# 骨骼绑定文档

余齐齐 2018013361

# 运行环境

## 项目环境与依赖

visual studio 2019 + OpenGL3.3

依赖的库：glad, glfw3, assimp

依赖库可以使用vcpkg安装：vcpkg设置 `VCPKG_DEFAULT_TRIPLET=x64-windows` 之后再 `vcpkg install glad,glfw3,assimp` 即可。

**注意**：在vs运行项目时报错 `Could not copy the file “targetBinaryFilePath” because it was not found.` 后来发现是vcpkg的问题，在git上已经有人提出issue，vcpkg已在2021/11/16 fixbug，vcpkg upgrade即可解决问题。

## 程序运行方式

1. visual studio编译项目，采用release x64运行。
2. `x64/Release` 文件夹下点击 `skinningOpengl.exe` 即可直接运行。

# 项目结构

整个项目的逻辑是参考learnopenGL的skeletal Animation教程，通过Assimp读取.dae模型中的信息，实现骨骼动画，在此基础上，重新实现了weight权重的计算算法。

## 文件结构

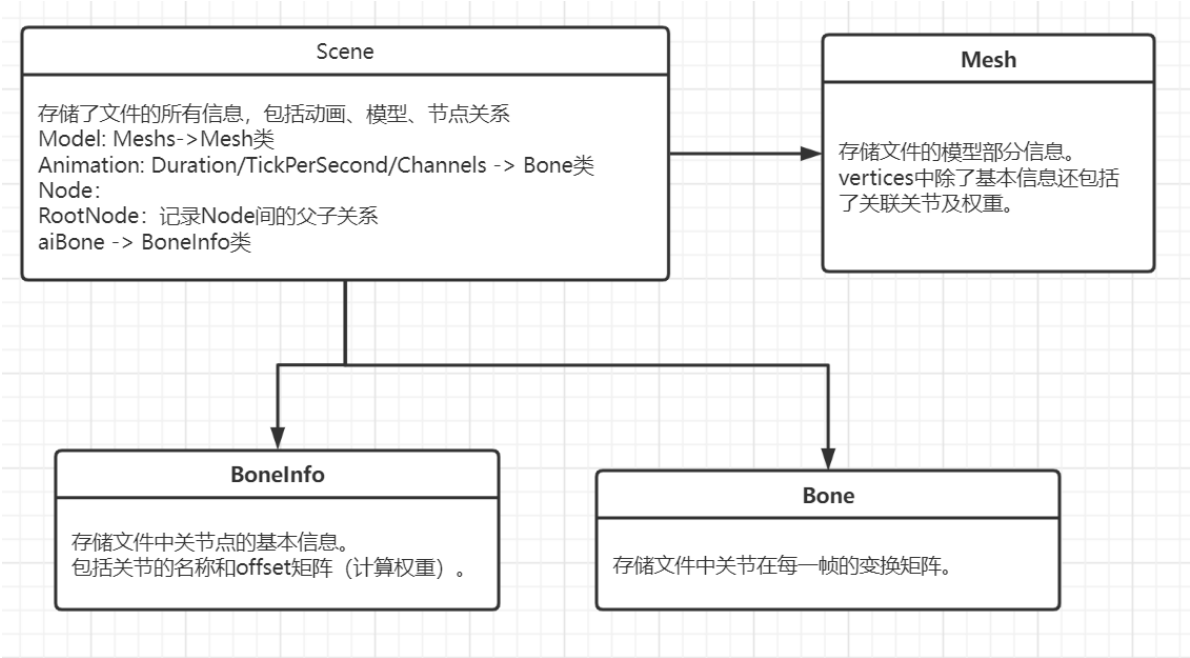
```
SkinningOpenGL
|_ main.cpp // 程序入口文件，创建openGL窗口，准备加载模型
|_ Scene.h + Scene.cpp // 场景类，类比Assimp::aiScene，包含了mesh/node/animation等
    |_ Mesh.h + Mesh.cpp // 模型类，类比Assimp::aiMesh
    |_ BoneInfo.h + BoneInfo.cpp // 关节基本信息类，存储关节名称和offset矩阵，类比Assimp::aiBone
    |_ Bone.h + Bone.cpp // 关节变换类，存储关节在每一帧的变换，类比Assimp::aiNodeAnim
    |_ utils.h + utils.cpp // 辅助类，include头文件、实现AssImp矩阵到GLM的矩阵的变换等。
    |_ shader.h // shader类
```

