**電腦圖學**

**Computer Graphics**

**Final Team Project**

**星空煙火舞台**

第八組

資工碩一

113598021 李彥儒

113598032 張字青

113598043 張育丞

113598093 陳怡茜

June 20, 2025

**目錄**

[一、專題簡介 1](#_Toc201338970)

[二、專題功能總覽 1](#_Toc201338971)

[三、設計說明 17](#_Toc201338972)

[四、技術實作難點與解法 17](#_Toc201338973)

[五、按鍵操作說明 18](#_Toc201338974)

[六、專案結構 19](#_Toc201338975)

[七、成果展示 21](#_Toc201338976)

[八、總結 22](#_Toc201338977)

[九、心得 22](#_Toc201338978)

# **一、專題簡介**

本專題以「星空煙火舞台」為設計核心，融合星空背景、煙火特效、撲克牌與機器人等元素進行整合，進而打造魔幻世界主題，以下是專題的各項特色：

1. 夜晚星空背景作為全景舞台布幕，採用全畫面貼圖方式渲染。
2. 煙火粒子特效遍布天際，並具備使用者觸發與自動連續施放兩種模式。
   * 使用者滑鼠觸發式爆炸（依據點擊位置計算三維射線）。
   * 數字鍵 4 啟動連續 5 秒的高空視角之自動高密度煙火秀。
3. 撲克牌裝置與草叢裝飾組成五組裝置藝術，當按下數字鍵 5 時將進入旋轉觀賞模式。
4. 機器人具備完整移動與手腳動畫，可切換第一人稱、第三人稱、遠距與俯視視角。

除了將機器人改為可控制移動，更載入多個 OBJ 三維物件模型製作有趣的動畫，以及利用TGA 平面影像將作為三維物件模型材質外，也增添星空場景、在互動煙火，撲克牌互動、多視角切換等增強互動與趣味性，以此實現魔幻世界。

# **二、專題功能總覽**

為完善專題功能，因此在系統方面以「物件貼圖」、「OBJ 載入」、「動畫控制」、「多視角切換」、「火焰粒子控制」、「光影變化實作」、「背景貼圖」及「機器人控制」等八項技術，讓整體互動更加奇妙魔幻。

1. 貼圖系統：使用九張的TGA 影像，應用於地面鋪設、太陽陰影、草叢色彩、桌子紋理、撲克牌印花及星空背景等多個元素。
2. OBJ 載入：將bush.obj、pottedPlant.obj、diningTable.obj和playingCard.obj等多個三維模型載入於系統中，並對其進行外觀及紋理等處理。
3. 動畫控制：具備Y軸自轉及攝影機的Pitch / Yaw 等操控，更加入環繞移動等動畫效果。
4. 多視角切換：具備第一人稱及第三人稱視角，結合遠距觀察與俯視等，支援四種鏡位視角模式。
5. 火焰粒子控制：使用粒子結構體FireworkParticle 紀錄位置、顏色、壽命週期及速度，進而反造出煙火的效果
6. 光影變化實作：以人造太陽作為光源，並計算投影矩陣，讓平面上能夠產生各種物件的光影變化。
7. 背景貼圖：將畫面中灰色背景改為星空影像背景，位避免色彩間的干擾，因此將使用GL\_REPLACE 模式進行處理。
8. 機器人控制：使用多層 Transform 進行組合，並採用分段繪製骨架刑事，透過關節間的旋轉與正弦波擺動的方式，實現行走、閒置以及頭部跟隨動畫進行改變的效果。

以下將針對各個程式設計技術進行程式進行深度描述：

1. OBJ 載入與貼圖系統

void DrawOBJModel(const OBJModel& model, GLuint textureID, GLint nShadow)

* model 是一個 OBJModel 結構，其中包含了模型的頂點、法線、貼圖座標、索引等資料。
* textureID 是 OpenGL 中的貼圖 ID。
* nShadow 是一個整數，用來判斷是否要畫出陰影版本（1 = 陰影，0 = 正常渲染）。

if (nShadow == 0){ … }else{ … }

* 若是陰影 (nShadow == 0)

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, textureID);

glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, fBrightLight);

* 若不是陰影 (nShadow != 0)

glDisable(GL\_TEXTURE\_2D);

glColor4f(0.0f, 0.0f, 0.0f, 0.6f);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, fNoLight);

* OBJ 物件頂點、法線與貼圖座標設定

if (!model.vertices.empty()) {

// 啟用頂點陣列

glEnableClientState(GL\_VERTEX\_ARRAY);

// 傳入頂點資料

glVertexPointer(3, GL\_FLOAT, 0, model.vertices.data());

}

if (!model.normals.empty()) {

glEnableClientState(GL\_NORMAL\_ARRAY); // 啟用法線陣列

glNormalPointer(GL\_FLOAT, 0, model.normals.data()); // 傳入法線資料

}

else {

glNormal3f(0.0f, 1.0f, 0.0f); // 若無法線資料，預設朝上

}

if (nShadow == 0 && !model.texCoords.empty()) {

glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY); // 啟用貼圖座標陣列（只有非陰影時用）

glTexCoordPointer(2, GL\_FLOAT, 0, model.texCoords.data()); // 傳入貼圖座標

}

* 繪圖階段

if (!model.indices.empty()) {

glDrawElements(GL\_TRIANGLES, model.indices.size(), GL\_UNSIGNED\_INT, model.indices.data());

}

else {

glDrawArrays(GL\_TRIANGLES, 0, model.vertices.size() / 3);

