**ASSIN2层级建模和SSD**

一、开始

使用示例解决方案，可以手动拖拽模型的关节；同样可以使用鼠标来变换视角，左键旋转，滚轮移动，右键缩放；示例解决方案包括了这次作业的全部内容，包括加载和展示一个骨骼，加载和骨骼绑定的网格，并且根据关节角度对骨骼和网格进行变换；可以按a和s来切换是否绘制坐标轴和骨骼的模式；

二、需求总结

1层级模型

你的实现必须正确地解析任意给定的骨骼文件(.skel)，并且构造一个场景图的数据结构，并且在关节中使用一个矩阵栈，通过简单的球体和圆柱体来绘制骨骼，最后你需要写代码实现从接口的输入中设置关节的转换；

2 SSD

这一部分要实现骨骼子空间变形，连接皮肤到骨骼上。这部分需要使用到作业0的解析网格代码（除去法线部分），并且在显示的时候生成它们。同时还需要写另外一个解析器来加载“连接权重”——指定了每一个顶点对于每一个骨骼的重要性。最后，真正实现SSD算法，需要应用一系列运算到骨骼层级上。

3.Artifact

制品，通过截图的方式加载角色模型（PNG/JPEG格式图片），

三、层级模型

1. 矩阵堆栈

（1）在**MatrixStack.h**中定义，在**MatrixStack.cpp**中实现（目前为空），建议对堆栈只使用vector；

（2）渲染时，将矩阵推入和弹出堆栈。在使用VertexRecorder类或drawSphere（）和drawCylinder（）函数绘制对象之前，必须更新transformation uniform变量；

（3）相机类有一个辅助接口，void Camera::SetUniforms(GLuint program, Matrix4f M = Matrix4f::identity());

要为后续绘制调用设置当前堆栈，请调用camera.SetUniforms（program，m\_matrixStack.top（））

要将模型矩阵重置为标识（identity），请调用不带第二个参数的方法：camera.SetUniforms（program）

1. 层级骨骼

（1）文件输入(**loadSkeleton()**方法，**SkeletalModel.cpp**文件)

.skel文件，是骨骼的文件格式，每行数据有4个字段，前三个是关节相对于其父关节的平移(translation)；最后一个字段是父物体的索引（从0开始）；

形成了一个有向无环图，根节点是-1，它的平移是是它自己在世界坐标系中的全局位置。

关节需要用指针来指向，因此有一些需要注意的地方，比如不能声明为局部指针，出范围会被析构，要用new字动态分配；

在**LoadSkeleton**方法中，需要创建一个关节的层次结构，这里的关节包含了（维护）一个指向关节的指针列表（关节是它的子节点）；还必须填充m\_joints这个关节列表，还有设置m\_rootJoint来指向根关节点。

（2）简笔画

**关节：SkeletalModel::drawJoints()**方法，任务是构造一个递归的方法，可以从**drawJoints()**方法里面调用，然后从根节点开始，遍历整个关节层次结构，使用矩阵堆栈来绘制每个关节的球体。代码已经提供了绘制球体的基本方法**drawSphere()**，记得在绘制前使用相机类来更新转换的uniform变量。

**骨骼：**骨骼其实就是在两个关节点之间绘制圆柱体，使用**SkeletalModel::drawSkeleton()**方法；就像关节一样，我们同样需要定义一个单独的递归方法来便利关节层次，然后在关节点和它的父关节之间绘制圆柱体（除非是根节点）。

绘制骨骼的时候，首先要定义骨骼的长度，我们可以调用**drawCyinder(6,0.02f,<length>)**方法来绘制一个骨骼（表示6个边，半径是0.02，长度是length），在调用绘制方法之前，必须要调用**Camera::SetUniforms()方法**，来移动圆柱体的位置到父节点的位置上，并旋转它到孩子关节的方向上。当绘制球体来显示关节的时候，记得跟示例代码对比下表现是否正确。

1. 用户接口

在关节旋转拖动条被拖拽的时候，这个应用调用**setJointTransform()**方法，传递关节的索引来得到更新，并由用户输入的数据去更新欧拉旋转角度。你应该实现这个方法，正确地设置关节的转换矩阵中的旋转分量。

四、SSD

4.1绑定姿势网格

使用作业0的代码来从obj文件中加载绑定姿势的顶点，样例代码自动使用了**Mesh::load**方法；唯一和作业0不同的是，我们提供的网格不包括法线；所以我的代码需要填充网格的“绑定顶点”部分和“面元部分”。注意网格结构体有两个顶点的拷贝，一份是绑定姿势、一份是当前的姿势，而当前的姿势就是由绑定姿势经过transformations来进行转换的。

