READ ME

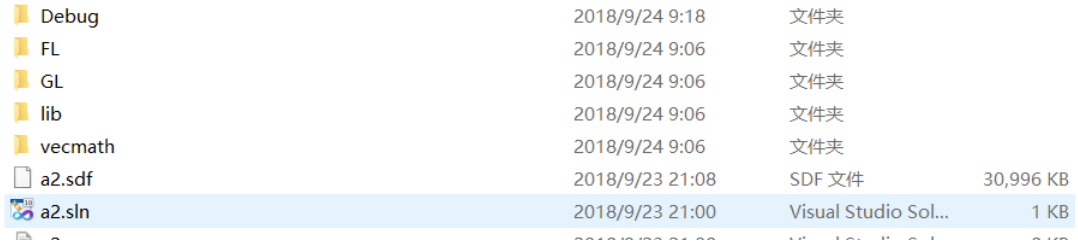
姓名：周宁
班级：111171
学号：20171004140

一、实习思路：

1.如何编译和运行代码

①将“作业 2 VS2017 版本”这个文件夹下的“distrib” 文件夹，复制到平时写代码的工作目录下（不复制也可以）

②直接双击“distrib”文件夹下的 a2.sln 文件即可自动打开 Visual Studio 进行代码编写。



2. 矩阵栈代码的编写

需要填写“MatrixStack.cpp”中相应的函数体，可以参照上面写的注释和“MatrixStack.h”中相对应的代码。

①实现构造函数（MatrixStack()）。可以在 MatrixStack.h 中看到私有成员为 **vector< Matrix4f > m_matrices**，明白了整个矩阵栈的实现都是靠 **m_matrices** 这个数组实现，而且查看注释可知只需要“用单位矩阵初始化矩阵栈”，所以构造函数只需要在 **m_matrices** 中 **push** 进一个“**Matrix4f::identity()**”即可。

②实现“void clear()”。由注释可以知道“恢复矩阵栈为仅包含单位矩阵”，可以先将数组 **m_matrices** 进行 clear()一下，然后再 **push** 一个“**Matrix4f::identity()**”即可。

③实现“Matrix4f top()”。由注释可以知道“返回栈顶的矩阵”，随意只需要返回最后一个 **push** 进栈的矩阵即可，我采用的是返回“**m_matrices.back()**”

④实现“void push(const Matrix4f& m)”。由注释可知“Your stack should have OpenGL semantics: the new top should be the old top multiplied by m”，就是新的栈顶矩阵应该为老的栈顶矩阵（“old top”）乘以传进来的矩阵“m”，即把两者相乘再 push 到 **m_matrices** 就行，但是在乘的时候需要注意不要将乘的两个顺序弄反了，如果弄反了将会造成许多错误。

⑤实现“void pop()”。这个比较简单我采用的方法是直接调用“this->m_matrices.pop_back()”。

3. 读取“.skel”后缀的文件

这个功能需要实现“SkeletalModel.cpp”中的 loadSkeleton(const char* filename)的函数体，可以先阅读一下“SkeletalModel.h”当中的代码，再阅读一下“Joint.h”当中的代码。

①阅读文档可知“.skel”后缀的文件的格式：每行包含 4 个用空格分隔开的字段。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（关节标号是其在.skel 文件中出现的顺序，从零开始），其中“-1”作为父节点。

②知道格式之后就比较简单，我采取的是使用 fscanf(file, "%f %f %f %i", &x, &y, &z, &index)按照格式读取。

③采用循环的方式读取文件，然后在循环体里面，对每个 joint 的 transform 进行设置，并且构建出对应的骨架树，。循环时先判断读取的 index 是否“-1”，是的话将 m_rootJoint 指向它；不是就找出 m_joints[index]，即找出读取的这个结点的父节点，并且将其加入到父节点的子节点当中去，最后将结点加入到 m_joints 当中，如下图所示。

```
if (index == -1)          //说明是根节点
    this->m_rootJoint = joint;
else {
    //说明不是根节点，那么需要寻找其父节点,给父节点添加孩子节点
     this->m_joints[index]->children.push_back(joint);
}
this->m_joints.push_back(joint);
```

4. 画出骨架 Joints

①采取编写一个私有的 void drawJoints(Joint* joint)函数，这个函数的功能是一个专门的递归函数，来实现实现从根向骨骼出发，遍历关节层次，在 void drawJoints () 中调用 drawJoints(m_rootJoint)。

②在 void drawJoints(Joint* joint)当中，先把 joint 中的 transform 压到栈 m_matrixStack 当中去；然后判断 joint 是否有子结点，如果有子结点，就遍历子结点，对每个子节点进行 drawJoints(child)，进行递归。

③递归到叶子结点或者子节点都被递归完，进行 glLoadMatrixf(m_matrixStack.top())，加载当前的矩阵，然后再进行画球，最后再进行 m_matrixStack.pop()，把开始 push 的矩阵给 pop 出去，就可以返回父结点。

5.画出骨骼 Bones

①采取编写一个私有的 **void drawSkeleton (Joint* joint)**函数，这个函数的功能是一个专门的递归函数，来实现实现从根向骨骼出发，遍历关节层次，在 **void drawSkeleton ()** 中调用 **drawSkeleton (m_rootJoint)**。

②在 **void drawSkeleton (Joint* joint)**当中，类似前面些的 **drawJoints(Joint* joint)** 函数，先把 **joint** 中的 **transform** 压到栈 **m_matrixStack** 当中去；然后判断 **joint** 是否有子结点，如果有子结点，就遍历子结点，对每个子节点进行 **drawSkeleton (child)**，进行递归。

③递归到叶子结点或者子节点都被递归完，先用 **Matrix4f::translation(0, 0, 0.5)**得出平移举证 m1，再使用 **Matrix4f::scaling(0.025, 0.025, L)**得到缩放矩阵 m2（其中 L 等于 z.abs()），然后再利用 z 和 rnd（rnd = Vector3f(0, 0, 1)），计算出向量 x、y,得到旋转矩阵 m3，其中 m3 的设置如图所示，因为旋转是影响前面的 3*3 的矩阵导致的，所以这样设置。

```
Matrix4f m3 = Matrix4f(
    x[0], y[0], z[0], 0.0f,
    x[1], y[1], z[1], 0.0f,
    x[2], y[2], z[2], 0.0f,
    0.0f, 0.0f, 0.0f, 1.0f);     //进行旋转
```

④将上述按照 m3、m2、m1 的顺序压入到栈 **m_matrixStack** 当中（先变化的后加入都栈中），然后加载当前矩阵再进行画立方体就能得到骨骼。最后将这三个矩阵 pop 出来，返回父节点