## 程序模块逻辑关系

`main.cpp` 作为程序的入口文件，创建了openGL窗口，链接了shader，准备开始读取.dae模型。

`scene`类为场景类，首先将scene文件中的模型读取到项目中，并存储到`Mesh`类；之后读取`Assimp::aiBone`的信息，将关节的名称和offset矩阵存储到`BoneInfo`类，并通过读取到的bone的offset矩阵与Mesh顶点的位置的求关节与顶点的欧式距离，计算权重；继续读取`Assimp::aiNodeAnim`的信息，将关节在每一帧的变换存储到`bone`类中；

在之后的glfw循环中，每次调用`scene.update`矩阵，通过获取到的关节在每一帧的变换，求出顶点在关节影响下的总变化，并绘制在屏幕上。




# 程序运行流程

1. 创建openGL窗口，准备shader和模型
2. 读取Assimp模型

   `loadModel`：根据Assimp模型的结构，先逐一处理Node，再逐一处理Node下的Mesh。

   `ProcessMesh:`

   1. 读取Mesh中的顶点信息。
   2. **读取Bone的基本信息，并计算顶点与骨骼权重信息**
3. 读取动画信息
   1. 读取Assimp中的Animation信息，将关节在每一帧的运动存储到`Bone`中，获取了所有关节的运动信息存储到了`vector<Bone> bones`数组中。
4. 读取关节之间的父子关系，为了便于之后的关节总变换计算。
5. 初始化openGL窗口，得到当前时间，调用`scene.update`获得当前时间的关节运动信息
6. `scene.update`
   1. 获取当前时间，**计算关节的变换矩阵**
7. 顶点随着关节运动而运动，调用shader，绘制动画。

## 关键步骤

- **计算顶点和骨骼的权重信息**
  - 获得骨骼和顶点的欧式距离

    首先要获得骨骼和顶点的欧式距离，注意到`Assimp`模型读取了bone的`offsetMatrix`，又称为`inverse-bind matrix`，这个矩阵是将vertex从local space变换到bone space的矩阵。

因此按下方公式就可以计算出vertex与骨骼的欧式距离的平方。

$$dis = dot(offset * vertex.Position, offset * vertex.Position)$$

采用了两种方式来决定权重：

1. 对每个模型表面点找一段距离最近的骨骼，认为最近骨骼对于该模型表面点的权重为1，其余骨骼对于该模型表面点的权重为0。
2. 对每个模型表面点取相邻的两段骨骼，简单地根据欧式距离用双线性插值赋予权重。

两种方式的实现效果相差不大，由于.dae模型中采用的是第一种，最后选择了1/0的权重选择方案。

- **计算关节的变换矩阵**
  - 计算关节本身的运动

    .dae模型中记录了关节在每个timestamp的变换矩阵，因此计算关节在某一时刻的运动是一个插值过程，对关节的平移、旋转、放缩进行插值即可。插值方案：

    $$t = (curTime - lastTime)/(nextTime - lastTime)$$

    $$curTransform = (1 - t) * lastTransform + t * nextTransform$$
  - 计算关节随父关节的运动，父节点的运动会一直往下传递，直到叶子节点。
- **计算顶点随关节的运动（shader）**
  - 从顶点到关节的空间，需要先左乘offset矩阵，变换到关节空间。
  - 根据权重计算最后的运动位置

    $$P_{out} = \sum_{1}^{maxBoneEffect} weight_i * B_i * P_{in}$$

# 程序演示方法

点击release.exe既可以看到生成的骨骼动画。

# 参考代码

- 实现骨骼动画部分参考了learnOpenGL的Skeletal-Animation模块:https://learnopengl.com/Guest-Articles/2020/Skeletal-Animation
- 加载模型部分参考了learnOpenGL的Model模块：https://learnopengl-cn.github.io/03%20Model%20Loading/03%20Model/
- Shader类参考了learnOpenGL的着色器模块：https://learnopengl-cn.github.io/01%20Getting%20started/05%20Shaders/
- Mesh类参考了learnOpenGL的网格模块： https://learnopengl-cn.github.io/03%20Model%20Loading/02%20Mesh/
- 使用的.dae模型来自mixamo：https://www.mixamo.com/#/?page=1&query=jump&type=Motion%2CMotionPack