

Final Project

112598016 許加宜

1. 專案設置說明：

- 套件：

- OpenGL
- OpenCV

安裝方式按照之前課程所教的即可。

- 檔案結構：

- 主程式：專案/final_112598016.cpp
- 存放 OBJ 檔案的資料夾：專案/obj
- 存放 texture 的資料夾：專案/textures

- 編譯和運行程式：

確保有 final_112598016.cpp、./obj、./textures，即可執行。

2. 專案使用說明：

- 按鍵操作：

- 一般按鍵：

按鍵	功能說明
W	向上移動相機。
A	向左移動相機。
S	向下移動相機。
D	向右移動相機。
P	停止或恢復 timer。
空白鍵 (space)	重置所有參數，包括旋轉角度、平移位移、飛船位置、相機位置。

- 特殊按鍵：

按鍵	功能說明
方向鍵	控制主星球、飛船和相機的旋轉和平移。
↑(向上)	向上移動或旋轉。
↓(向下)	向下移動或旋轉。
←(向左)	向左移動或旋轉。
→(向右)	向右移動或旋轉。
F1	飛船平移模式 開/關。
F2	飛船垂直移動模式 開/關。
F3	飛船旋轉模式 開/關。
F4	相機旋轉模式 開/關。

- **滑鼠操作：**

- 滑鼠滾輪：控制**相機的前後移動**，即視角遠近控制。

滾輪	功能說明
向前	向前移動相機。
向後	向後移動相機。

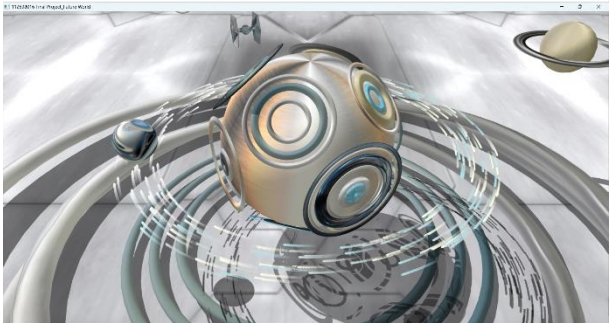
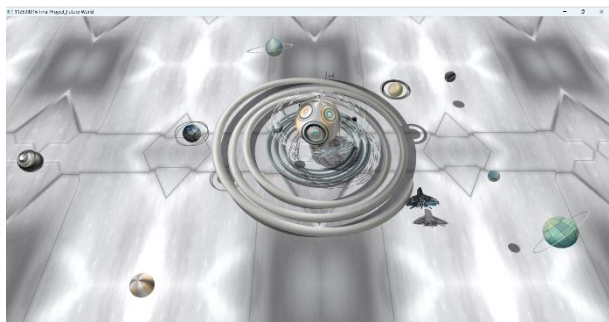

- 按下**滑鼠左鍵**：**拖動來旋轉相機視角**。
 - 按著左鍵，可以往任意方向拖動，旋轉相機視角。
 - 放開左鍵即停止旋轉視角。

3. 程式螢幕截圖：



• 任務完成展示：

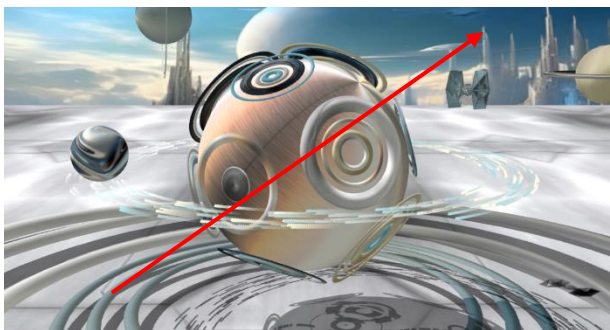
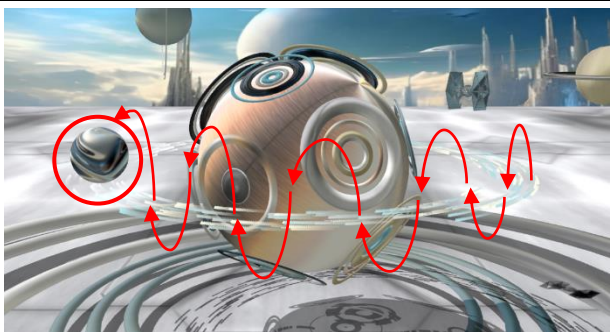
1. 使用 4 張或更多圖像/紋理，應用於不同種類的物件(完成)：