4.2网格渲染

验证网格加载的正确性，示例代码自动使用了**Mesh::draw**方法；确定渲染使用的是m\_mesh.currentVertices而不是bindVertices；

与先前作业的网格不同，这些网格不提供任何基于顶点的法线，因为它们不是经过分析计算的。 相反，我们将通过获取边的叉积为渲染循环内的每个三角形生成一个单一法线。 不要忘了规范化法线。（请注意您的模型是如何“多面化”的：相邻法线之间的光照不连续，因为法线会突然变化。）

4.3连接权重

示例代码是用了Mesh::loadAttachments方法自动完成，连接的文件格式是.attach；它包含多行文本，网格中每个顶点一行。 每行包含关节数量减一个字段，以空格分隔。 每个字段都是一个浮点数，它指示顶点与第（i + 1）个关节的连接强度。 假设第0个关节的权重（根）为零。

4.4实现SSD

（1）计算转换

计算绑定姿势的世界到关节的变换（一次），然后运动姿势的节点到世界的变换（每次骨骼改变的时候），示例代码在适当的地方自动调用了**computeBindWorldToJointTransforms**和

**updateCurrentJointToWorldTransforms**方法；

computeBindWorldToJointTransforms应该设置每一个关节的bindWorldToJointTransform矩阵，我应该使用类似渲染骨骼的递归算法；注意要搞清楚哪一个空间是输入空间，哪一个空间是输出空间；

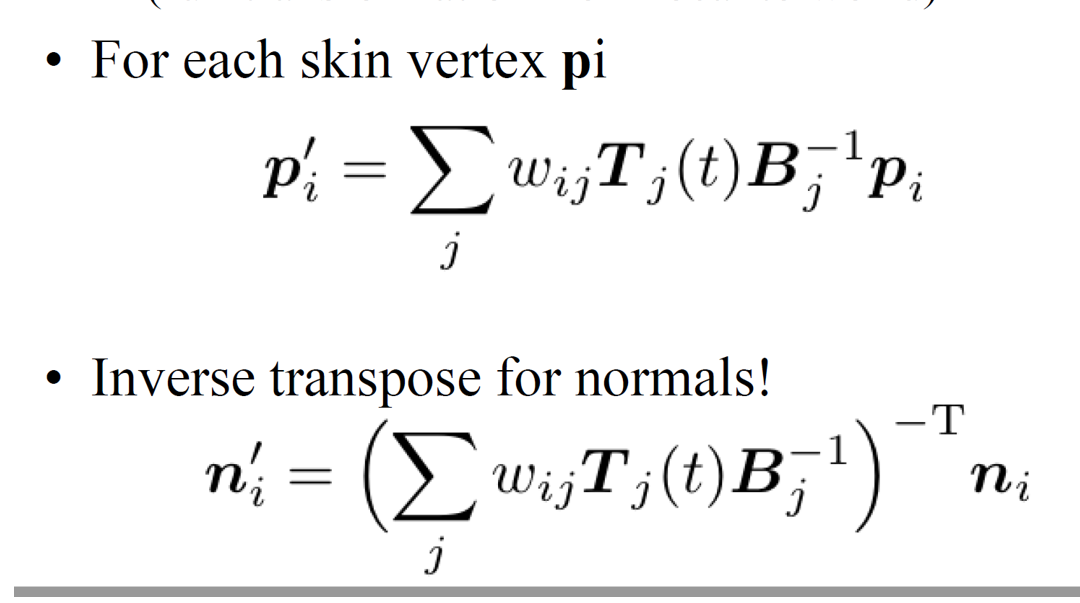
computeBindWorldToJointTransforms当骨骼每次发生改变时调用，我的实现应该更新每一个关节的currentJointToWorldTransform矩阵。再次注意空间之间的映射。

一种方便的调试方法是，如果您的骨架没有变化（即您没有触摸任何滑块），则任何“关节”的“绑定世界到关节变换”应该是当前关节到世界变换的逆。

（2）网格变形

当滑条滑动的时候调用**SkeletalModel::updateMesh()**方法，在m\_mesh.currentVertices中更新每个顶点的当前位置，通过骨骼的当前姿势。该姿势是由m\_mesh.bindVertices进行加权计算的结果；

课件的公式：



**整理总结：**

**第三方库：**

nanogui：GUI工具库 支持opengl绘制的

vecmatch：向量数学库 包括vector、matrix等

**头文件：**

相机类、opengl库文件；

1. joint.h：定义了关节

（1）每个关节有一个局部的transform转换，并有一个孩子关节的指针列表；

（2）并且，在使用蒙皮技术的时候，每个关节会存储一个bindWorldToJointTransform，即关节位置转换到世界空间的转换矩阵的逆矩阵，就是一个translation变换的逆矩阵；这个矩阵的作用是，可以通过该矩阵获取绑定姿势（皮肤）上的所有顶点相对于该关节的局部坐标；因为计算SSD的时候，每个关节影响顶点的权重，亦即partial position是在局部空间下计算的，再累加，所以需要获得顶点相对于该关节的局部坐标；

