigned i, j, k;
     for (i = 0; i < 4; ++i) {
          for (j = 0; j < 4; ++j) {
               mvp[i + j * 4] = 0.0;
               for (k = 0; k < 4; ++k) {
                    mvp[i + j*4] += p[i + k*4] * mv[k + j*4];
               }
          }
     }
}
```

計算 MVP 矩陣

```
void calcNormalMatrix(float * ans, float * mv)
{
     // inverse + transpose
     ans[0] = mv[5] * mv[10] - mv[6] * mv[9];
     ans[1] = -(mv[4] * mv[10] - mv[6] * mv[8]);
     ans[2] = mv[4] * mv[9] - mv[5] * mv[8];
     ans[3] = -(mv[1] * mv[10] - mv[2] * mv[9]);
     ans[4] = mv[0] * mv[10] - mv[2] * mv[8];
     ans[5] = -(mv[0] * mv[9] - mv[1] * mv[8]);
     ans[6] = mv[1] * mv[6] - mv[2] * mv[5];
     ans[7] = -(mv[0] * mv[6] - mv[2] * mv[7]);
     ans[8] = mv[0] * mv[5] - mv[1] * mv[4];

     float det = ans[0] * ans[4] * ans[8]
          + ans[1] * ans[5] * ans[6]
          + ans[2] * ans[3] * ans[7]
          - ans[2] * ans[4] * ans[6]
          - ans[1] * ans[3] * ans[8]
          - ans[0] * ans[5] * ans[7];
     for (unsigned i = 0; i < 9; ++i) {
          ans[i] /= det;
     }
}
```

計算光源在 modelview 之下的位置

```
     GLfloat light0_localpos[3] = {
          light_modelview[0] * light_pos[0] +
          light_modelview[4] * light_pos[1]  +
          light_modelview[8] * light_pos[2]  +
          light_modelview[12] ,
          light_modelview[1] * light_pos[0]  +
          light_modelview[5] * light_pos[1]  +
          light_modelview[9] * light_pos[2]  +
          light_modelview[13] ,
          light_modelview[2] * light_pos[0]  +
          light_modelview[6] * light_pos[1]  +
          light_modelview[10] * light_pos[2] +
          light_modelview[14]};
```

如此，可以減少 shader 中的重複地乘法計算