

---

# Computer Graphics\_HW4

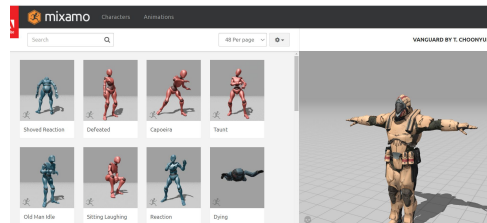
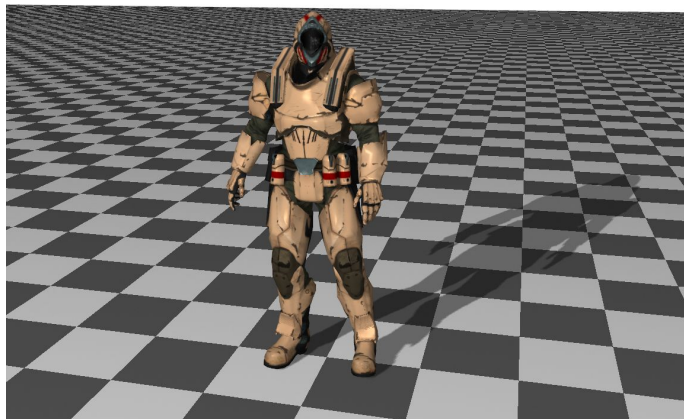
NE6121084 傅信豪

---

# 一、作業環境

1. windows 10
2. visual studio 2022
3. glfw 3.4
4. glut
5. opengl
6. assimp
7. model src : [mixamo](https://www.mixamo.com/)

assimp	2024/12/14 下午 02:29	檔案資料夾
glad	2024/12/14 上午 11:46	檔案資料夾
GLFW	2024/12/14 上午 11:46	檔案資料夾
glm	2024/12/14 上午 11:46	檔案資料夾
KHR	2024/12/14 上午 11:46	檔案資料夾



