README

作业 2

1. 实现功能

本次作业涉及两个部分：第一个部分需要完成显示骨骼并完成转换；第二部分需要完成 SSD。

1.1 第一部分

功能全部实现。

1. 矩阵栈实现。
2. 读入.skel 文件，解析装入。
3. 绘制关节。
4. 绘制骨骼。
5. 实现旋转骨骼接口。

1.2 第二部分

功能全部实现。

1. 文件输入：绑定姿态网格。读入并解析文件中的点面数据。
2. 网格渲染。实现 Mesh.cpp 中的 draw，用保存的 currentVertices 中的数据进行网格渲染。
3. 文件输入：附加权重。读入.attach 中的权重信息，解析并保存。
4. 世界坐标系与局部坐标系互转。
5. 对于模型进行变换，及时进行权重计算并更新。

2. 实现过程

2.1 第一部分

1.矩阵栈

- 栈具有两个基本功能，push(压栈)，pop(出栈)。在本作业中还需要实现 clear(清空矩阵栈)，top(返回栈顶元素)。为了模拟栈的操作，使用 vector 容器可以很容易的实现上述功能，因为 vector 自带的 push_back,pop,back，与栈的机制类似。
- 初始化中，根据注解“Initialize the matrix stack with the identity matrix.”的提示，得知在栈初始化将 identity matrix 放入栈中。
- Clear 注解“Revert to just containing the identity matrix.”，即 clear 容器之后，将 identity matrix 放入栈中。
- Top 和 pop 直接调用 vector 中的 back 和 pop 方法即可。
- 需要注意的是 push 操作，因为是层次进行建模，最底层的节点需要进行所有祖先节点的叠加，即乘积，所以进行压栈时需要将压栈元素与栈顶相乘。

2.skel 文件解析并装入

运行程序发现执行的文件名为

```
data/Model4.skel
```

打开指定的文件

```
0.462702 0.510341 0.510937 -1
0 0 0 0
0.005663 -0.054877 0.002566 1
0 0 0 0
0.000058 -0.029687 0.000456 2
-0.038986 0.010282 0.001456 4
-0.027839 -0.127258 0.001539 5
-0.011721 -0.189657 -0.016647 6
0.000058 -0.029687 0.000456 2
0.050455 0.026546 0.005865 8
0.031276 -0.133588 -0.003279 9
0.014518 -0.204181 -0.015486 10
0 0 0 0
-0.105144 0.019533 0.039457 12
-0.087850 -0.136181 0.013335 13
0 0 0 0
0.105745 0.024512 -0.051900 15
0.133815 -0.119244 -0.080915 16
```

前三个字段是浮点数给出关节相对其父关节的平移。第四个字段是其父关节的标号。

即将读入的三个字段进行平移后转化成 Matrix4f 其次矩阵，作为变换矩阵，然后保存到相应父关节标号下的 children 即可。如果标号是-1 则保存到 m_rootJoint（根节点）。

3.绘制关节

层次结构保存关节，需要递归绘制。Opengl 绘制的过程为先将变换压入栈，等到所有变换都压入栈之后，再从栈顶逐一弹栈操作。这里的原理相同，不过需要递归操作，递归过程为，先将根节点的变换压入栈，然后寻找根节点的子节点，压入子节点的变换入栈，以此类推，遍历所有子树之后，再弹栈，用 glutSolidSphere(0.025f, 12, 12)进行绘制。