}

* + 如果有索引資料，就用 glDrawElements 繪製（效率較高）。
  + 否則用 glDrawArrays 直接根據頂點陣列繪圖。
* 關閉狀態

glDisableClientState(GL\_VERTEX\_ARRAY); // 關閉頂點陣列

if (!model.normals.empty()) {

glDisableClientState(GL\_NORMAL\_ARRAY); // 關閉法線陣列

}

if (nShadow == 0 && !model.texCoords.empty()) {

glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY); // 關閉貼圖座標陣列（僅正常渲染時關）

}

1. 動畫控制

* 餐桌(diningTable.obj)：靜態不參與任何動畫。
* 盆栽(pottedPlant.obj)：固定繞 Y 軸旋轉，在舞台中央旋轉。
* 草叢5叢(bush.obj)：繞盆栽旋轉。
* 撲克牌5張(playingCard.obj)：兩種模式。
* 一般模式：在草叢後跟著繞圈
* 展示模式：排成扇形、繞機器人一圈
* 機器人：操控驅動。

void DrawInhabitants(GLint nShadow) {

// 草叢與植物繞圈的速度倍率

static float orbitSpeed = 1.0f;

// Draw diningTable.obj 餐桌

glPushMatrix();

glTranslatef(0.0f, -0.4f, -2.5f);

glRotatef(270, 1.0f, 0.0f, 0.0f);

glScalef(0.3f, 0.3f, 0.3f);

DrawOBJModel(diningTable\_obj, textureObjects[DINING\_TABLE\_TEXTURE], nShadow);

glPopMatrix();

// Draw 5 Draw bush.obj 草叢 + 撲克牌

for (int i = 0; i < 5; ++i) {

glPushMatrix();

float amplitude = 5.0f;

float frequency = 0.5f;

float offsetZ = 2.0f;

float xOffset = amplitude \* cos(i \* frequency);

float zPosition = i \* 1.5f - offsetZ;

if (cardShow) { // 進入「撲克牌動畫」時的定位

glTranslatef(characterPos[0], characterPos[1] + 0.6f, characterPos[2]);

glRotatef(farYaw \* 180.0f / M\_PI, 0, 1, 0);

glTranslatef(10.0f + xOffset, -0.4f, zPosition);

glScalef(0.3f, 0.3f, 0.3f);

}

else { // 平時：草叢與舞台一起繞圈

glTranslatef(0.0f, -0.4f, -2.5f);

glRotatef(yRot \* orbitSpeed \* 2.0f, 0.0f, 1.0f, 0.0f);

glTranslatef(10.0f + xOffset, -0.4f, zPosition);

glScalef(0.3f, 0.3f, 0.3f);

DrawOBJModel(grass\_obj\_model, textureObjects[BUSH\_OBJ\_TEXTURE], nShadow);

}

glPopMatrix();

// Draw playing cards behind the bushes 撲克牌（位置依 i 改變）

glPushMatrix();

float cardOffsetFromBushZ = -0.5f; // 與草叢 Z 差

xOffset = amplitude \* cos(i \* frequency);

zPosition = i \* 1.5f - offsetZ;

float spacing = 2.5f; // 每張橫向間距

float y\_height = 1.25f; //牌面高度

float z\_bihind = 1.5f; // 前後層次差

if (cardShow) { // 撲克牌動畫定位

glTranslatef(characterPos[0], characterPos[1] + 0.6f, characterPos[2]);

glRotatef(farYaw \* 180.0f / M\_PI, 0, 1, 0);

glTranslatef((i - 2) \* spacing, y\_height, abs((i - 2)) \* z\_bihind - 2.5f);

glScalef(0.3f, 0.3f, 0.3f);

}

else { // 平時：藏在草叢後面

glTranslatef(0.0f, -0.4f, -2.5f);

glRotatef(yRot \* orbitSpeed \* 2.0f, 0.0f, 1.0f, 0.0f);

glTranslatef(5.0f + xOffset, 1.25f, zPosition - cardOffsetFromBushZ);

glScalef(0.3f, 0.3f, 0.3f);

glRotatef(90.0f, 0.0f, 1.0f, 0.0f);

}

// 奇偶分別使用方塊 & 紅心材質

if (i % 2 == 0) {

DrawOBJModel(playingCard\_obj, textureObjects[DIAMOND\_CARD\_TEXTURE], nShadow);

}

else {

DrawOBJModel(playingCard\_obj, textureObjects[HEART\_CARD\_TEXTURE], nShadow);

}

glPopMatrix();

}

// Draw pottedPlant.obj(盆栽)

glPushMatrix();

glTranslatef(0.0f, -0.2f, -2.5f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

glTranslatef(0.0f, 0.5f, 0.0f);

glScalef(0.02f, 0.02f, 0.02f);

DrawOBJModel(pottedPlant\_obj, textureObjects[POTTED\_PLANT\_TEXTURE], nShadow);

glPopMatrix();

// Draw Robot(機器人)

glPushMatrix();

if (nShadow == 1) {

glColor4f(0.0f, 0.0f, 0.0f, 0.6f);

glDisable(GL\_TEXTURE\_2D);

}

else {

glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

glEnable(GL\_TEXTURE\_2D);

}

glPushMatrix();

glTranslatef(characterPos[0], characterPos[1], characterPos[2]);

glRotatef(thirdPersonYaw \* 180.0f / M\_PI, 0.0f, 1.0f, 0.0f);

DrawRobot();

glPopMatrix();

if (nShadow == 0 && camMode == MODE\_FIRST) {

glColor4f(1.0f, 1.0f, 1.0f, 1.0f);

glEnable(GL\_TEXTURE\_2D);

glPushMatrix();

glTranslatef(0.0f, -1.5f, -2.0f);