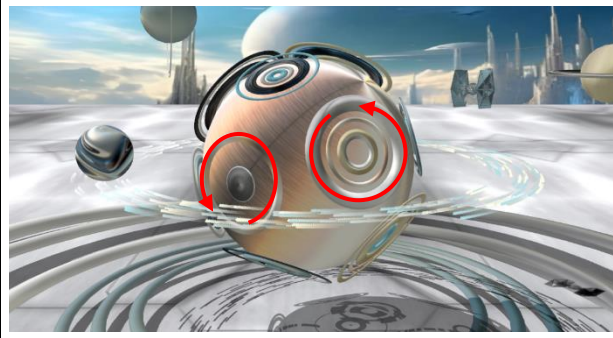
截圖	有貼上 texture 的物件
	<ol style="list-style-type: none"> 1. 主星球 2. 主星球上方的 43 個 Torus 3. cos-like wave 的行星 4. 繞著主星球的 Tie Fighter
	<ol style="list-style-type: none"> 5. 地板 6. 地板上的兩種 Torus 7. 周圍七顆星球運用不同的 texture 8. 飛船
	<ol style="list-style-type: none"> 9. Skybox (一個巨大的球體)

2. 載入與未來世界主題相關的 obj 檔案，並為其添加動畫(完成)：

截圖	說明
	<ol style="list-style-type: none"> 1. Load 飛船的 obj file。 2. 上下垂直浮動。 3. 可以透過鍵盤操作移動/旋轉飛船。
	<ol style="list-style-type: none"> 1. Load Tie Fighter 的 obj file。 2. 會繞著主星球旋轉。

3. 創建各種動畫(完成)：

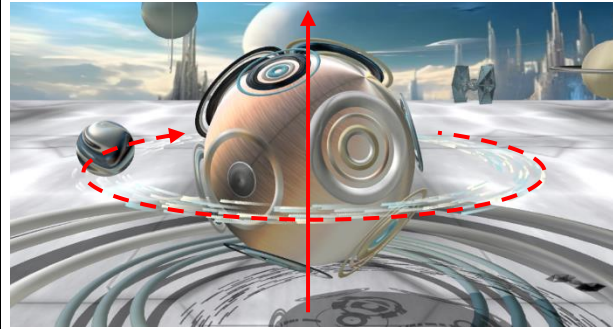
截圖	動畫
	<p>A. 主星球繞其自身軸旋轉：</p> <p>在 DrawMainSphere 函數中。</p> <pre>glPushMatrix(); glRotatef(yRot + MainYRot, 0.0f, 1.0f, 0.0f); glRotatef(yRot + MainXRot, 1.0f, 0.0f, 0.0f); glDrawSphere(sphereRadius, 30, 30);</pre>
	<p>B. 行星以餘弦波形繞主星球旋轉：</p> <p>在 DrawSpaceship 函數中。</p> <pre>glPushMatrix(); glRotatef(-yRot*2.0f, 0.0f, 1.0f, 0.0f); glTranslatef(2.8f, cos(-yRot * PI / 180 * 45) * 0.4, 0.0f); glDrawSphere(0.34f, 33, 33); glPopMatrix();</pre>



