## 编程作业2 ： 层次模型和骨架子空间变换(SSD)

在本作业中，需要构造层次角色模型并能与用户接口交互和控制。层次模型可以表示人形角色（如普通人类，机器人和外星人），动物（狗，猫，蜘蛛），机械装置(手表，三轮车)等。你需要实现骨架子空间变换，在层次骨架上的简单蒙皮，并且当转动骨架的关节角时蒙皮会自然的变形。

本文档由下面几个部分组成：

1. 如何开始
2. 需求摘要
3. 层次模型
4. 骨架子空间变换
5. 附加分
6. 如何提交作业

# 1 如何开始

下载所提供的代码及工程，创建执行文件，并且运行所产生的执行文件，使用第一个模型作为命令行参数如下：

```
LINUX:     ./a2 data/Model1

WINDOWS:     a2 data/Model1
```

将弹出二个窗口。一个是空的，并将最终包含对你的角色的渲染。另一个包含一些关节变量或者简单的就是关节的列表。通过点击关节名，将出现滑块让你操作该变量。通过按住shift或者control选中多个变量名，将弹出多个滑块。可以鼠标来控制相机：左键旋转，中键移动，右键缩放。可以按下a来切换是否画坐标轴。

a2soln文件中我们给出了做好的作业示例，包括装入并显示骨架，装入包围骨架的网格,根据关节角变换骨架和网格。可能按下s来切换显示骨架还是网格。

# 2 需求说明

## 2.1 层次模型 (40分)

本作业的第一部分，要求大家正确装入、显示并操作层次骨架。你的实现应该能够正确的解析所提供任意骨架文件(*.skel)，构造关节层次，并使用矩阵栈和OpenGL基本图元实现骨架渲染。最后，你需要编写代码来设置关节变换，变换所需要的关节角由用户接口提供。

## 2.2 骨架子空间变换(55分)

本作业的第二部分，你需要实现骨架子空间变换来对骨架蒙皮。SSD（骨架子空间变换）让你能够产生真实的角色，而不仅仅只有骨架。本部分要求大家把作业0的代码再实现一次，只是这里的网格没有法向量会在显示时自动生成。（This part first requires you to adapt your assignment 0 code to parse a mesh without normals and generate them at display time）。还需要编写另外一个解析器来装入附带的权重，该权重将对每个节点指定关节的权重。最后，你将实现真正的SSD算法，该算法将对变换层次应用一组变换。

### 2.3 艺术品？技巧？ (5分)

本次作业的艺术品（artifact）很好制作：只用截屏某个角色，并保存为PNG, JPEG或GIF格式。 可以从用户接口的文件(file)菜单中选择“Save bitmap file”。请花几分钟来摆放角色并选择合适的照相机位置。一个直观的扩展可以是装入多个角色，并以有趣的形式将其摆到一起，你也可能想要画一个扁平的长方体的方式来增加一个地板。

## 3 层次建模

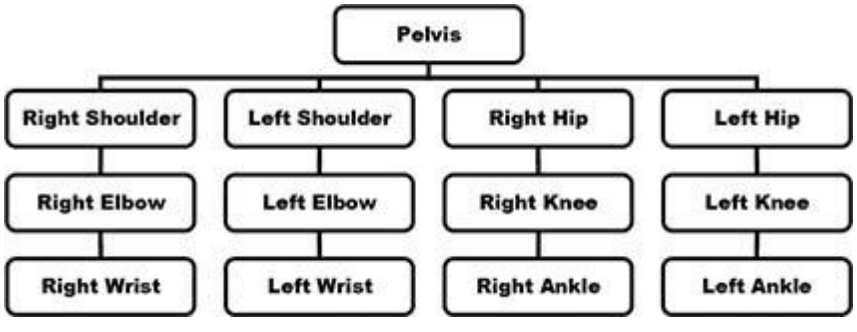
在作业一中，我们实现了静态几何模型的生成。正如大家所做到的，作业一很好地产生了如球面、茶壶、玻璃杯、雕像等，但其有些限制，包括给角色摆个POSE或者让角色动起来。例如在游戏中，人体模型需要与真实环境交互，并通过移动枝体来模仿行走或奔跑。

产生这样动画的一种方法是分别操作结点和控制点。快速完成此工作非常繁琐。一种更好的方法是定义一个层次结构（如人体角色的骨架）,同时给定少量控制点，如骨架的关节角。通过操作这些参数（有时称为关节变量或关节），用户可以很容易的摆放层次形状。进一步，物体表面也可由同样的关节参数计算出来。人类角色骨架的一个示例见下图。

骨架的每个关节都与变换相关联，这些变换定义了相对父结点的局部坐标。这些变换典型的有平移和旋转。一般而言，仅旋转变量由用户操作的关节变量控制(改变平移变量意味着拉伸骨头)。可以通过沿树向下乘以局部变换来确定结点的全局坐标框架 (即相对世界的坐标系统)。