DrawRobot();

glPopMatrix();

}

glPopMatrix();

}

1. 多視角切換
2. 本程式提供四種攝影機模式：

* MODE\_FIRST：玩家以角色第一人稱視角觀看。
* MODE\_THIRD：角色背後固定距離與高度的追蹤視角。
* MODE\_FAR：全景視角，可俯瞰整個場景。
* MODE\_TOP: 從角色上方垂直俯視。

1. 模式切換與更新函式:

* UpdateCameraView**()**：依照 camMode 設定攝影機模式（MODE\_FIRST、MODE\_THIRD、MODE\_FAR、MODE\_TOP）並計算對應的攝影機位置、朝向與上向量。

void UpdateCameraView() {

switch (camMode) {

case MODE\_FIRST: // 第一人稱：與角色位置和朝向一致

frameCamera.SetOrigin(characterPos);

frameCamera.SetForwardVector(firstForward);

frameCamera.SetUpVector(firstUp);

break;

case MODE\_THIRD: // 第三人稱：角色後方固定距離與高度

{

// 1. 計算角色前方向量：由 yaw, pitch 決定

savedForward[0] = sin(thirdPersonYaw) \* cos(thirdPersonPitch);

savedForward[1] = sin(thirdPersonPitch);

savedForward[2] = cos(thirdPersonYaw) \* cos(thirdPersonPitch);

m3dNormalizeVector(savedForward);

// 2. 設定世界上方向量

M3DVector3f worldUp = {0.0f, 1.0f, 0.0f};

// 3. 計算右向量

M3DVector3f right;

m3dCrossProduct(right, worldUp, savedForward);

m3dNormalizeVector(right);

// 4. 計算攝影機位置：角色位置 - forward \* 距離 + 高度偏移

M3DVector3f camPos = {

characterPos[0] - savedForward[0] \* thirdPersonDist,

characterPos[1] + thirdPersonHeight,

characterPos[2] - savedForward[2] \* thirdPersonDist

};

frameCamera.SetOrigin(camPos);

frameCamera.SetForwardVector(savedForward);

frameCamera.SetUpVector(worldUp);

}

break;

case MODE\_FAR: // 遠景：環繞場景中心

{

float cp = cosf(farPitch), sp = sinf(farPitch);

float sy = sinf(farYaw), cy = cosf(farYaw);

// 1. 計算球面坐標轉換得到偏移

M3DVector3f offset = {

farDist \* cp \* sy,

farDist \* sp,

farDist \* cp \* cy

};

//2. 設定攝影機位置 = 場景中心 + 偏移 + 高度

freeCamPos[0] = sceneCenter[0] + offset[0];

freeCamPos[1] = sceneCenter[1] + offset[1] + farHeight;

freeCamPos[2] = sceneCenter[2] + offset[2];

//3. 設定朝向場景中心

savedForward[0] = sceneCenter[0] - freeCamPos[0];

savedForward[1] = sceneCenter[1] - freeCamPos[1];

savedForward[2] = sceneCenter[2] - freeCamPos[2];

m3dNormalizeVector(savedForward);

frameCamera.SetOrigin(freeCamPos);

frameCamera.SetForwardVector(savedForward);

frameCamera.SetUpVector((M3DVector3f){0.0f, 1.0f, 0.0f});

}

break;

case MODE\_TOP: // 俯視：垂直向下俯瞰

{

freeCamPos[0] = sceneCenter[0];

freeCamPos[1] = sceneCenter[1] + topHeight;

freeCamPos[2] = sceneCenter[2];

//保存朝向與上向量

savedForward[0] = 0.0f;

savedForward[1] = -1.0f;

savedForward[2] = 0.0f;

savedUp[0] = 0.0f; savedUp[1] = 0.0f; savedUp[2] = -1.0f;

frameCamera.SetOrigin(freeCamPos);

frameCamera.SetForwardVector(savedForward);

frameCamera.SetUpVector(savedUp);

}

break;

}

// 通知重繪

glutPostRedisplay();

}

1. 視角切換按鍵：

* Keyboard**()** ：
  + C：切換到遠景模式（MODE\_FAR），獲得環繞全局的全景視角。
  + V：切換到俯視模式（MODE\_TOP），從上方直接俯瞰場景。
  + Q：在人稱視角（第一／第三）與特殊視角（遠景／俯視）之間切換，保留先前人稱模式的設定，方便快速切回。

void Keyboard(unsigned char key, int x, int y) {

switch(key) {

// ...

case 'c': // 切換到遠景模式

camMode = MODE\_FAR; // 設定遠景視角

farYaw = thirdPersonYaw; // 保留當前 yaw

farPitch = farPitchANGLE; // 使用預設俯視仰角

UpdateCameraView();

break;

case 'v': // 切換到俯視模式

camMode = MODE\_TOP; // 設定俯視視角

topYaw = 0; // 重置水平角

topPitch = -M\_PI/2; // 俯視仰角

UpdateCameraView();

break;

case 'q': // 切換第一/第三人稱模式

if (camMode == MODE\_FAR || camMode == MODE\_TOP)

{

// 若在遠景或俯視，則回到先前的人稱模式

camMode = isFirstPerson ? MODE\_FIRST : MODE\_THIRD;

} else {

// 在人稱模式間切換

isFirstPerson = !isFirstPerson;

camMode = isFirstPerson ? MODE\_FIRST : MODE\_THIRD;

}

// 更新攝影機方向與位置

if (camMode == MODE\_FIRST) {

frameCamera.SetOrigin(characterPos);

frameCamera.SetForwardVector(firstForward);

frameCamera.SetUpVector(firstUp);

} else {

frameCamera.GetForwardVector(savedForward);

thirdPersonYaw = atan2(savedForward[0], savedForward[2]);

thirdPersonPitch = asin(savedForward[1]);

}

UpdateCameraView();

break;

}

// ...