4.绘制骨骼

递归过程大致与绘制关节相同，首先将所有的变换压入栈中，然后 pop 出变换。接下来开始对变换进行修改。先沿 z 轴平移 1 个单位，直接调用 scaling 返回一个 Matrix4f 作为变换矩阵 m1。接下来进行缩放[0.025, 0.025,L]T，L 下一关节与当前关节的距离，可以直接通过变换矩阵保存的平移量表示，因为代表的是相对父节点的平移量，分别平方和开放则为与父节点的距离，实现为获取 transform 的 getCol 中的第三列的所有元素,在调用 abs 求距离,最后调用 scaling(0.025, 0.025, L)，返回一个 Matrix4f 作为变换矩阵 m2。最后进行 z 轴旋转直接调用 normalize，根据老师建议直接调用映射 y =(z × rnd).normalized()，而 x =(y × z).normalized()，其中 rnd 为 [0, 0, 1]T，生成旋转矩阵 m3，三个变化矩阵都存入栈中。使用 glutSolidCube(1.0f)画图，再将三个变化矩阵都弹栈。

5.旋转骨骼

使用 rotateX，rotateY，rotateZ 分别将相对 x,y,z 轴的旋转量转化成相应的旋转变化矩阵。然后将原变化矩阵的旋转部分通过 setSubmatrix3x3 修改相应的值。

2.2 第二部分

1.读点面数据并绘制

这里的读点面操作和绘画操作和作业 0 相似，只不过不同的是作业 0 没有法向量，所以当时绘制出的图形显示时，并不能看出来兔子是以三角形进行拼接而成。本作业原始代码已经提供了叉乘的方法直接调用，法向量的计算方法即一个面上的任意两个向量进行叉乘，这里使用(v2-v1)X(v3-v2),代码表示为:Vector3f::cross(v1 - v2, v1 - v3).normalized();

2. attach 文件解析保存

这里注意 attach 文件对关节 1 进行缺省操作，所以在保存关节权重值时，关节数会少 1，并且根关节的权重为 0。

3. 世界坐标与局部坐标互转

- 世界坐标转化成局部坐标。因为目前的模型坐标是基于当前世界坐标进行变换而成，如果要将世界坐标转化成局部坐标，那么局部坐标里的点就要以局部坐标系为世界坐标，即映射到世界坐标中，那么就是进行逆操作 T^(-1)，可以想成局部坐标系原点移动到世界坐标系的原点中。
- 想象如果局部坐标系已经移动到世界坐标系原点，那么如果想要回到原处，就要进行上一操作的逆操作，即 T 变换，这就完成了局部坐标转化成世界坐标的过程。

4. 变换网格

只需要读入权重数据，通过加权操作进行赋值即可，公式如下:

ΔP = W * CurrentJointToWorldTransform * BindWorldToJointTransform * P

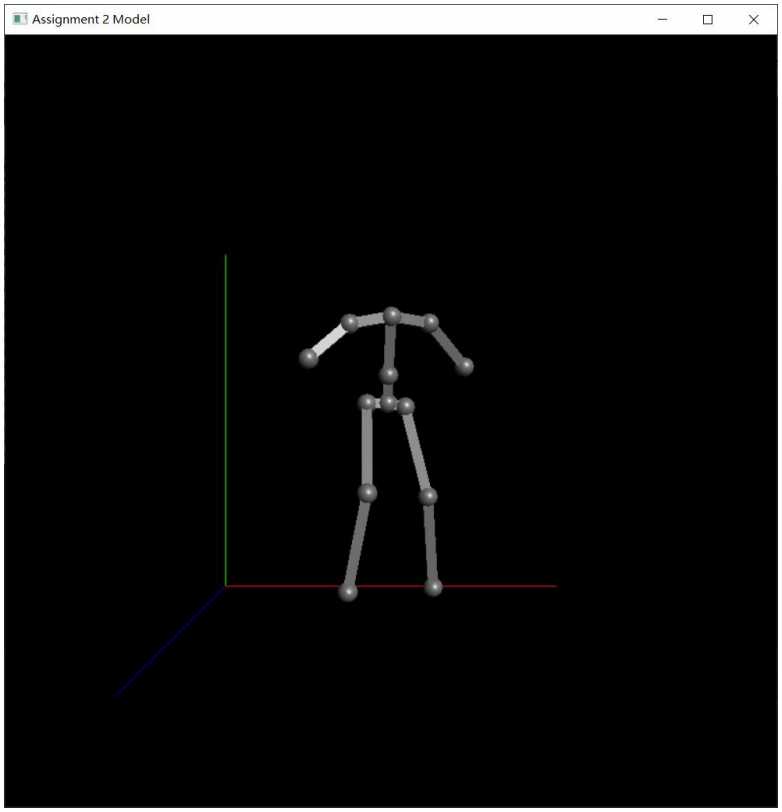
则具体的变换过程为先将世界坐标转化成局部坐标，然后进行变换完成之后再转化回世界坐标，之后再复制权重。