（3）同样，还需要存储一个currentJointToWorldTransform，表示该关节当下，转换到世界坐标的一个变换；也就是运动的时候，每次都要计算各个关节的世界坐标；再根据该关节的皮肤上的顶点的局部坐标，计算出皮肤的世界坐标；

2.matrixstack.h

矩阵栈，用于累计存储当前层级的变换；初始化时用的是一个Identity()单位矩阵；top()表示的是当前层级节点的变换矩阵；

3. mesh.h 网格类 其实就是存储了皮肤的数据

通过load方法加载顶点信息，将皮肤的顶点数据存储在bindVertices里面，currentVertices是运动的时候通过SSD计算出来的皮肤顶点数据；attachment是权重数据，通过LoadAttachment方法加载；

绘制皮肤仍然是通过一个Recorder辅助类来绘制，对该作业来说，是一个顶点recorder，record方法会记录顶点位置、法线、颜色等信息；并调用Opengl的绘制方法最终绘制出来；

该作业有两个特定的绘制方法，绘制球体和圆柱体，通用思想仍然是通过曲面细分，在球面上通过循环画出各个点，然后构造出三角形，记录到recorder里面从而绘制出来；

4. skeleton.h 骨骼类 核心文件

存储了m\_joints，所有个关节（指针）,m\_rootJoints，根节点，皮肤网格m\_mesh，矩阵堆栈m\_matrixstack，以及着色器program（id）；

在这个类中有很多个关键的方法，比如加载骨骼数据，绘制关节和骨骼，都是需要通过递归的方法遍历整个层级结构，计算相应的矩阵堆栈中的变换矩阵，然后对节点进行正确的变换；

Gui的交互数据，会调用setJointTransforms，实时更新节点的变换；

实现SSD有三个方法：

computeBindWorldToJointTransforms、updateCurrentJointToWorldTransforms；前者只用计算一次，后者在每次动画变换时都需要调用，并且都是需通过递归来遍历整个层级结构；还有一个updateMesh方法，是实现SSD的核心方法，会计算皮肤顶点的权重累计值；

5. starter2\_util.h

工具类，包含角度和弧度的转换方法，创建opengl窗口，编译着色器程序等，当然也包含来顶点、片段着色器的内容；

6. tuple.h C++原生tuple的一个包装，主要用于三个字段的组合，mesh中的face数据用到了；

7. vertexrecorder.h 刚才说到了 调用Opengl方法用于顶点的绘制。

额外提升：

Easy:

1 整合代码 使得可以通过存储多个骨骼、皮肤网格、和权重数据来加载多个角色；这个应该需要将骨骼定义成一个列表，然后加载和绘制的时候，都通过遍历列表来完成；

2 应用纹理映射来渲染模型；

3 这个作业使用的是每个面的法线（即每个面三个顶点的法线相同，整个面的法线也相同），为了更平滑的着色，我们可以计算每个顶点的法线，可以设置顶点的法线为周围面元的法线的平均值；（这个咋实现，如何知道一个顶点周围的面元是哪个）

~~4 用伪色来显示顶点权重，例如，为每个关节分配具有不同色调的颜色，并通过计算相应的关节颜色加权平均值，根据分配的权重为模型中的每个顶点着色。（这个就不实现了吧）~~

Medium:

使用顶点着色器实现SSD。您应该将绑定姿势存储在顶点缓冲区中，并为关节变换定义一组统一变量。对于每一帧，上传最新的关节变换，并在着色器中计算变换后的顶点位置和法线。

问题：

如何在着色器里面计算法线？

原来的计算方法，是计算出当前顶点的世界坐标，然后用两条边叉乘计算出三角形的法线（世界空间），现在只能传递一个绑定姿势的数据，即每一个顶点的固定世界坐标；

传递进来的都是皮肤顶点在世界坐标下的顶点位置和顶点法线；主要还是那个公式计算；

坐标直接按公式算，法线需要转换矩阵的转置再取逆；

如何传递权重值以及关节的变换值？

需要传递三个Uniforms，一个是bindWorldToJointTransform数组，一个是currentJointToWorldTransform数组；还有一个是权重值attachments数组；

如何确定当前着色器正在绘制的顶点是哪一个顶点？？这是关键的问题，因为如果不知道是哪个顶点（顶点下标），就无法获得对应的权重列表，也就无法得到各个关节的权重值，从而计算顶点的位置和法线；

还要解决哪个权重vecn无法存储的问题；