每个结点的全局坐标架构可以用于产生角色模型，方法是对每个关节实施几何变换。例如，角色的躯干可在骨盆（pelvis）坐标系下画出来。而角色的大腿(thighs)可在臀部（hips）坐标系下画出。请确保自己明白这些全局坐标框架的含义，输入在什么坐标系？输出又在什么坐标系？

在你的代码中，模型应以此种方式来画。譬如通过在左臀部画一个圆柱体，就可以画出角色的大




腿。在层次结构中对所有结点这样操作将产生一个简单粘连的人物形状，如图所示。

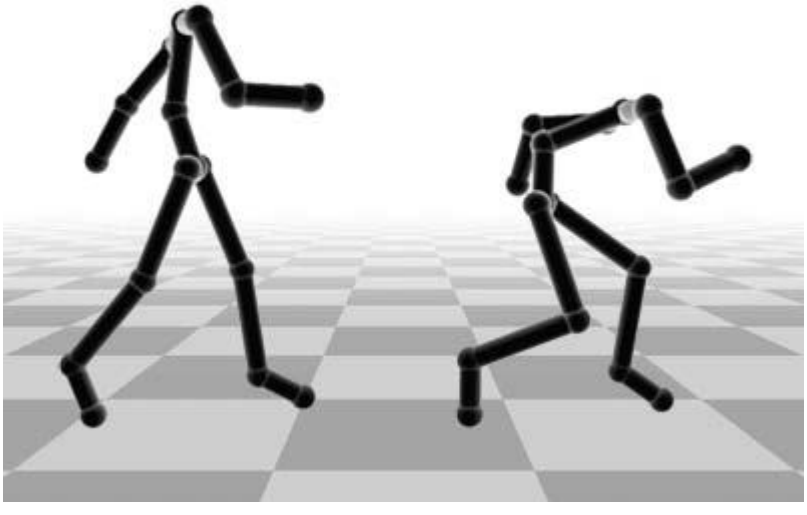
### 3.1 矩阵栈(5分)

第一个任务是实现矩阵栈。矩阵栈会跟踪当前的变换(编码在矩阵中),当渲染时矩阵栈会应用到几何物体上。它会存储在栈中，允许你维护坐标架构的层次，其由相对于另一个来定义。如脚的坐标框架相对于腿的坐标框架来定义。

OpenGL提供了维护矩阵栈的框架。但在本作业中，你需要构建自己的，而不是使用OpenGL的。构建自己的矩阵栈，将能拥有更灵活的数据结构而不受渲染系统的影响。 例如，可以同时维护多层次角色，并对他们进行碰撞检测。

在你的实现中，如果没有当前变换应用到栈中，那么应返回不变。压入栈中变换应该与之前的变换相乘。这将把你放到放到与其父结点相关联的正确坐标系中。文件MatrixStack.h中已经给大家定义好矩阵栈的接口。文件MatrixStack.cpp的实现目前是空的，需要你来完成。推荐大家使用使用STL向量来实现栈，当然具体使用什么，你想怎么样就怎么样。

渲染时，需要压、出栈。在每次压出栈后，你应调用glLoadMatrixf( m matrixStack.top() )



来告诉OpenGL你希望所有的几何图元由当前栈顶矩阵来变换。其后的OpenGL基本图元(glVertex3f, glutSolidCube等)都会被此矩阵所变换。起始代码的SkeletalModel类包含MatrixStack的一个实例称为m matrixStack。起始代码也将相机（camera）矩阵作为栈中的首项压入栈中。

## 3.2 层次骨架 (30分)

### 3.2.1 输入文件

你的下一个任务是解析骨架（skeleton）。初始代码自动调用方法SkeletalModel::loadSkeleton 使用正确的文件名（filename）(可在SkeletalModel.cpp中找到)。骨架文件(.skel) 格式是直观的。它包含一些文本行，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)，因此形成一个关节结点的有向无环图。根结点以-1作为其父结点，其变换是角色在世界中全局位置。

.skel文件的每一行表示一个关节，你应该以指针方式来装入，该指针指向Joint类的一个新 实例。可以这样来初始化一个新的关节

```
Joint *joint = new Joint;
```

因为Joint是一个指针，注意我们必须使用‘new’关键词来初始化指针，以在内存中为此对象分配空间,该对象直到函数结束前会持久存在。(如果你试图给局部变量创建一个指针，当局部变量不在范围内时，指针将会无效，而此时访问它将导致崩溃)。同样注意当处理指向对象的指针时，访问对象的成员变量应使用箭头算符“->” 而不是“.”(例如joint->transform)，表示涉及到需要查找内存。

loadSkeleton的实现必须创建Joints层次结构，其中每个关节（Joint）维护一组关节（Joints）指针，这组关节是其子结点。还必须填充SkeletalModel中所有关节m_joints的列表，并设置m_rootJointr指向根关节。(You must also populate a list of all Joints m_joints in the SkeletalModel and set m_rootJoint to point to the root Joint.)