}

1. 旋轉輸入處理:

* SpecialKeys**()** 方向鍵：根據 pitchDelta 、 yawD 呼叫對應旋轉函式：
* ApplyFirstPersonRotation(yawD, pitchD)：處理鍵盤方向鍵在第一人稱模式下的 yaw/pitch 旋轉。
* ApplyThirdPersonRotation(yawD, pitchD)：第三人稱模式下，基於角度變化計算攝影機新朝向與位置。
* ApplyFarCameraRotation(yawD, pitchD)：遠景模式下旋轉攝影機繞場景中心轉動。
* ApplyTopCameraRotation(yawD, pitchD)：俯視模式下攝影機繞場景中心水平旋轉。

// 根據模式執行不同旋轉邏輯，yawD/pitchD 由方向鍵設定

void SpecialKeys(int key, int x, int y) {

float yawD = 0.0f, pitchD = 0.0f;

switch (key) {

case GLUT\_KEY\_UP: pitchD = +pitchDelta; break;

case GLUT\_KEY\_DOWN: pitchD = -pitchDelta; break;

case GLUT\_KEY\_LEFT: yawD = +pitchDelta; break;

case GLUT\_KEY\_RIGHT: yawD = -pitchDelta; break;

}

switch (camMode) { //根據模式執行相對應的旋轉函式

case MODE\_FIRST: ApplyFirstPersonRotation(yawD, pitchD); break;

case MODE\_THIRD: ApplyThirdPersonRotation(yawD, pitchD); break;

case MODE\_FAR: ApplyFarCameraRotation(yawD, pitchD); break;

case MODE\_TOP: ApplyTopCameraRotation(yawD, pitchD); break;

}

glutPostRedisplay();

}

1. 火焰粒子控制

// 在 Keyboard 函式中

void Keyboard(unsigned char key, int x, int y) {

// ...

switch (key) {

// ...

case '4':

// 1. 設置模式旗標和計時器

isFireworkMode = true;

fireworkTimer = 5.0f; // 設定煙火秀持續 5 秒

// 2. 切換攝影機到遠景視角

camMode = MODE\_FAR;

farYaw = thirdPersonYaw; // 繼承當前的水平視角方向

farPitch = farPitchANGLE; // 使用一個預設的俯瞰角度

UpdateCameraView(); // 立刻更新攝影機位置和朝向

break;

// ...

}

// ...

}

* 1. 進入煙火模式：
     + isFireworkMode = true;：這是一個全域布林（boolean）旗標，用來告訴程式的其他部分（如計時器、輸入處理）現在正處於煙火秀狀態。在此狀態下，大部分的玩家控制（如移動、滑鼠點擊）會被鎖定。
     + fireworkTimer = 5.0f;：設定一個 5 秒的倒數計時器。
  2. 切換攝影機視角：
     + camMode = MODE\_FAR;：將攝影機模式強制切換到「遠景模式」，以便能有更廣闊的視野來觀賞天空中的煙火。
     + UpdateCameraView();：立即呼叫此函式，根據新的 MODE\_FAR 設定來重新計算攝影機的位置和方向，讓使用者立刻看到視角的變化。

// 在 TimerFunction 函式中

void TimerFunction(int value) {

float dt = 0.033f;

UpdateParticles(dt); // 更新所有現存粒子的位置和生命

// ...

if (isFireworkMode) {

fireworkTimer -= dt; // 倒數計時

// 核心：每一幀都發射一次新的煙火

LaunchAutomaticFirework();

if (fireworkTimer <= 0.0f) {

ResetScene(); // 時間到，重置整個場景

}

}

// ...

glutPostRedisplay();

glutTimerFunc(33, TimerFunction, 1);

}

* 1. 倒數計時：
     + fireworkTimer -= dt; 在每一幀都減少計時器的時間。
  2. 持續生成煙火：
     + LaunchAutomaticFirework(); 這是煙火秀的核心。它在每一幀都被呼叫，意味著每隔約 33 毫秒，就會有一個新的煙火在天空的隨機位置爆炸。這創造了極其密集、充滿整個天空的壯觀效果。
  3. 結束與重置：
     + 當 fireworkTimer 倒數到 0 或更少時，呼叫 ResetScene()，將遊戲狀態、攝影機、角色位置等全部恢復到初始狀態，煙火秀結束。
  4. 粒子更新：
     + UpdateParticles(dt); 這個函式負責物理模擬。它遍歷所有粒子，根據它們的速度更新位置，並減少它們的生命值。生命值耗盡的粒子會被移除。

void LaunchAutomaticFirework() {

// 1. 在一個廣闊的空間內隨機選擇一個爆炸中心點

float worldX = ((float)rand() / RAND\_MAX) \* 80.0f - 40.0f;

float worldY = ((float)rand() / RAND\_MAX) \* 30.0f + 25.0f;

float worldZ = ((float)rand() / RAND\_MAX) \* 80.0f - 40.0f;