3. 问题与解决

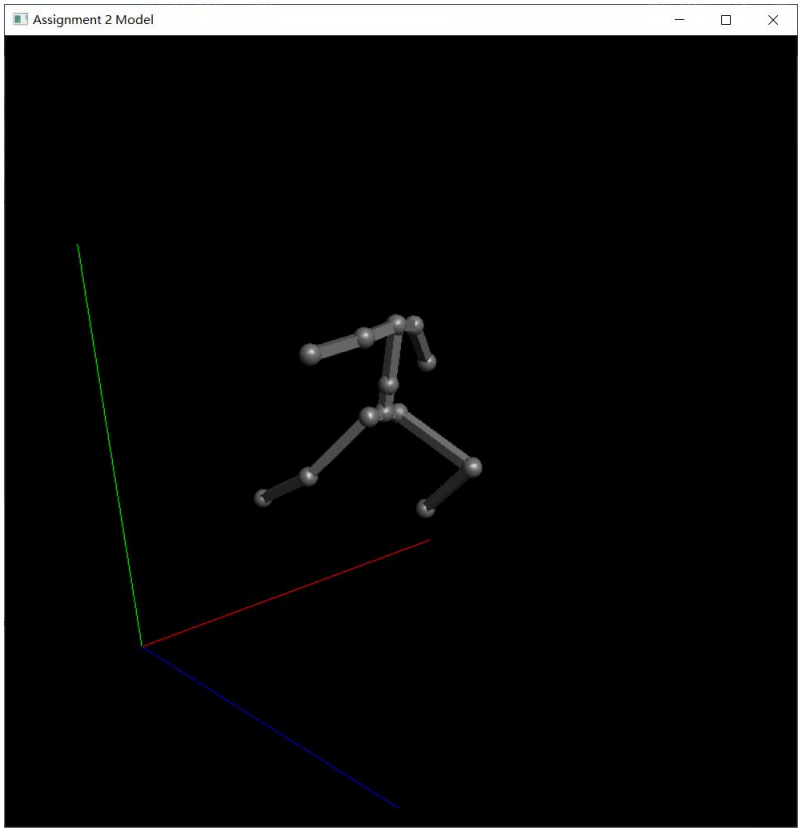
- 在实现第一部分的步骤 5 中，如果直接将旋转矩阵 3x3 赋值给关节的变换矩阵中，会发错误因为先前已经进行了相对于原点的平移操作，直接赋值模型就会回到坐标原点，并且都进行旋转操作，解决方法就是仅修改旋转矩阵控制的左上方 3x3 部分。4x4 矩阵其他部分不修改。
- 纠结了很久世界坐标与局部坐标的相互转换的关系，思路在第二部分的第三点进行说明。
- 在进行旋转时，因为要进行计算权重等信息，所以要花费一些时间，所以会发生卡顿现象。
- 附加部分没有实现。