6.用户接口

当用户拖动关节旋转滑块时，应用程序会调用 setJointTransform, 传入需要更新的关节的序号（下标 index）和用户设置的欧拉角，我们需要实现此函数来正确设置关节变换矩阵。

①利用 **Matrix4f::rotateX(rX)**等得到 x、y、z 方向上的旋转矩阵，然后将这三个矩阵相乘得到最终的旋转矩阵 m。

②设置 **m_joints[jointIndex]**的 transform 矩阵，将其前面的 3*3 设置为 m 前面的 3*3 矩阵。原因是因为旋转是前面 3*3 的矩阵导致的，所以这样设置。

7.读取“.obj”后缀的文件

读取 obj 文件再作业 0 就已经实现过了，所以在这里再次实现比较简单。

①使用 fscanf(file, "%c ",&c)，先读取字符，判断字符是‘v’还是‘f’。

②如果是‘v’，则使用 **fscanf(file, "%f %f %f", &x, &y, &z)**，读取三个坐标，并且把这个点加入到 bindVertices 中

③如果是‘f’，则使用 **fscanf(file, "%d %d %d", &v1, &v2, &v3)**，读取三个点的下标，然后使用 **faces.push_back(Tuple3u(v1, v2, v3))**加入到面当中。

④最后令 currentVertices = bindVertices。

8. 网格渲染

网格渲染在作业 0 的时候已经玩过了，说白了就是画三角形比较简单，集体的实现代码在 draw()函数当中。

①对 faces 进行遍历，找出每个 face 对应的三个点，然后对这三个求法向量后，设置相应的法向量，再进行三角形的绘制即可，但是值得注意的是绘制的时候，是选取 currentVertices 数组当中的点进行绘制，而不是 bindVertices 当中的点。

9. 读取“.attach”后缀的文件

①由文档可知.attach 文件的格式，为结点附着到第（i+1）个关节的强度，然后第 0 个关节的权重为 0。并且最终需要完成 **void loadAttachments(const char* filename, int numJoints)**函数。

②我采取的是使用 ifstream in(filename, ios::in)进行读取。先创建一个 **vector< float > a**，接下来的代码如图所示。

```
while (attachments.size()<=bindVertices.size()) {
    vector< float > a;        //存储权重数组
    a.push_back(0);           //根关节权重为0

    for (int i = 0; i < numJoints-1;++i) {
        in >> weight;
        a.push_back(weight);
    }
    this->attachments.push_back(a);

}
```

10. 计算变换

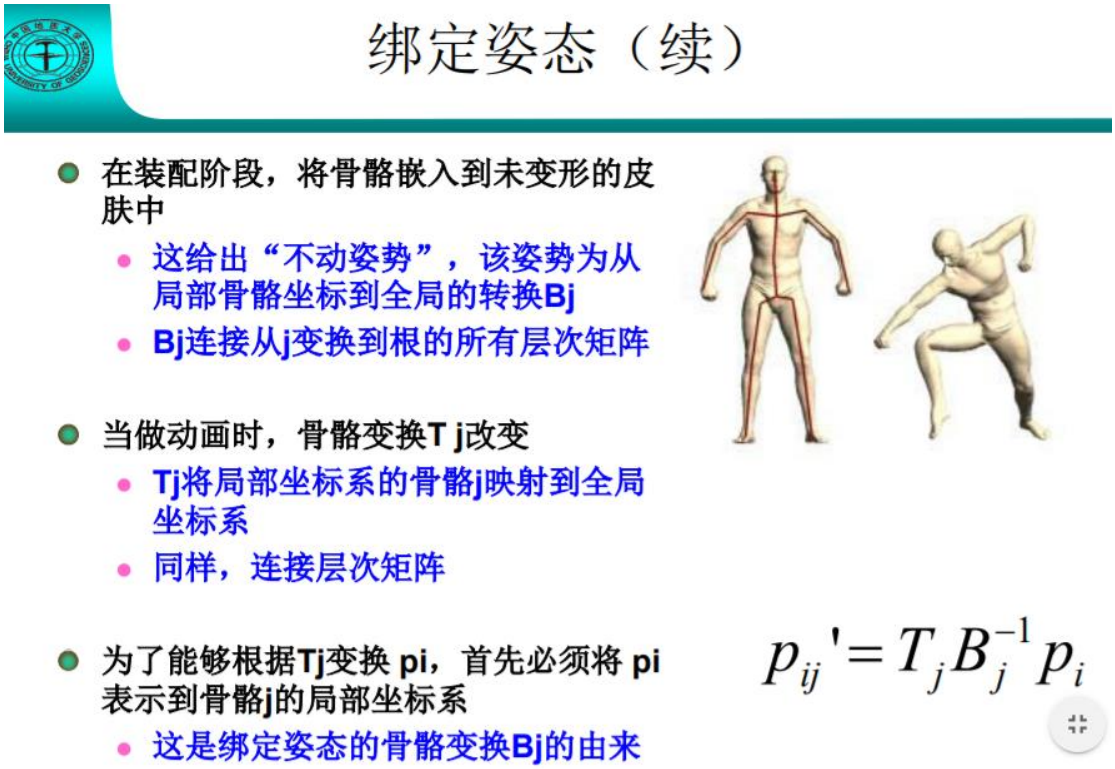
在实现计算变化之前，需要先搞清楚 bindWorldToJointTransform、currentJointToWorldTransform 的具体含义。其中 currentJointToWorldTransform 是将局部坐标系映射到全局坐标系，其实就是 ppt 当中的T_j，其中 bindWorldToJointTransform 的含义是从世界坐标系转化到局部坐标，但是我个人的感觉是将皮肤绑定到相对于骨骼坐标，相对于 ppt 当中的B_j^{-1}。bindWorldToJointTransform 在最开始时计算，在计算之后并不会进行改变，因为这是这个点相对于骨骼的矩阵，后面滑动不会再改变这一点，并且 bindWorldToJointTransform 最开始和 currentJointToWorldTransform 应该互为逆矩阵。

①实现函数 **updateCurrentJointToWorldTransforms**（），为了实现这个函数，编写了一个函数 **void updateCurrentJointToWorldTransforms(Joint* joint)**，里面的实现方式类似

于前面写的 drawJoints(Joint* joint)，不过在递归到叶子结点或者子节点时，直接使用 joint->currentJointToWorldTransform = m_matrixStack.top()。

②实现函数 computeBindWorldToJointTransforms（），为了实现这个函数，编写了一个函数 void computeBindWorldToJointTransforms (Joint* joint)，里面的实现方式类似于前面写的 drawJoints(Joint* joint)，不过在递归到叶子结点或者子节点时，直接使用 joint->bindWorldToJointTransform= m_matrixStack.top().inverse()。

参考的 ppt：



11. 变换网格

变换网格其实求出 currentVertices 数组的每个元素的坐标，这个坐标受每个关节影响，影响的权值由我们之前读入的.attch 文件所决定（有些权值为 0）。最终计算可以采用公式