// 2. 在該中心點生成大量粒子，模擬一次爆炸

for (int i = 0; i < 350; ++i) { // 每次爆炸產生 350 個粒子

FireworkParticle p;

p.x = worldX; p.y = worldY; p.z = worldZ; // 初始位置都在爆炸中心

// 3. 給予每個粒子隨機的向外速度、顏色和生命週期

float angle1 = ((float)rand() / RAND\_MAX) \* 2 \* M\_PI; // 球面座標隨機角度

float angle2 = ((float)rand() / RAND\_MAX) \* M\_PI;

float speed = ((float)rand() / RAND\_MAX) \* 0.15f; // 隨機速度

p.vx = speed \* cos(angle1) \* sin(angle2); // 計算速度分量

p.vy = speed \* sin(angle1) \* sin(angle2);

p.vz = speed \* cos(angle2);

p.r = 0.5f + ((float)rand() / RAND\_MAX) \* 0.5f; // 隨機明亮顏色

p.g = 0.5f + ((float)rand() / RAND\_MAX) \* 0.5f;

p.b = 0.5f + ((float)rand() / RAND\_MAX) \* 0.5f;

p.life = 2.0f + ((float)rand() / RAND\_MAX) \* 2.0f; // 隨機生命時長

particles.push\_back(p); // 將新粒子加入到全域的粒子列表中

}

}

void DrawParticles() {

// 關閉光照和貼圖，因為粒子是自發光的點

glDisable(GL\_LIGHTING);

glEnable(GL\_BLEND); // 啟用混合，讓粒子有透明效果

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glDisable(GL\_TEXTURE\_2D);

glPointSize(2.5f); // 設定每個點的大小

glBegin(GL\_POINTS); // 開始繪製點

for (const auto& p : particles) {

// 根據粒子的剩餘生命值設定其透明度

glColor4f(p.r, p.g, p.b, p.life);

glVertex3f(p.x, p.y, p.z);

}

glEnd();

// 還原狀態

glDisable(GL\_BLEND);

glEnable(GL\_LIGHTING);

glEnable(GL\_TEXTURE\_2D);

}

* 1. 粒子繪製
     + glBegin(GL\_POINTS): 這是最適合繪製大量微小粒子的方式。
     + glEnable(GL\_BLEND): 啟用顏色混合，glColor4f 的第四個參數 (Alpha/透明度) 才會生效。p.life 被用作透明度，使得粒子在生命結束時會逐漸淡出，而不是突然消失。

1. 撲克牌展示動畫

// Card Show

bool cardShow = false;

float cardYawStart = 0.0f;

float cardRotTime = 0.0f;

const float cardRotDur = 4.0f;

const float cardRotSpeed = 2.0f \* M\_PI / cardRotDur;

// 在 Keyboard 函式中

void Keyboard(unsigned char key, int x, int y) {

// ...

switch (key) {

// ...

case '5':

// 若撲克牌動畫已開啟，直接離開避免重複觸發

if (cardShow) break;

// 切換到「遠景」攝影機模式

camMode = MODE\_FAR;

farYaw = thirdPersonYaw;

farPitch = farPitchANGLE;

UpdateCameraView();

// 設置模式旗標和計時器

cardShow = true;

cardYawStart = farYaw;

cardRotTime = 0.0f;

break;

// ...

}

// ...

}

* 1. 進入撲克牌表演模式：
     + cardShow = true;：這是一個全域布林（boolean）旗標，用來告訴程式的其他部分（如計時器、輸入處理）現在正處於撲克牌表演狀態。在此狀態下，大部分的玩家控制（如移動、滑鼠點擊）會被鎖定。
     + cardRotDur = 4.0f;：動畫總長度。
     + cardRotSpeed：設定一圈所需角度速度。
  2. 切換攝影機視角：
     + camMode = MODE\_FAR;：將攝影機模式強制切換到「遠景模式」，攝影機開始繞行。
     + UpdateCameraView();：立即呼叫此函式，根據新的 MODE\_FAR 設定來重新計算攝影機的位置和方向，讓使用者立刻看到視角的變化。

void TimerFunction(int value) {

// ...

if (cardShow) {

cardRotTime += dt;

farYaw = cardYawStart + cardRotSpeed \* cardRotTime;

UpdateCameraView();

if (cardRotTime >= cardRotDur) {

cardShow = false;

ResetScene(); // 動畫結束，場景重置

}

// ...

glutPostRedisplay();

glutTimerFunc(33, TimerFunction, 1);

}

}

* 1. 動畫時間累積：
     + cardRotTime += dt; ：每幀加上時間差 dt，用於計算動畫進度。
  2. 鏡頭旋轉：
     + farYaw = cardYawStart + cardRotSpeed \* cardRotTime;：攝影機以一定速度繞 Y 軸旋轉，達成環繞角色效果。
  3. 動畫結束判斷：
     + if (cardRotTime >= cardRotDur)：播放達到 4 秒後，自動結束動畫並重設場景。

1. 光源及陰影實作

void Keyboard(unsigned char key, int x, int y) {

// ...

switch (key) {

case '1': // 東方 (早晨)

fLightPos[0] = 100; fLightPos[1] = 25; fLightPos[2] = 0;

m3dMakePlanarShadowMatrix(mShadowMatrix, g\_pPlane, fLightPos);

break;

case '2': // 正上方 (正午)

fLightPos[0] = 0; fLightPos[1] = 100; fLightPos[2] = 0;

m3dMakePlanarShadowMatrix(mShadowMatrix, g\_pPlane, fLightPos);

break;

case '3': // 西方 (黃昏)

fLightPos[0] = -100; fLightPos[1] = 25; fLightPos[2] = 0;

m3dMakePlanarShadowMatrix(mShadowMatrix, g\_pPlane, fLightPos);

break;

// ...