4. 效果展示

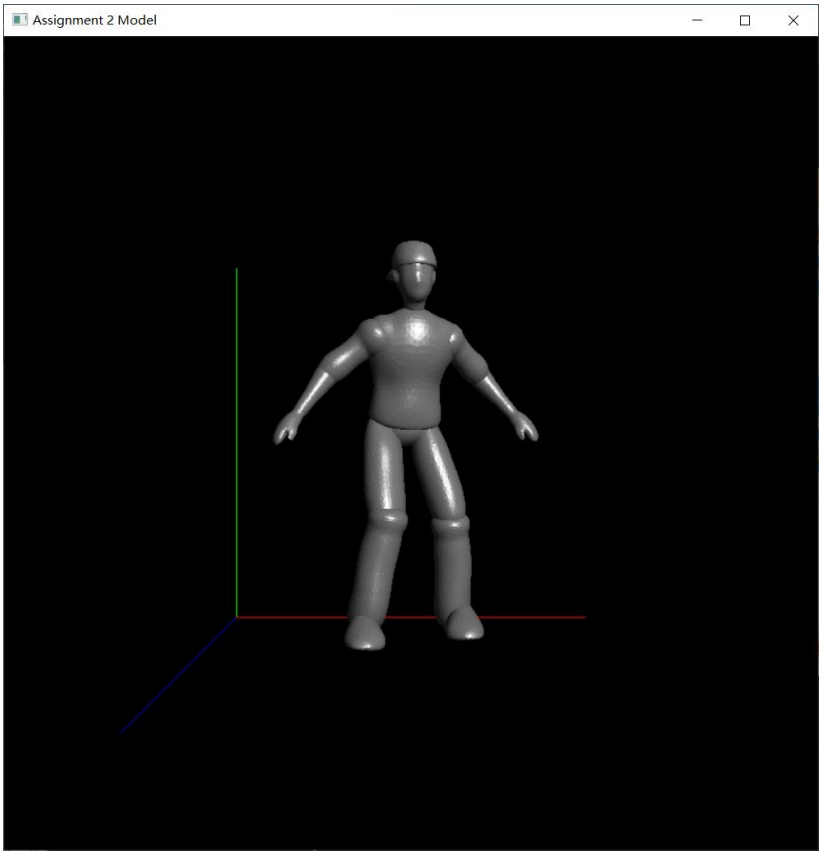
- 未经旋转骨骼模型



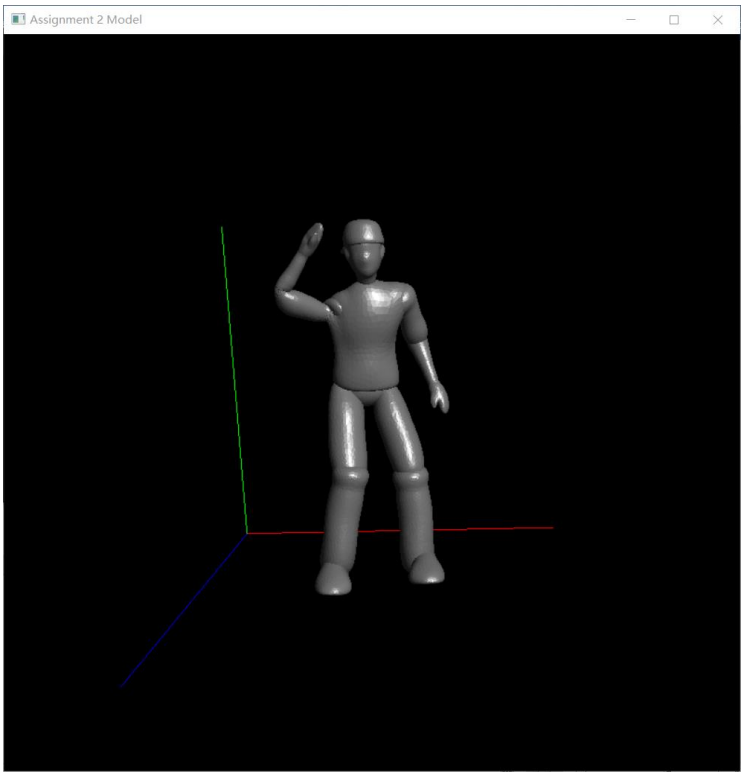
- 进行旋转后的骨骼模型



- 未进行旋转的模型



- 旋转后的模型


Assignment 2 Model