## 二、程式碼 (main.cpp)

- 參數定義
  - 取得根目錄
  - 設置相機初始距離角度
  - 滑鼠相關參數
  - 初始化animator
- 載入模型、貼圖、骨架動畫
  - 載入帶skin的.dae + 骨架動畫
- 鍵盤輸入

```
// 設定速度 = 4.0f * deltaTime; // 根據時間計算移動速度
bool Idle = true; // 判斷是否處於Idle狀態

// 設定初始位置
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
    character.position.x += 1.0f * speed;
    camera.x += 1.0f * speed;
    animator.playAnimation(character[1]); // 播放Idle動畫
    Idle = true;
}

else if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
    character.position.x -= 1.0f * speed;
    camera.x -= 1.0f * speed;
    animator.playAnimation(character[2]); // 播放Idle動畫
    Idle = true;
}

else if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) {
    character.position.x += 1.0f * speed;
    camera.x += 1.0f * speed;
    animator.playAnimation(character[3]); // 播放Idle動畫
    Idle = true;
}

else if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) {
    character.position.x -= 1.0f * speed;
    camera.x -= 1.0f * speed;
    animator.playAnimation(character[4]); // 播放Idle動畫
    Idle = true;
}
```

```
namespace fs = std::filesystem;

// 回調函數的宣告
void framebuffer_size_callback(GLFWwindow* window, int width, int height); // 處理視窗大小調整
void processInput(GLFWwindow* window, Animation* animations); // 處理鍵盤與滑鼠輸入
void mouse_callback(GLFWwindow* window, double xpos, double ypos); // 處理滑鼠移動
void renderNode(Node* node); // 渲染場景節點
void updateNodeTransformations(Node* node, glm::mat4 transformationThusFar); // 更新節點變換矩陣
void setUniformBoneTransforms(std::vector<glm::mat4> transforms, unsigned int shaderId); // 設定骨骼變換到着色器
// 取得根目錄
string getRootPath();
// 全域變數的定義
bool VSYNC = true; // 是否啟用垂直同步
bool FULLSCREEN = false; // 是否啟用全屏模式
int WINDOW_WIDTH = 1920; // 視窗寬度
int WINDOW_HEIGHT = 1080; // 視窗高度
int FPS = 999999; // 最大幀率限制

// 攝影機相關參數
glm::vec3 cameraPos = glm::vec3(2.0f, 2.0f, 5.0f); // 攝影機初始位置
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f); // 攝影機前方向
glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f); // 攝影機上方向
bool canAdjustCameraPos = true;

// 滑鼠相關參數
bool firstMouse = true; // 第一次檢測滑鼠移動
float yaw = -90.0f; // 攝影機的 Yaw 角度
float pitch = 0.0f; // 攝影機的 Pitch 角度
float lastX = 800.0f / 2.0f; // 滑鼠的初始 X 座標
float lastY = 800.0f / 2.0f; // 滑鼠的初始 Y 座標
float fov = 45.0f; // 攝影機視野範圍 (Field of View)

// 時間相關參數
float deltaTime = 0.0f; // 每幀的時間差 (用於移動計算)

// 動畫管理器
Animator animator = Animator(); // 用於控制動畫播放的物件

// 場景節點
Node* checkerFloor = createSceneNode(); // 地板的場景節點
Node* character = createSceneNode(); // 角色的場景節點
```

```
// 加載模型與動畫資源
std::string daeFile = projectRoot + "resource/vanguard/vanguard.dae";

std::string animFile1 = projectRoot + "resource/vanguard/Offensive_Idle.dae";
std::string animFile2 = projectRoot + "resource/vanguard/Running.dae";
std::string animFile3 = projectRoot + "resource/vanguard/Left_Strafe.dae";
std::string animFile4 = projectRoot + "resource/vanguard/Right_Strafe.dae";
std::string animFile5 = projectRoot + "resource/vanguard/Running_Backward.dae";
std::string animFile6 = projectRoot + "resource/vanguard/Jump.dae";
std::string animFile7 = projectRoot + "resource/vanguard/HipHopDancing.dae";
std::string animFile8 = projectRoot + "resource/vanguard/Wave_Hip_Hop_Dance.dae";

std::vector<TextureOverride> overrides = {
    {0, DIFFUSE, projectRoot + "resource/textures/vanguard_diffuse1.png"},
    {0, NORMAL, projectRoot + "resource/textures/vanguard_normal.png"},
    {0, SPECULAR, projectRoot + "resource/textures/vanguard_specular.png"},
};

// 加載模型
Model m = Model(daeFile, overrides);
vector<Mesh> squareMeshes = m.meshes;
std::cout << "loaded meshes: " << m.meshes.size() << std::endl;
```

```
// 加載動畫
Animation anim1(animFile1, &m);
Animation anim2(animFile2, &m);
Animation anim3(animFile3, &m);
Animation anim4(animFile4, &m);
Animation anim5(animFile5, &m);
Animation anim6(animFile6, &m);
Animation anim7(animFile7, &m);
Animation anim8(animFile8, &m);

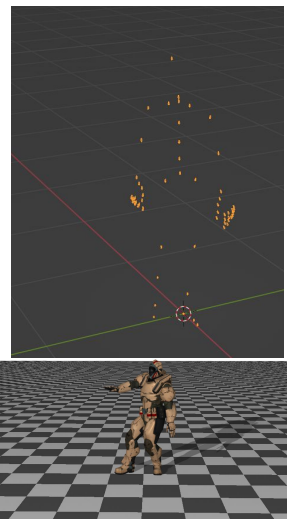
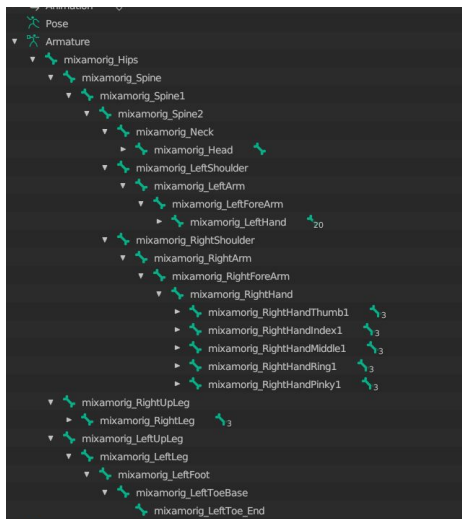
Animation animations[] = { anim1, anim2, anim3, anim4, anim5, anim6, anim7, anim8 };
//Animation animations[] = { anim1 };
```

## 二、程式碼 (animation.hpp)

- 讀取模型和動畫檔案
  - 使用 Assimp 庫載入指定的動畫檔案。
  - 檢查是否有動畫資料存在，沒有則直接跳出。
- 儲存動畫資訊
  - 取得動畫的持續時間和每秒幀數(Ticks Per Second, tps)。
- 生成骨骼樹結構
  - 利用 `generateBoneTree` 方法，遞迴建立骨骼的層次結構(從根節點到子節點)。
- 處理骨骼動畫數據
- 查找骨骼