}

glutPostRedisplay();

}

* 1. 改變光源位置實作：
     + 在程式中透過修改全域變數 fLightPos 的值改變光源 CL\_LIGHT0的位置。
     + '1' (東方): 將光源設在 (100, 25, 0)，即X軸正向遠方且高度較低，模擬日出。
     + '2' (正上方): 將光源設在 (0, 100, 0)，模擬太陽高掛。
     + '3' (西方): 將光源設在 (-100, 25, 0)，即X軸負向遠方且高度較低，模擬日落。
  2. 更新陰影變化實作：
     + 每當fLightPos 更新後，程式會立刻呼叫 m3dMakePlanarShadowMatrix(...)。
     + 這個函式會根據新的光源位置重新計算一個特殊的「陰影投影矩陣」mShadowMatrix。
     + 在 RenderScene 函式中，這個矩陣被用來將物體「壓平」到地面上，從而繪製出方向和長度都正確的陰影。

void DrawSun(void) {

// 1. 設定太陽的專屬渲染狀態 (讓它像發光體)

glDisable(GL\_LIGHTING); // 太陽自身不應該被光照影響，否則會有陰暗面

glColor3f(1.0f, 1.0f, 1.0f);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_D, textureObjects[SUN\_TEXTURE]);

// 關鍵：設定貼圖模式為 GL\_REPLACE

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

// 2. 移動到光源位置並繪製球體

glPushMatrix();

glTranslatef(fLightPos[0], fLightPos[1], fLightPos[2]); // 移動到與光源完全相同的位置

// 使用 GLU 工具來建立和繪製球體 (這段是優化，只在第一次呼叫時建立)

static GLUquadric\* pSun = NULL;

if (pSun == NULL) {

pSun = gluNewQuadric();

gluQuadricDrawStyle(pSun, GLU\_FILL);

gluQuadricNormals(pSun, GLU\_SMOOTH);

gluQuadricTexture(pSun, GL\_TRUE); // 必須啟用才能貼上紋理

}

// 修正貼圖座標，讓太陽貼圖看起來是正的

glRotatef(180.0f, 0.0f, 1.0f, 0.0f);

glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);

// 正式繪製球體：半徑5.0，精細度30x30

gluSphere(pSun, 5.0f, 30, 30);

glPopMatrix();

// 3. 還原渲染狀態，以免影響後續其他物件的繪製

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glEnable(GL\_LIGHTING);

}

* 1. 發光效果 (GL\_REPLACE)：
     + 正常情況下，OpenGL 的貼圖模式是 GL\_MODULATE，它會將貼圖顏色與物體的光照計算結果相乘。這會使物體有明暗變化。
     + 但對於太陽，我們不希望它有陰暗面。通過設定 glTexEnvi(..., GL\_REPLACE)，程式告訴 OpenGL：「忽略所有光照計算，直接用貼圖的原始顏色覆蓋到物體表面」。這就產生了太陽看起來像是在自己發光的視覺效果。
  2. 幾何體生成 (gluSphere)：
     + 程式使用了 GLU (OpenGL Utility Library) 中的 gluSphere 函式來快速生成一個球體。
     + static GLUquadric\* pSun 是一個聰明的優化：GLUquadric 物件（用來描述如何繪製球體）只需要被創建一次。static 關鍵字讓它在函式結束後不會被銷毀，下次再呼叫 DrawSun 時可以直接使用，避免了重複創建的開銷。
     + gluQuadricTexture(pSun, GL\_TRUE) 這行至關重要，它命令 GLU 為生成的球體自動產生紋理座標。
  3. 定位 (glTranslatef):
     + glTranslatef 使用了與光源完全相同的 fLightPos 座標。這確保了我們眼睛看到的太陽球體，其位置和場景中那個看不見的、真正產生光照效果的光源是完全一致的。
  4. 狀態還原:
     + 繪製完太陽後，程式立刻將光照和貼圖模式還原。這是一個非常好的習慣，確保了接下來要繪製的機器人、地面等其他物體能被正常的光照和貼圖模式所渲染。

1. 機器人控制
2. DrawRobot()：

使用即時矩陣堆疊（glPushMatrix/glPopMatrix）與基本原語（cube、sphere）組成機器人模型，配合可調式關節角度實現扭動與行走。

* + - torsoRot：控制軀幹旋角度。
    - 繪製頭部時，使用 headYaw、headPitch 進行雙軸旋轉。
    - 上臂和肘部運動分別由 rightArmRot、rightElbowRot（或左臂對應變數）驅動。
    - 雙腿關節角度 leftLegRot、rightLegRot 用於行走姿態。

void DrawRobot() {

glPushMatrix();

// 1. 軀幹旋轉：根據動畫參數 torsoRot

glRotatef(torsoRot, 0.0f, 1.0f, 0.0f);

// 2. 繪製軀幹

glPushMatrix();

glScalef(1.0f, 1.5f, 0.5f);

glutSolidCube(1.0f);

glPopMatrix();

// 3. 繪製頭部

glPushMatrix();

glTranslatef(0.0f, 0.75f+0.2f, 0.0f);

glRotatef(headYaw \* RAD\_TO\_DEG, 0,1,0); // 水平轉動

glRotatef(headPitch \* RAD\_TO\_DEG, 1,0,0);// 垂直轉動

glutSolidCube(0.4f);

glPopMatrix();

// 4. 繪製手臂與關節（以右臂為例，左臂類似）

glPushMatrix();

glTranslatef(-0.6f, 0.75f, 0.0f);

glRotatef(rightArmRot, 1,0,0); // 上臂擺動

glutSolidSphere(0.15f, 12, 12);