C. 更多物件，使用不同類型的動畫：

1. 主星球上 43 個圓環的旋轉和縮放：

在 DrawMainSphere 函數中。



2. 主星球的半透明行星環旋轉：

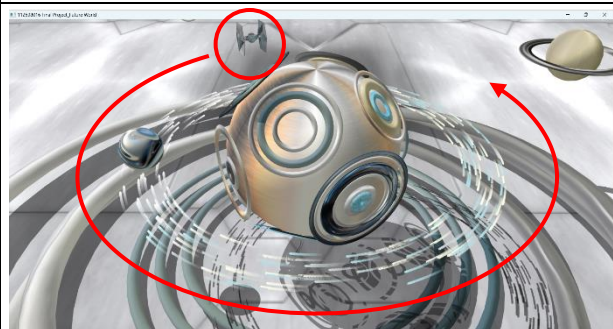
在 DrawMoon 函數中。

```
for (int i = 0; i < NUM_MOON; i++) {
    float theta = -(moon[i][0] + yRot - k * 0.3f);
    float y = moon[i][1] * cos(theta * PI / 180.0f * 11);
    glPushMatrix();
    glRotatef(theta, 0.0f, 1.0f, 0.0f);
    glTranslatef(moon[i][2], 0.0f, 0.0f);
```



3. 飛船的上下浮動：

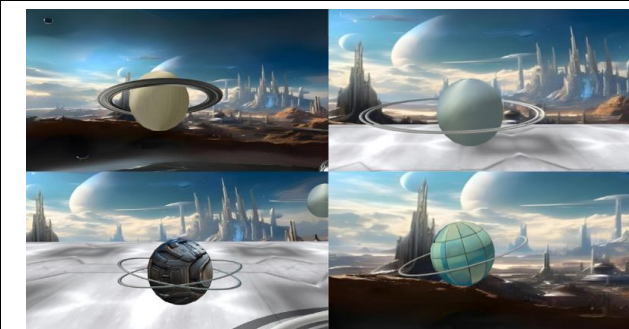
在 DrawSpaceship 函數中。



4. Tie Fighter 繞主星球旋轉：

在 DrawStarWars 函數中。

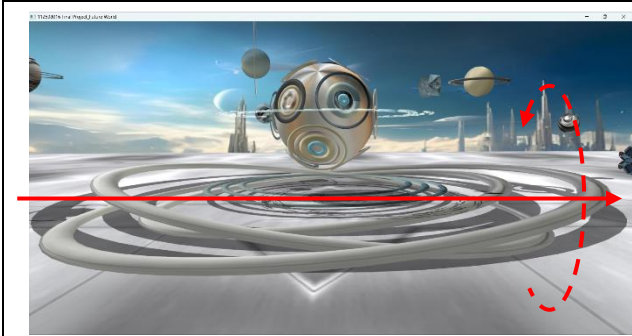
```
glPushMatrix();
glRotatef(-2.0f, 1.0f, 0.0f, 0.0f);
glRotatef(yRot*10.0f, 0.0f, 1.0f, 0.0f);
glTranslatef(3.6f, 1.0f, 0.0f);
```



5. 其他散布的 7 顆星球動畫：

- 星球自身旋轉/平移
- 星球外的行星環旋轉

在 DrawPlanet 函數中。



6. 地板上的圓環旋轉：

在 `DrawPlatform` 函數中，三個環分別進行固定角度的旋轉。

4. 技術困難和解決方案：

1. Skybox：

我嘗試將整個場景放在一個 skybox 中，遇到的問題有：

1. 無法將場景置中於 Skybox 中心。

解決方法：

把 Skybox 移動到相機位置之前獲取相機的位置，確保 Skybox 圍繞相機。

```
// 獲取相機位置
M3DVector3f cameraPos;
frameCamera.GetOrigin(cameraPos);

// 將Sky Dome移動到相機位置
glTranslatef(cameraPos[0], cameraPos[1], cameraPos[2]);
```

2. 將 texture 貼在球體內部後，場景變得很奇怪。

解決方法：

先禁用深度測試，等繪製完 Skybox 再重新啟用。

```
// 禁用深度測試、光照和背面剔除
glDisable(GL_DEPTH_TEST);
glDisable(GL_LIGHTING);
glDisable(GL_CULL_FACE);
glEnable(GL_TEXTURE_2D);
```