$p_i(t)=\Sigma\ w_{ij}\ T_j(t)\ B_j^{-1}p_i$ 计算出相对应的点的坐标。

①for 循环遍历 bindVertices 的各个点，并且需要将每个点转化为 Vector4f，然后再遍历每个点对应的权值数组，然后按照公式求出对应点即可。

二、实习中遇到的困难和解决方法：

1.画出骨骼时开始矩阵相乘的顺序相反，导致画出的东西乱七八糟的。
解决方式：按照先变化的后乘的原则，将乘的顺序交换一下即可。

2.进行网格渲染时，总是报越界错误。
解决方式：通过调试发现 faces[i][j]的值超过了 currentVertices 的大小，原因是未进行“-1”，直接在原有基础下，进行下标“-1”即可。

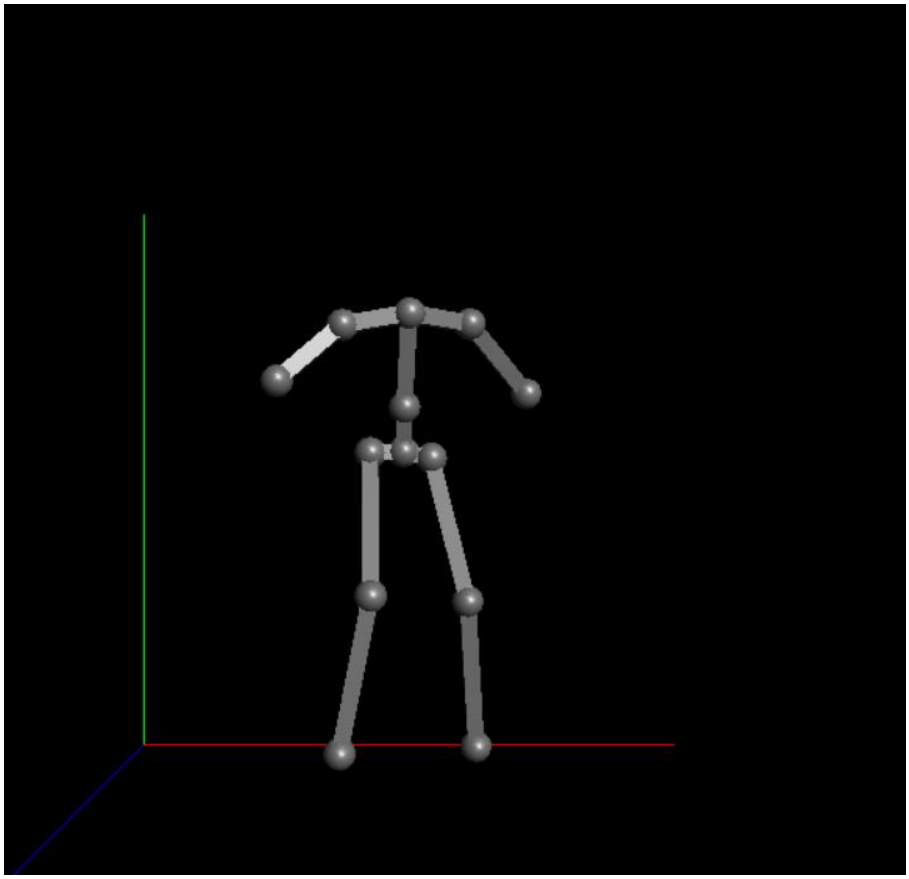
3.写计算网格时理解 CurrentJointToWorldTransform、BindWorldtoJointTransform 理解不了。
解决方式：查看 ppt，在回忆老师上课所讲解的内容，逐渐理解 CurrentJointToWorldTransform 和 BindWorldtoJointTransform 的意义。

4.开始写 computeBindWorldToJointTransforms 和 updateCurrentJointToWorldTransforms 函数时，一开始没有进行 m_matrixStack 的 clear，所以导致最后缩放的大小有问题
解决方式：在查看代码后，思考之后觉得这里需要 clear 一下，不然矩阵栈里面可能不是单位矩阵会影响最后的缩放。

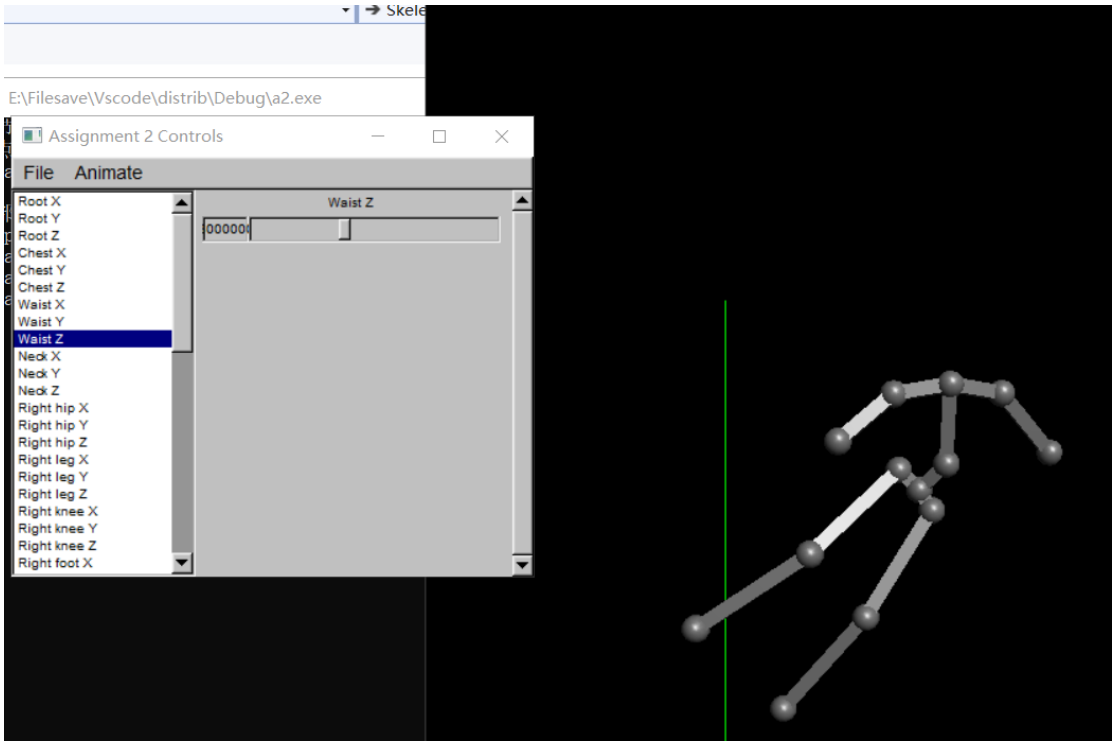
三、实习结果展示

Model1:

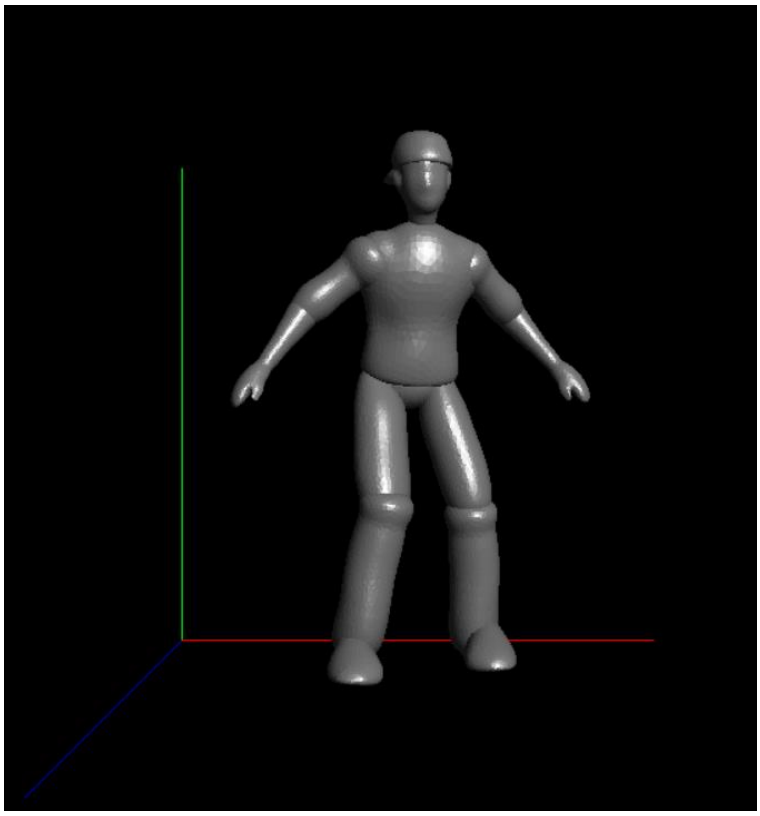
杆状图形:



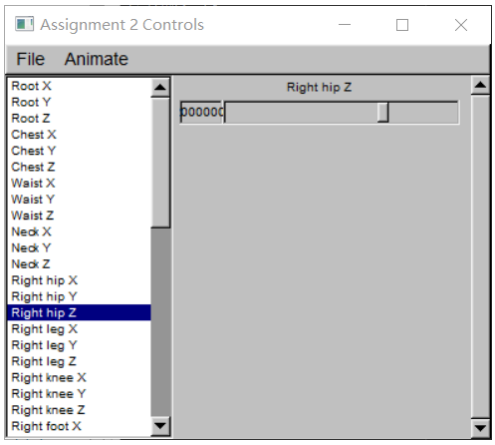
旋转关节：

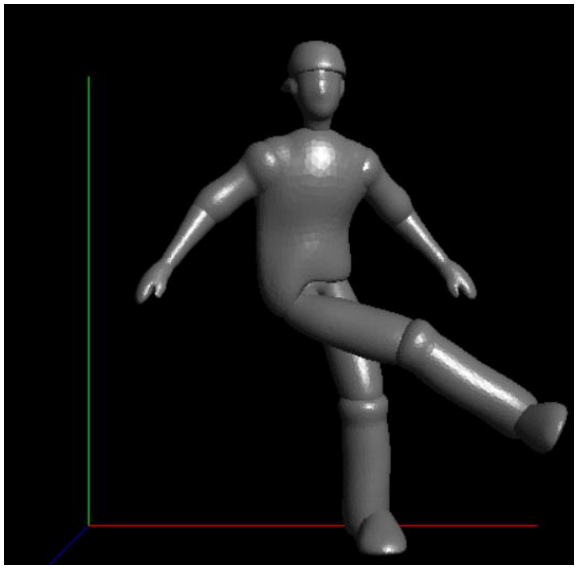

E:\Filesave\Vscode\distrib\Debug\a2.exe
Assignment 2 Controls
File
Animate
Root X
Root Y
Root Z
Chest X
Chest Y
Chest Z
Waist X
Waist Y
Waist Z
Neck X
Neck Y
Neck Z
Right hip X
Right hip Y
Right hip Z
Right leg X
Right leg Y
Right leg Z
Right knee X
Right knee Y
Right knee Z
Right foot X
Waist Z

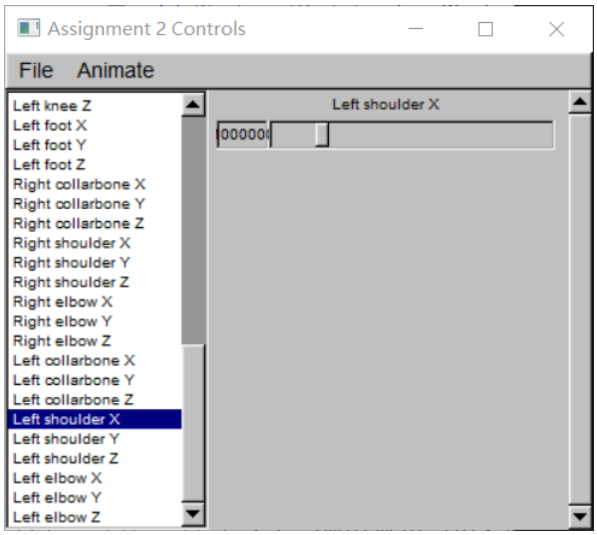
网格：

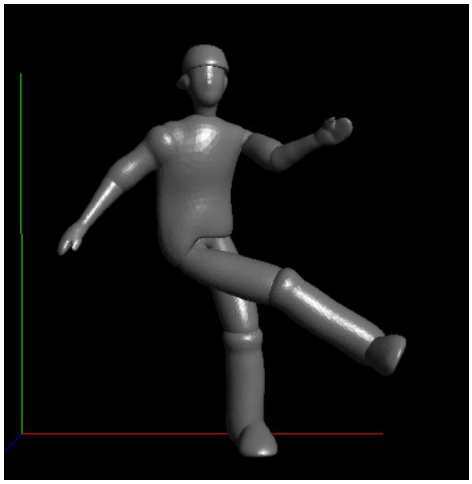


进行滑块移动：


Assignment 2 Controls
File Animate
Root X
Root Y
Root Z
Chest X
Chest Y
Chest Z
Waist X
Waist Y
Waist Z
Neck X
Neck Y
Neck Z
Right hip X
Right hip Y
Right hip Z
Right leg X
Right leg Y
Right leg Z
Right knee X
Right knee Y
Right knee Z
Right foot X
Right hip Z