// 繪製下臂

glTranslatef(0.0f, -0.5f, 0.0f);

glRotatef(rightElbowRot, 1,0,0);

glScalef(0.2f, 0.8f, 0.2f);

glutSolidCube(1.0f);

glPopMatrix();

// 5. 雙腿（類似手臂流程）

for (int i = 0; i < 2; ++i) {

float offsetX = (i==0) ? -0.3f : 0.3f;

float legRot = (i==0) ? leftLegRot : rightLegRot;

glPushMatrix();

glTranslatef(offsetX, -0.75f, 0.0f);

glRotatef(legRot, 1,0,0);

glScalef(0.3f, 1.0f, 0.3f);

glutSolidCube(1.0f);

glPopMatrix();

}

glPopMatrix();

}

1. 行走動畫:
   * + movefunction**(key)**：

* 根據 key（W/A/S/D）更新角色位置 characterPos，並以正交向量 forward、right 計算移動方向。
* 以 animTime × swingSpeed 作為相位計算手臂與腿部擺動角度（sin 加權）。

void movefunction(unsigned char key) {

const float step = 0.3f;

// 根據鍵值更新位置

if (key=='w' || key=='W') characterPos += forward \* step;

else if (key=='s'|| key=='S') characterPos -= forward \* step;

else if (key=='a'|| key=='A') characterPos -= right \* step;

else if (key=='d'|| key=='D') characterPos += right \* step;

// 擺臂擺腿：sin函式驅動擺幅

float phase = animTime \* swingSpeed;

rightArmRot = 30.0f \* sinf(phase);

leftArmRot = -30.0f \* sinf(phase);

rightLegRot = -30.0f \* sinf(phase);

leftLegRot = 30.0f \* sinf(phase);

}

1. 動畫定時更新:
   * + TimerFunction：

* dt 固定為 0.033s，累加 animTime 追蹤動畫進度。
* 當 moving == true 時，呼叫 movefunction(lastKey) 同步位置與關節動作。
* 以 glutTimerFunc 重複調度達到持續動畫更新。

void TimerFunction(int value) {

// ...

float dt = 0.033f;

animTime += dt;

if (moving) movefunction(lastKey);

glutPostRedisplay();

glutTimerFunc(33, TimerFunction, 1);

}

# **三、設計說明**

1. 物件動畫說明

|  |  |  |
| --- | --- | --- |
| **物件** | **動畫** | **說明** |
| Object A | 植物盆栽 | 自轉動畫，使用 yRot 控制角度 |
| Object B | 草叢裝置 | 繞行 Object A 並呈現 x 軸正弦波動 |
| Object C | 撲克牌 | 依視角與模式擺動呈現旋轉卡片 |
| Object D | 機器人 | 腳步擺動、關節旋轉、上半身扭動、頭部 Pitch / Yaw |
| Object E | 煙火 | 每顆粒子獨立運動、顏色與壽命控制、爆炸時空間均勻分佈 |

1. 陰影實作說明

* 使用自訂平面陰影矩陣 m3dMakePlanarShadowMatrix。
* 計算平面方程式後投影至地面，搭配 GL\_STENCIL 實現遮罩陰影。
* 陰影部分關閉光照與貼圖並改為半透明黑色繪製。

# **四、技術實作難點與解法**

在本專題開發過程中，我們面臨多項技術挑戰，並針對每個問題提出具體且有效的解決方案。首先，OBJ 模型檔案中複雜的面資料結構，包含頂點（v）、貼圖座標（vt）與法線（vn）組合，且同時存在四邊形面（Quad）時，我們設計了 parseFacePart 函式，拆解並解析這些複合索引，進而建構唯一的頂點映射表，並將四邊形面拆分為三角形以符合 OpenGL 渲染標準，確保模型能正確且完整地呈現。

粒子動畫的效能也是一大挑戰。為避免過多粒子導致渲染瓶頸，我們對粒子數量設置上限，統一生命週期的管理，並優化粒子更新與繪製流程，成功提升了渲染效率，確保煙火效果在高密度狀態下依然保持流暢。

在陰影投影部分，為了達到動態光源與多視角下陰影的準確投射，我們將陰影投影矩陣與當前攝影機的變換矩陣相乘，並利用 OpenGL 的 GL\_STENCIL 模板測試避免陰影重疊帶來的視覺錯誤，進一步增強場景光影的真實感與穩定性。

此外，我們發現部分 OBJ 模型因頂點順序或貼圖座標錯誤而出現不顯示或變形問題，透過修正頂點定義及自訂正規化函式避免除以零錯誤，解決了這些幾何與紋理的異常，確保模型在場景中正常呈現。

針對背景與物件混色導致視覺不清的問題，我們調整了 OpenGL 的貼圖環境設定，將背景貼圖獨立採用 GL\_REPLACE 模式，使背景色彩不受物件光照與貼圖影響，成功維持星空背景的純淨視覺效果。

最後，考量到場景需要多種不同視角，我們使用枚舉型別 CameraMode 統一管理第一人稱、第三人稱、遠景與俯視四種攝影機模式，並根據視角動態調整 pitch、yaw 與位置參數，使視角切換流暢且同步，提升用戶的沉浸式體驗。