3. 圖片解析度過低並且位置難控制，skybox 呈現效果不佳。

解決方法：

背景是下圖 1 紅框處，所以解析度會很低，效果不好。於是比對後將圖片縮小(如圖 2)，再用 AI 擴圖成圖 3，並利用 Canva 調整解析度。



圖 1

圖 2

圖 3

2. Moon :

我想繪製一個圍繞主星球的半透明行星環，遇到的問題有：

1. 無法繪製半透明的漸變效果

解決方法：

1. 首先要使用：

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

使得透明度效果可以正確顯示。

2. 透過迴圈，逐漸增加每一個四面體的透明度。(我是以 300 個四面體呈現行星環效果)

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
for (int k = 30; k >= 0; k--)
    for (int i = 0; i < NUM_MOON; i++) {
        float theta = -(moon[i][0] + yRot - k * 0.3f);
        float y = moon[i][1] * cos(theta * PI / 180.0f * 11);
        glPushMatrix();
        glRotatef(theta, 0.0f, 1.0f, 0.0f);
        glTranslatef(moon[i][2], 0.0f, 0.0f);

        if (nShadow == 0) {
            glBindTexture(GL_TEXTURE_2D, 0);
            glEnable(GL_BLEND);
            glColor4f(moon[i][3], moon[i][4], moon[i][5], 1.0f - k * 0.03)
        }
        else {
            glColor4f(0.00f, 0.00f, 0.00f, 0.5f);
        }
    }
```

2. 半透明的行星環顏色會很詭異

解決方法：

1. 第一是如果將繪製行星環的 DrawMoon 函數放在繪製主星球的 DrawMainSphere 函數之前呼叫，透明部分將與背景的颜色混合，所以當之後主星球渲染時，主星球的颜色會覆蓋在月球的透明部分上，導致透明效果不正確。

下圖 1 為不正確的效果，圖 2 為將 DrawMoon 放在 DrawMainSphere 後面呼叫後的效果，差別請見紅圈處。

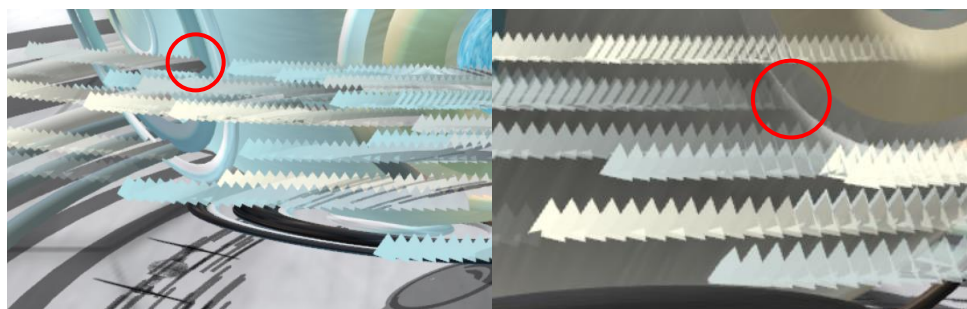


圖 1

圖 2

2. 順序對了之後，行星環的顏色會有一點不自然，後來發現是因為我的環境光 fLowLight 設定為象牙色，於是在 MoonColors 中加上 fLowLight 分量後顏色看起來更自然。如下圖 1 為沒加上環境光分量的結果，圖 2 為修正後的顏色。

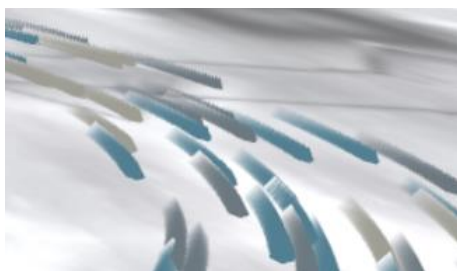


圖 1

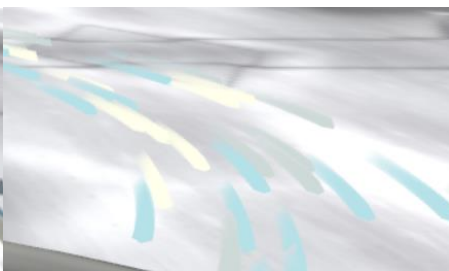


圖 2

3. 主星球表面的 43 個圓環：

我想在主星球上加入大量的圓環，遇到的問題有：

1. 無法將圓環貼齊球體表面

這部分是困擾我最久的地方，一開始把環放上去的時候是插在主星球上(如圖 1)，後來發現不能隨便亂放，要利用球體的坐標系算圓環位置，我是利用笛卡爾坐標系計算。

圓環的坐標由半徑 r 、方位角 θ 和仰角 φ 表示，如下：

- $x = r \times \sin \varphi \times \cos \theta$