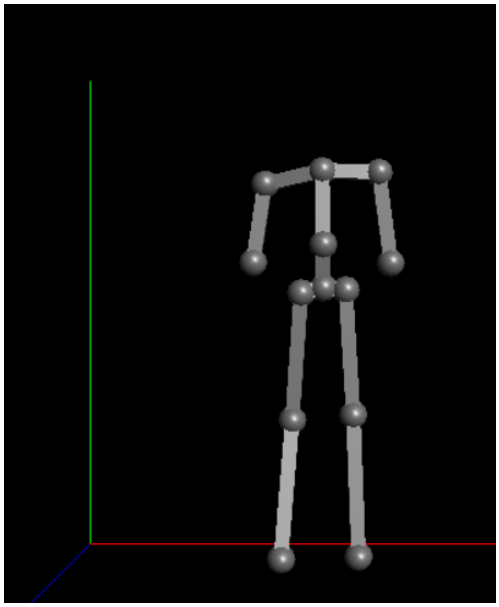



Assignment 2 Controls
File Animate
Left knee Z
Left foot X
Left foot Y
Left foot Z
Right collarbone X
Right collarbone Y
Right collarbone Z
Right shoulder X
Right shoulder Y
Right shoulder Z
Right elbow X
Right elbow Y
Right elbow Z
Left collarbone X
Left collarbone Y
Left collarbone Z
Left shoulder X
Left shoulder Y
Left shoulder Z
Left elbow X
Left elbow Y
Left elbow Z
Left shoulder X

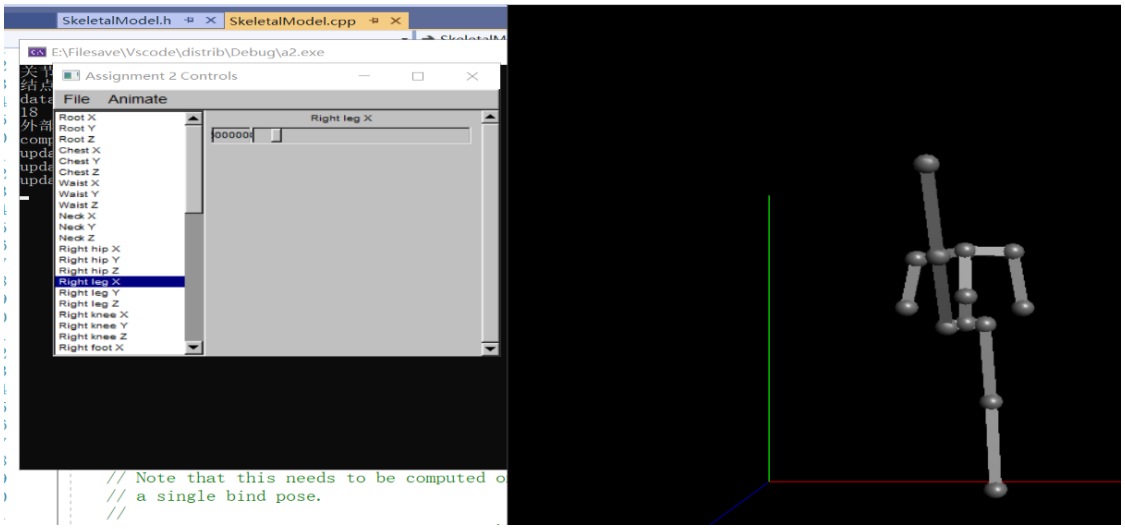


Model2:

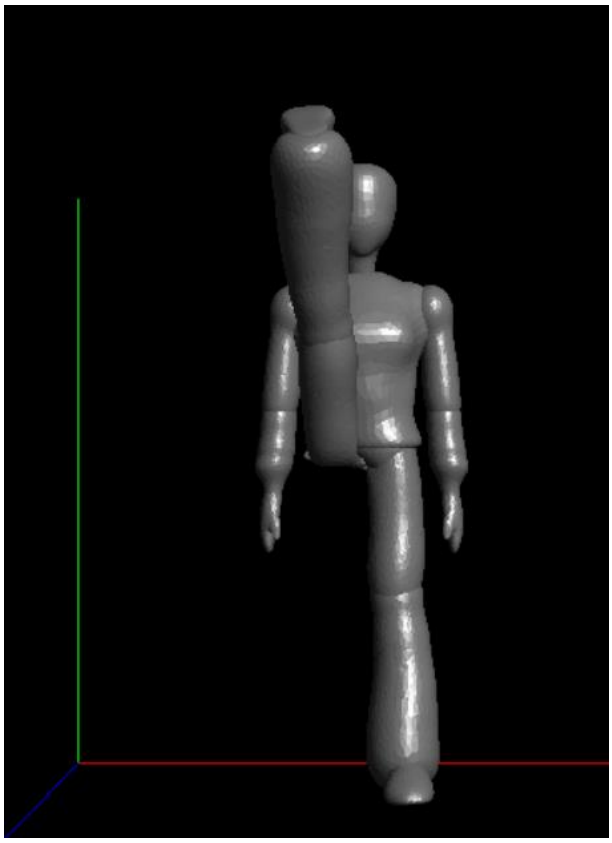
杆状图形:



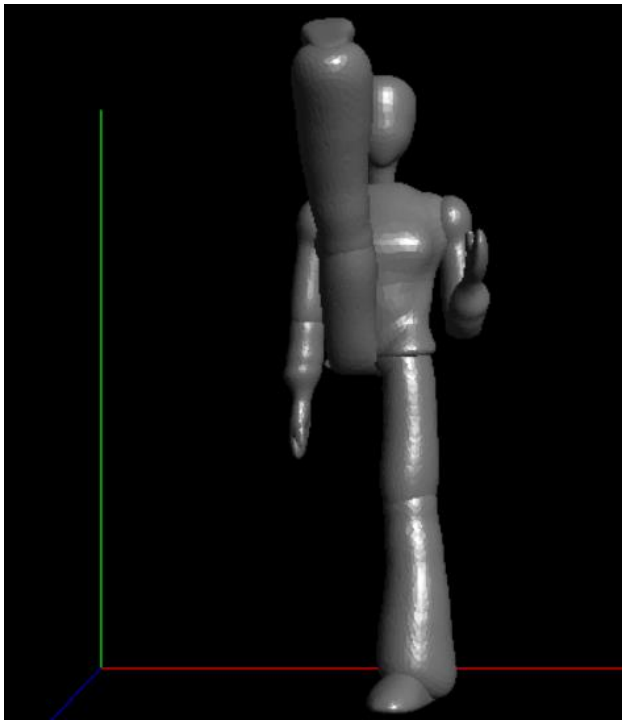
关节旋转:



网格：

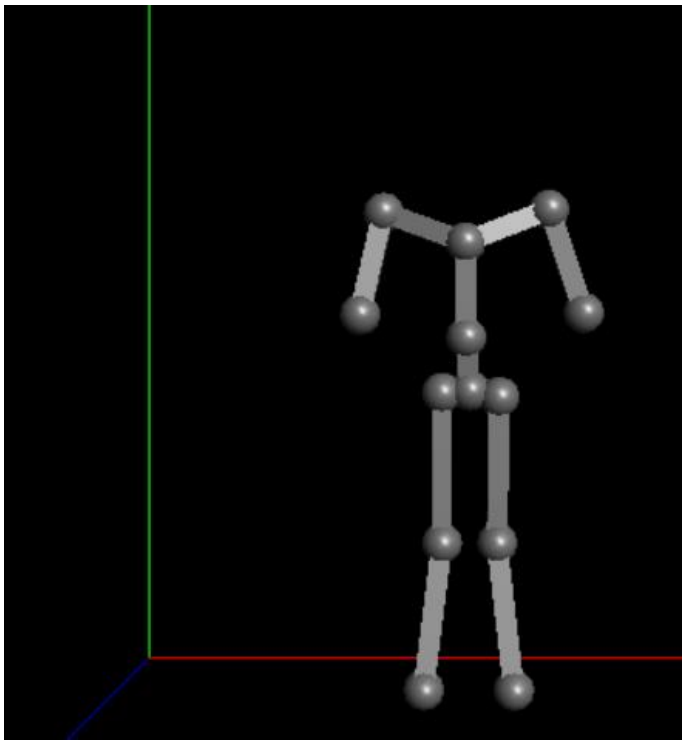


进行滑块滑动：

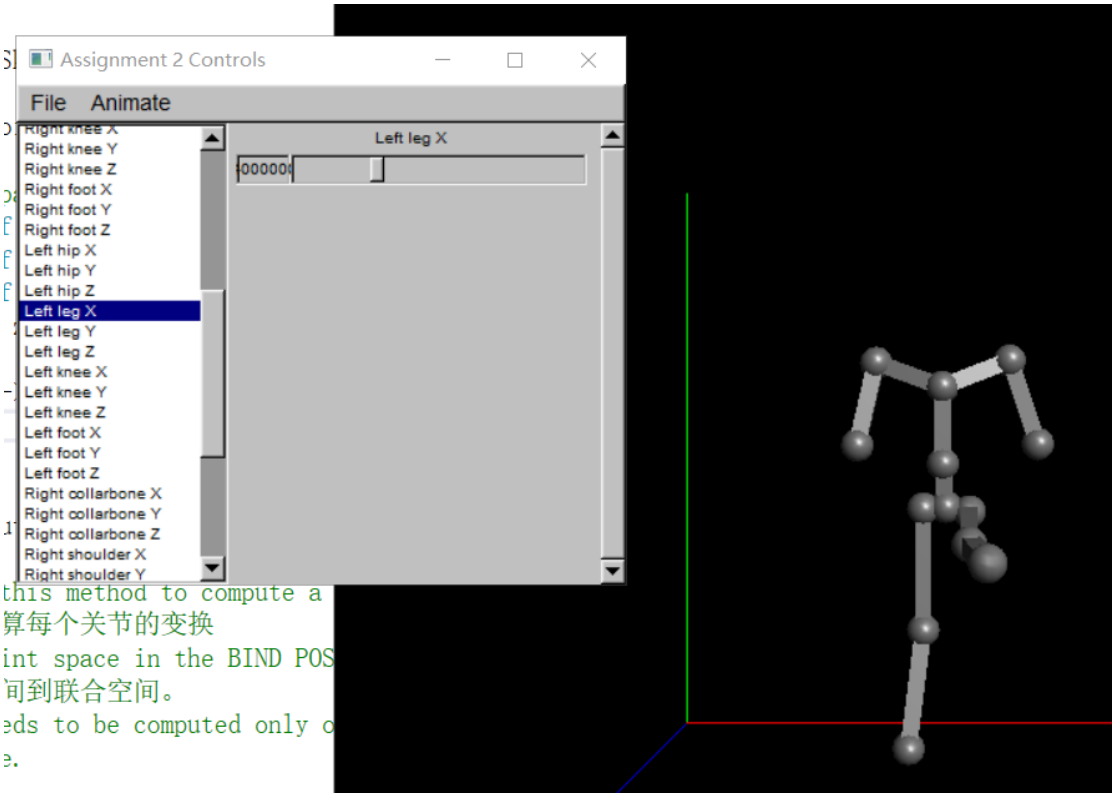


Model3:

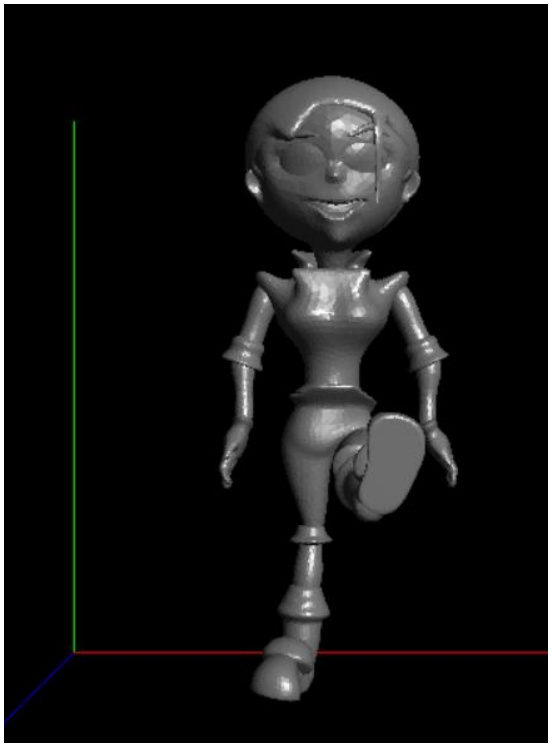
杆状图形：



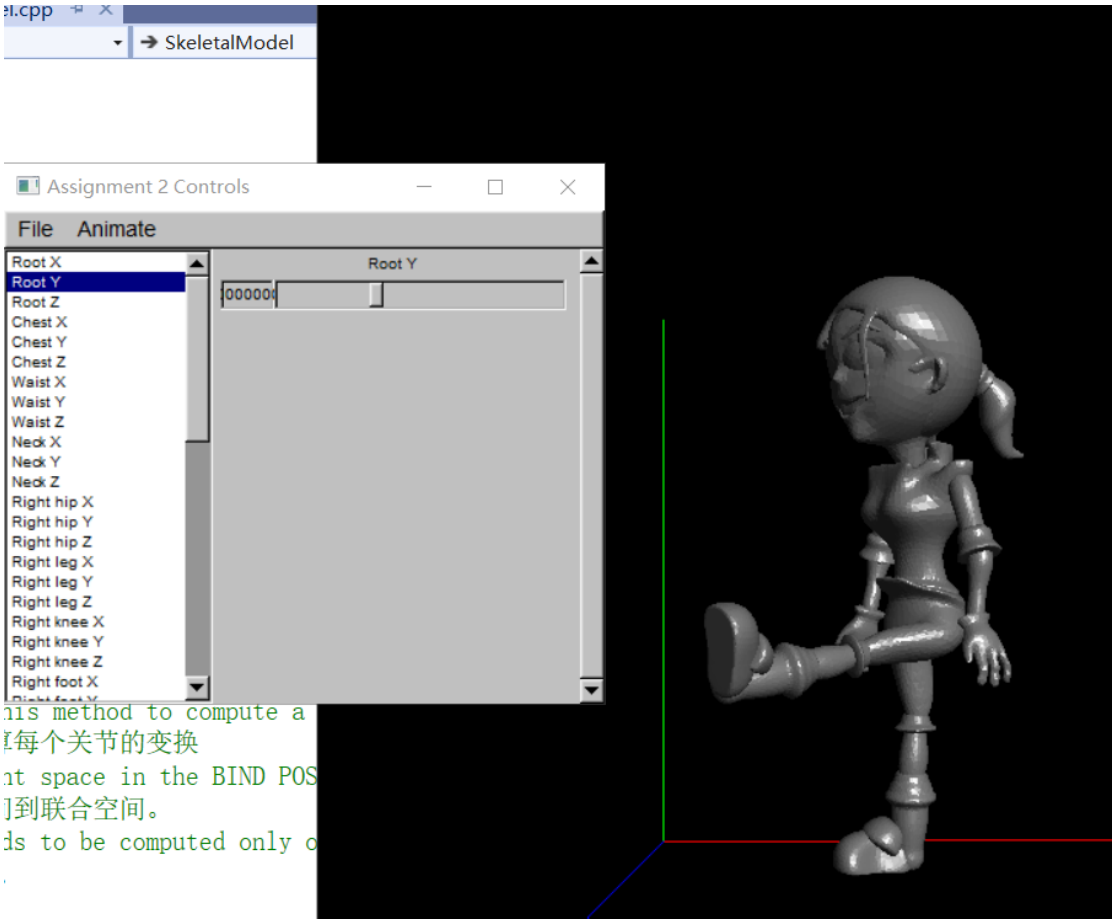
关节旋转：



网格：

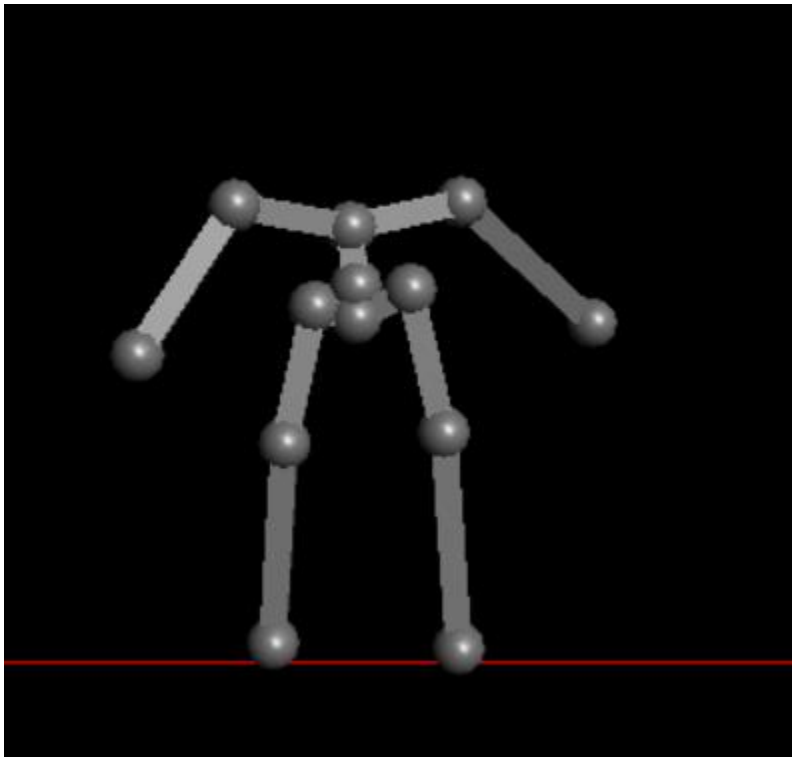


进行旋转：

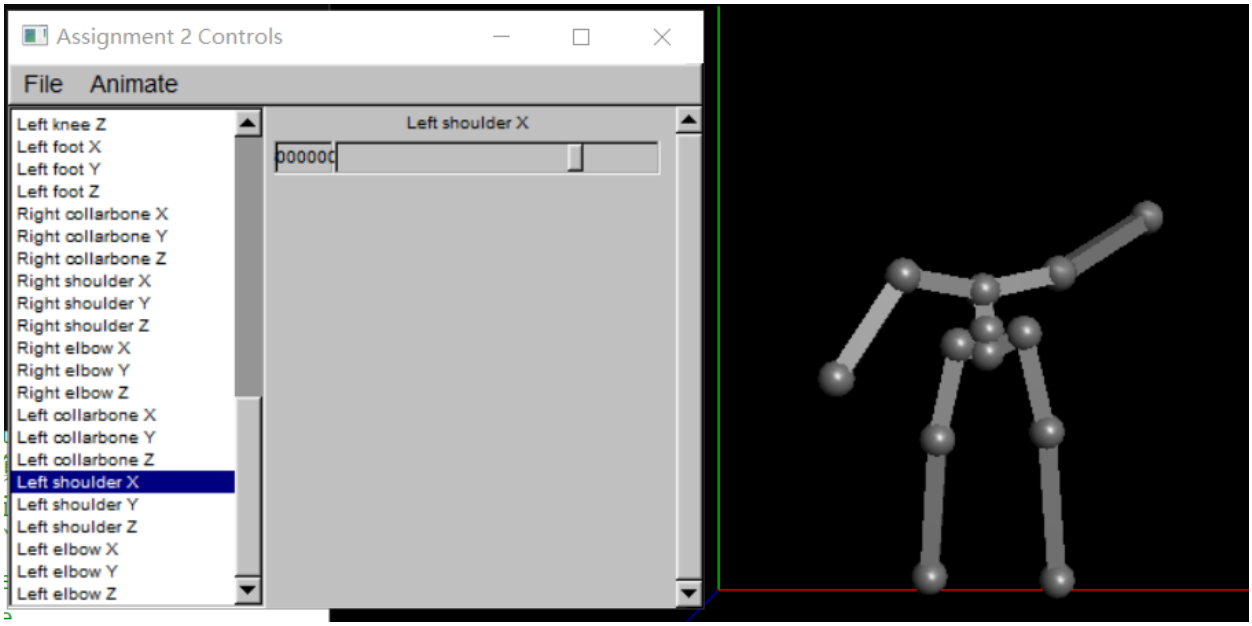

SkeletalModel
Assignment 2 Controls
File
Animate
Root X
Root Y
Root Z
Chest X
Chest Y
Chest Z
Waist X
Waist Y
Waist Z
Neck X
Neck Y
Neck Z
Right hip X
Right hip Y
Right hip Z
Right leg X
Right leg Y
Right leg Z
Right knee X
Right knee Y
Right knee Z
Right foot X
Root Y