### 3.2.2 画杆状图形

为确保骨架装入正确,需要画出简单的杆状图，如上图所示。

**Joints**：首先需要在每个关节处画一个球，以观察骨架的总体形状。初始代码调用SkeletalModel::drawJoints. 你的任务是创建一个专门的递归函数，该函数在drawJoints中调用，其功能是从根出发遍历关节层次，并使用你的矩阵栈在每个关节处画一个球。推荐使用glutSolidSphere( 0.025f, 12, 12 )来画合适大小的球。

必须使用你的矩阵栈来进行变换。如果使用OpenGL的矩阵栈将不会得分。在实现时，必须将关节的变换压栈，调用glLoadMatrixf装入变换,递归画出其任意子关节，然后弹出栈。通过与示例解答对比来检查渲染效果，可能会有帮助。

**Bones**：缺乏骨骼的杆状模型不是很好玩。为了画骨骼，需要在方法SkeletalModel::drawSkeleton中，在每对关节之间画出拉长的盒子。正如joints一样,需要你自己定义一个专门的递归函数，该函数将遍历关节层次。在每个关节，画一个盒子，该盒子连接关节和其父关节(除非其是根结点，此时不用做任何处理).

不幸的是，OpenGL的盒子基本图元glutSolidCube只能画中心位于原点立方体；为此，我们推荐下面方法。从边长为1的立方体开始(简单调用glutSolidCube( 1.0f ))。沿z平移使盒子范围从$[-0.5, -0.5, 0]^T$ 到 $[0.5, 0.5, 1]^T$ 。进行缩放变换，将盒子范围变换到从$[-0.025, -0.025, 0]^T$ 到

$[0.025, 0.025, L]^T$ 其中L是在递归中到下一关节的距离。最后，需要旋转z轴，使得其与父关节的方向对齐：z = parentOffset.normalized(). 由于x和y轴任意，推荐这样映射y =(z × rnd).normalized(), 而 x =(y × z).normalized(), 其中rnd 为 $[0, 0, 1]^T$ .

对于盒子基本图元的平移、缩放和旋转,必须在调用glutSolidCube前先将变换压栈,但在画其子结点前必须将其出栈，因为这些变换并不是骨架层次结构的一部分。和之前画joints一样,可以对比示例程序来验证你的实现的正确性。

## 3.3 用户接口(5分)

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。你需要实现此函数来正确设置关节变换矩阵的旋转组分。

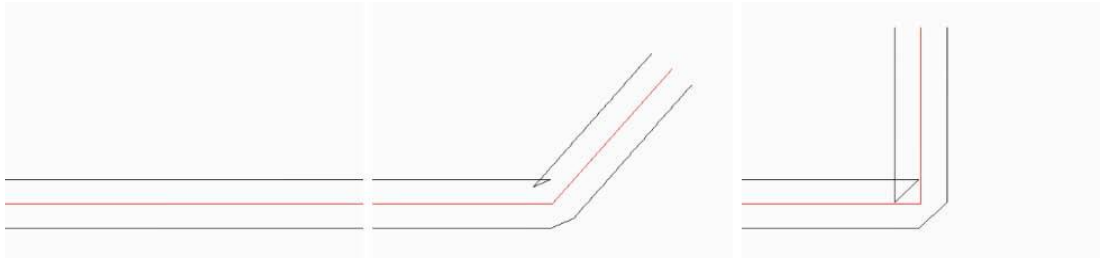
# 4 骨架子空间变换（Skeletal Subspace Deformation)

层次骨架让你在3D中渲染和摆放木棒做的人的模样。本节，我们将使用骨骼子空间变换来在骨架上附加一个自然变形的网格。

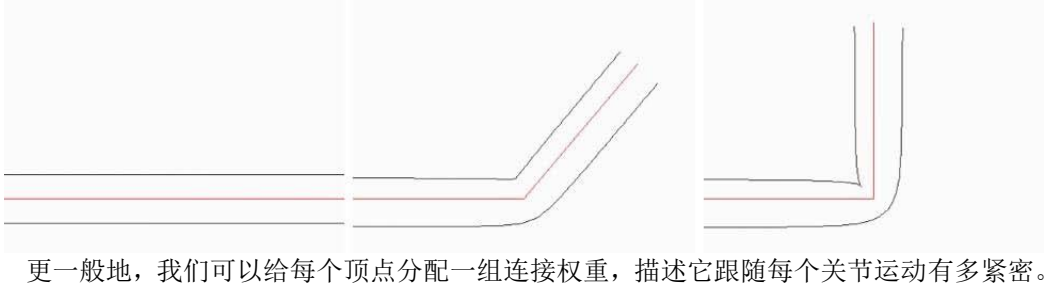
在渲染骨骼的方法中，身体部位（球体和立方体）被绘制在单个关节的坐标系中。然而，这种方法可以产生一些不期望的错误结果。观察关节附近的顶点。