## 二、程式碼 (model.hpp)

- 原始碼在載入模型時容易在查找骨骼拋出異常
  - 檢視blender模型.dae跟骨架動畫對應名稱皆正常(o)
  - 但骨骼綁定似乎有次序問題!(x)
  - 修改extractBoneWeightForVertices()後正常載入(o)



```
void extractBoneWeightForVertices(vector<glm::ivec4>& boneIds_all, vector<glm::ivec4>& weights_all, aiMesh* mesh, const aiScene* scene)
{
    std::map<std::string, int> boneNameToId; // 快速查找骨骼名稱到索引的映射

    // 初始化 boneNameToId
    for (unsigned int i = 0; i < boneProps.size(); ++i) {
        boneNameToId[boneProps[i].name] = i;
    }

    //std::cout << "Bone names in the model: " << std::endl;
    for (unsigned int i = 0; i < mesh->mNumBones; ++i) {
        std::string boneName = mesh->mBones[i]->mName.C_Str();
        //std::cout << "Bone " << i << ": " << boneName << std::endl;

        int boneID = -1;

        // 使用骨骼名稱查找索引
        if (boneNameToId.find(boneName) != boneNameToId.end()) {
            boneID = boneNameToId[boneName];
        }
        else {
            // 如果名稱不存在，添加新骨骼並更新映射
            boneProps.push_back({ boneName, aiMatrix4x4ToIdm(mesh->mBones[i]->mOffsetMatrix) });
            boneID = boneProps.size() - 1;
            boneNameToId[boneName] = boneID; // 更新映射
            boneCounter++;
        }

        // 驗證 boneID
        if (boneID == -1) {
            std::cerr << "Warning: Bone " << boneName << " not found or invalid!" << std::endl;
            continue;
        }

        // 處理頂點權重
        aiVertexWeight* weights = mesh->mBones[i]->mWeights;
        unsigned int numWeights = mesh->mBones[i]->mNumWeights;

        for (unsigned int weightIndex = 0; weightIndex < numWeights; ++weightIndex) {
            unsigned int vertexId = weights[weightIndex].mVertexId;
            float weight = weights[weightIndex].mWeight;

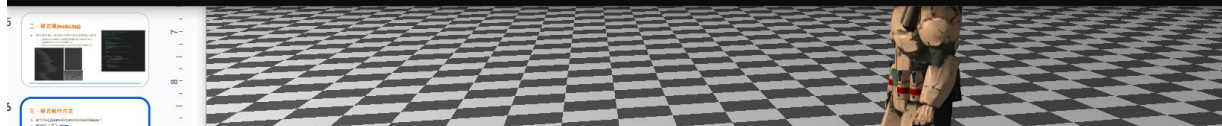
            if (vertexId >= boneIds_all.size()) {
                std::cerr << "Error: Vertex ID " << vertexId << " out of range!" << std::endl;
                continue;
            }

            // 更新頂點骨骼權重
            for (int j = 0; j < 4; ++j) {
                if (boneIds_all[vertexId][j] < 0) { // 空槽位
                    boneIds_all[vertexId][j] = boneID;
                    weights_all[vertexId][j] = weight;
                    break;
                }
            }
        }
    }
}
```

# 三、程式執行方式

1. cd 到hw4\_SkeletonAnimation\hw4\x64\Release下
2. terminal下指令 `./hw4.exe`

```
(base) PS D:\CSIE_COURSE\Computer-Graphic\hw4_SkeletonAnimation\hw4\x64\Release> hw4.exe
Root Directory: D:/CSIE_COURSE/Computer-Graphic/hw4_SkeletonAnimation/
*****
*
*   GL Vendor      : NVIDIA Corporation
*   GL Renderer   : NVIDIA GeForce RTX 2060/PCIe/SSE2
*   GL Version Name : 4.6.0 NVIDIA 560.94
*   GL Version    : 4.6
*   GLSL Version  : 4.60 NVIDIA
*
*****
Loaded custom texture: textures/vanguard_diffuse1.png
Loaded custom texture: textures/vanguard_normal.png
Loaded custom texture: textures/vanguard_specular.png
Processed 2 bones, triangle count: 570
Loaded texture: textures/vanguard_diffuse1.png
Loaded texture: textures/vanguard_diffuse1.png
Loaded texture: textures/vanguard_specular.png
Loaded texture: textures/vanguard_specular.png
Processed 52 bones, triangle count: 33558
Loaded meshes: 2
Starting..
FPS: 119.455
```



## 四、程式流程

1. 關閉視窗esc
2. 啟用滑鼠調整視角
  - 2.1. alt : 啟用
  - 2.2. 左鍵 : 關閉
3. WASD 執行前後左右動畫
4. Space : 跳躍
5. Z : Hiphop1 動畫
6. X : Hiphop2 動畫
- 7.