Model4:

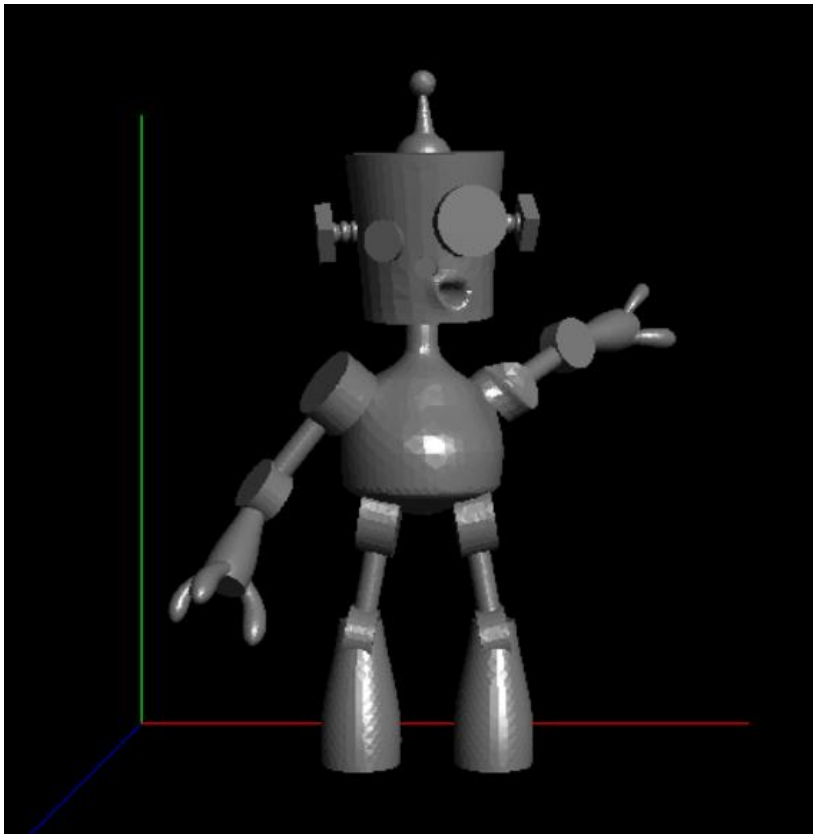
杆状图形：



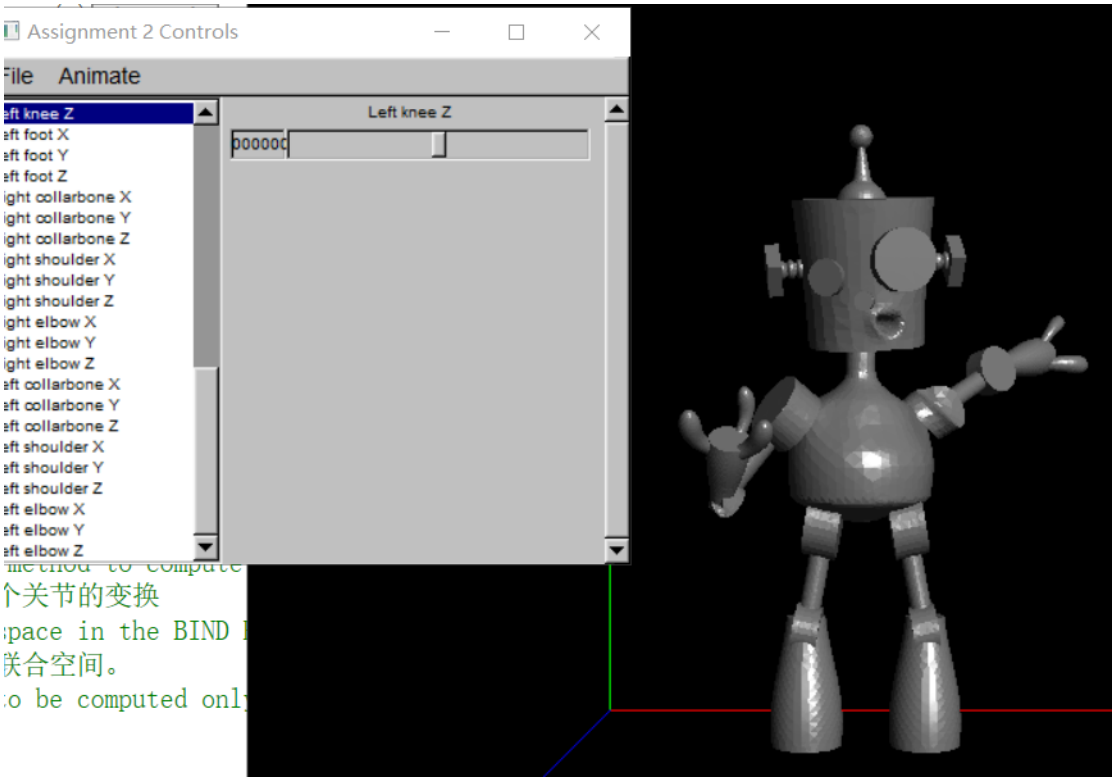
关节旋转：



网格：



滑块移动：


Assignment 2 Controls
File Animate
Left knee Z
个关节的变换
pace in the BIND
咲合空间。
o be computed onl

四、实习总结

本次实习是对层次模型和骨架子空间变换(SSD)的练习，在实习当中，我学会了层次模型以及骨架子空间变换，除此之外我对于矩阵的理解也多了一些，感觉又学习到了许多新的知识。但是也又许多需要改进的地方，首先是由于时间原因我附加分并没有写，其次是我代码读取 skel 文件似乎存在一些，问题，最后一行会进行两次读取，我只能使用比较“暴力”的方式将这个问题解决，就是当读取的次数超过 18 次时，自动退出，但是这个方法显然不是很好，但是我并没有方法对此进行改进。希望之后继续努力学习，学习到更多计算机图形学的知识。