整體來說，這些技術難點的克服，不僅提升了專題的穩定性與效能，也為未來更複雜的3D互動系統開發奠定了堅實的技術基礎。

# **五、按鍵操作說明**

為提升使用者與系統的互動體驗，本專題設計了一套直觀且多樣的按鍵操作機制，讓使用者能靈活掌控場景中的視角切換、動畫觸發及角色移動等功能。透過明確的按鍵映射，不僅降低學習門檻，也確保了操作的即時性與反饋效果。以下將逐一說明各功能對應的按鍵設定與使用方式，期望藉此打造流暢且具沉浸感的使用者互動流程。

|  |  |
| --- | --- |
| **按鍵** | **功能說明** |
| 1~3 | 控制太陽位置（早上、正午、傍晚）。 |
| 4 | 觸發煙火秀模式（持續 5 秒）。 |
| 5 | 撲克牌展示旋轉模式。 |
| WASD | 操控機器人前 / 左 / 後 / 右移動。 |
| ↑↓← → | 控制視角 pitch / yaw。 |
| Q | 在第一人稱 ↔ 第三人稱間切換。 |
| C | 切換至遠距環繞視角。 |
| V | 切換至垂直俯視視角。 |
| P | 暫停/繼續物件旋轉與角色動畫。 |
| R | 重設場景。 |
| 左鍵按下 | 於點擊處（以射線投影至空中 y = 5）觸發單次煙火。 |

# **六、專案結構**

為確保專案開發流程的高效與維護便利，本專題採用清晰且模組化的目錄結構設計。各類資源如模型檔（OBJ）、材質貼圖（TGA）、核心程式碼與設定檔分別歸類，便於團隊協作與後續擴充。以下將詳細介紹專案的目錄配置與主要檔案功能，協助使用者快速掌握系統架構與內容分布。

1. 專案目錄結構

/FinalProject/

├── /TGA/

│ ├── blackandwhiteTiles.tga

│ ├── bush.tga

│ ├── diamond.tga

│ ├── heart.tga

│ ├── night.tga

│ ├── pottedPlant.tga

│ └── wood.tga

├── /OBJ/

│ ├── bush.obj

│ ├── diningTable.obj

│ ├── playingCard.obj

│ └── pottedPlant.obj

├── CMakeLists.txt

├── FinalProject.cpp

└── README.md

1. 專案 TGA 目錄結構

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| blackandwhiteTiles.tga | bush.tga | diamond.tga | heart.tga |
|  |  |  |  |
| moon.tga | night.tga | pottedPlant.tga |

1. 專案 OBJ 目錄結構

|  |  |
| --- | --- |
|  |  |
| bush.ogj | playingCard.obj |
|  |  |
| pottedPlant.obj | diningTable.obj |

# **七、成果展示**

本章節將展示專題最終實現的視覺效果與互動功能，透過多角度截圖與動態影片，具體呈現系統在場景渲染、動畫控制、粒子特效及多視角切換上的成果。這些展示不僅驗證了各項技術的整合成效，更突顯了「星空煙火舞台」的魅力，由於功能眾多，相關細節請查看YouTube影片：<https://youtu.be/DYsL7GeFb-4>

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 初始化 | 撲克牌與樹叢移動 | 抬頭 | 低頭 |
|  |  |  |  |
| 左轉 | 右轉 | 煙火 | 太陽 |
|  |  |  |  |
| 日出 | 正午 | 日落 | 滿空煙火 |
|  |  |  |  |
| 環視 | 第一人稱 | 第三人稱 | 桌面及花瓶特寫 |
|  |  |  |  |
| 俯視 | 俯視移動 | 俯視角度變化 |

# **八、總結**

本專題以「星空煙火舞台」為核心主題，成功整合了多種電腦圖學技術，包含多視角攝影機控制、OBJ 模型動態載入與貼圖、粒子系統實現煙火效果，以及基於矩陣變換的物件動畫控制。藉由貼圖與光影投影的細膩搭配，呈現出一個具備魔幻氛圍的沉浸式3D場景，並融合多元互動模式如滑鼠觸發煙火、鍵盤啟動高空煙火秀、撲克牌動畫展示以及機器人角色的多視角控制。

其中，物件動畫與粒子系統在效能與視覺效果之間取得良好平衡，煙火粒子結構設計與自動發射機制達成高密度爆炸的動態表現。多視角攝影機系統不僅提供第一人稱、第三人稱，還包含遠景與俯視視角，使得使用者能夠從多元角度觀賞舞台全貌。光影與陰影透過陰影投影矩陣與貼圖模式細節調整，成功營造出時間變換（日出、正午、日落）的自然光影變化。

此專題不僅展示了OpenGL在3D互動場景建構上的技術實力，更凸顯團隊在系統整合、動畫控制與用戶體驗上的深度思考，為未來複雜場景的動態渲染與互動設計奠定堅實基礎。

# **九、心得**

透過本專題的實作，我們深刻體會到電腦圖學不僅是單純的視覺呈現，更是交織物理演算、數學矩陣與使用者互動的多重挑戰。尤其在粒子系統設計與多視角切換上，需要同時考量效能優化與視覺流暢度，這讓我們在細節調校與架構規劃中獲得寶貴經驗。

此外，專案中對於光影變化與陰影矩陣的應用，讓我們更深刻理解了光線模擬在增強場景真實感上的關鍵作用。面對複雜的三維物件貼圖與動畫切換，團隊的協作與模組化設計策略確保了程式的可維護性與擴展性。

展望未來，我們期望將此系統進一步提升為支持更高複雜度粒子物理、光線追蹤陰影，以及更智慧的用戶互動介面，例如語音控制或動作偵測介面，帶來更加沉浸式與智能化的體驗。唯有持續跳脫傳統圖形框架，勇於嘗試新技術，才能引領電腦圖學邁向真正的次世代視覺革命。

在此，也特別感謝團隊成員的全力付出與默契協作，讓這個「星空煙火舞台」能夠完整且生動地呈現在我們眼前，彷彿將虛擬與現實的邊界再次拉近。