- $y = r \times \sin \varphi \times \sin \theta$
- $z = r \times \cos \varphi$

而法向量由球心指向該點的向量來表示。

- $nx = x/r$
- $ny = y/r$
- $nz = z/r$

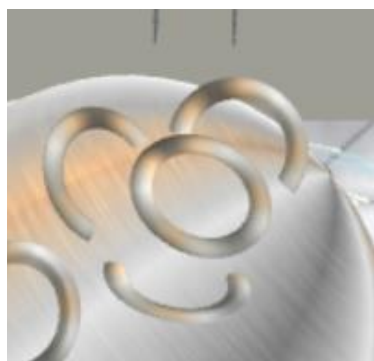


圖 1

並且為了有更好的視覺效果，我放了 43 個圓環，並分別幫每個圓環設置不同粗細、大小、texture，以及縮放、旋轉的動畫，花了非常多時間。

4. Load obj file :

將 OBJ 檔案 Load 進程式中遇到的問題有：

1. OBJ 檔案尺寸太大/太小

在讀 OBJ 檔案的時候，載入的模型尺寸不符合我的預期，並且太大的尺寸在我渲染的時候會花非常多時間(我找了很多 OBJ 檔案，最後只選定效果最好的兩個。所以在不停 Load 進場景看效果會花很多時間)。

解決方法：

在下載各個 OBJ 檔案後，先打開看每個頂點的座標，判斷物體大致大小。接著使用自己寫的 python 程式 scale_obj.py (一同附在./obj 資料夾中)將 OBJ 檔案縮放成我理想的大致比例。

2. OBJ 檔案渲染結果破碎

在渲染模型時，有一些模型非常破碎(如圖 1，為我原本想用的飛船造型)，我原本以為是我使用三角形繪製的結果，於是將渲染模型從 `glBegin(GL_TRIANGLES);` 改成 `glBegin(GL_TRIANGLE_STRIP);`，但結果還是很糟糕，最後只好放棄，選了比較不破碎的模型。這個問題我到現在還是不知道是我程式有問題，還是免費的 3D 模型 Obj 檔案有問題。



圖 1

5. 滑鼠左鍵移動視角：

1. 捕捉滑鼠的移動事件

解決方法：

1. 為了能夠捕捉滑鼠的移動事件，我在滑鼠按下和鬆開時更新 `isLeftMouseButtonPressed` 變數，並記錄滑鼠的位置。

```
void My_Mouse(int button, int state, int x, int y) {  
    if (button == GLUT_LEFT_BUTTON) {  
        if (state == GLUT_DOWN) {  
            isLeftMouseButtonPressed = true;  
            lastMouseX = x;  
            lastMouseY = y;  
        }  
        else if (state == GLUT_UP) {  
            isLeftMouseButtonPressed = false;  
        }  
    }  
}
```

2. 使用 `MouseMotion`：當滑鼠移動時，如果左鍵按下就更新相機的旋轉角度。

```
void MouseMotion(int x, int y) {  
    if (isLeftMouseButtonPressed) {  
        int dx = x - lastMouseX;  
        int dy = y - lastMouseY;  
  
        cameraAngleX += dx * 0.2f;  
        cameraAngleY += dy * 0.2f;  
  
        lastMouseX = x;  
        lastMouseY = y;  
  
        glutPostRedisplay(); // 重新繪製場景  
    }  
}
```