这是骨架的某个剖面视图，其网格连接到单个节点。可以发现，当骨架弯曲时，两个网格是如何相互碰撞和交织。前面实现的棒状人形角色通过在每个关节上画球隐藏了此错误。然而，对于上

述方法将人形模型的顶点刚性地附加到层次结构的各个节点这一事实，这只是一个快速修复。对于更复杂的生物角色（如人和动物），这并不合适，因为皮肤不牢固地附着在骨头上。相反，它根据骨骼的状态而平滑变形，如下所示。



下图结果是使用骨架子空间变形(SSD)实现的，SSD 将各个顶点与邻近关节相关联,并通过关节变换的加权平均来求顶点的位置。例如，通过平均与肩关节和肘关节相关的变换来定位模型肘关节附近的顶点。靠近骨骼中部（远离关节处）的顶点仅受单个关节的影响,它们刚性地移动，就像在上图中的状态设置中那样。



更一般地，我们可以给每个顶点分配一组连接权重，描述它跟随每个关节运动有多紧密。若顶点对某个关节权重为1，该顶点将完全随该关节运动，就像前图一样。关节的权重为零的顶点完全不受关节的影响。其他中间顶点需要混合——先按结点固定在关节上计算其位置，再根据所赋权重对这些位置进行平均。

在上一节中，顶点定义在给定关节的局部坐标中,你可能使用glutSolidCube之类的方法画关节，然后通过平移，将关节移到对象所在位置。现在,结点不在属于单个关节，因此不能在所属的关节局部坐标框架中定义结点。相反，我们对整个人体定义网格，并且跟踪绑定姿势。人体每个关节的绑定姿势，需要与网格中顶点的位置匹配。想象先获得角色的表皮，然后在表皮里安装骨架。与皮肤位置相匹配的骨架姿态（各个关节的角度）即是角色的绑定姿态。

假设P是绑定姿态下，角色坐标系中顶点的位置。P受关节1和关节2的影响。我们还假设关节1的绑定姿态变换是B1,该变换实现从关节1的局部坐标（动画前的）变到角色坐标系。最后，从关节1的局部坐标系到角色坐标系的动画转换为T1。然后变换后的结点位置,如果该顶点刚好连接到关节1，将是 $T_1B_1^{-1}p$ 。注意，在变换它之前，我们必须先将点变换成关节的局部坐标系（$B^{-1}p$）（记住，p在角色的绑定姿态坐标系中）。类似地，关节2的绑定变换由$B_2$描述，T2描述从关节2的非动画局部坐标系到动画角色坐标系的转换。如果顶点刚性地连接到关节2，其位置将是 $T_2B_2^{-1}p$ 。但是，如果给定的顶点靠近关节1和关节2的两个骨骼的连接处，我们希望它连接到两个关节。我们根据关节对结点的影响程度来给每个关节分配权重。每个关节的权重量通常在0到1之间，所有关节的权重和通常为1。我们希望它与关节1的关联权重为W，所以它与关节2关联权重为1-w。最后计算结点的最终位置为 $wT_1B_1^{-1}p+(1-w)T_2B_2^{-1}p$

注意，由于我们通常对某个角色仅有一个绑定姿态，所以反向绑定变换$Bi^{-1}$只需计算一次。另一方面，由于我们想要使用用户界面实现角色动画，所以每次关节角度改变时，都需要重新计算动画变换Ti。这意味着一旦骨架改变,结点位置也需要更新。（虽然对仅有少量关节的角色重新计算Ti计算机量相对很少，但是更新整个网格可能相当费时。现代游戏通常使用图形硬件在许多结点上并行执行SSD。

## 4.1 文件输入： 绑定姿态网格(5分)

首先可先修改作业0的代码，以适应能从OBJ文件中装入绑定姿态顶点。所提供的初始代码会用合适的文件名自动调用Mesh::load。本部分和作业0之间唯一的区别是所提供的网格不包括法向。相反，会在渲染时实时生成法向。你的代码应填充网格的bindVertices和faces字段。请注意，我们的Mesh结构体有两个结点：绑定姿态和当前姿态。我们将对当前姿态顶点渲染，这些顶点是由绑定姿态顶点变换得到的。初始代码为你产生原始版本。

## 4.2 网格渲染(5分)

接下来，我们将通过渲染网格来验证网格加载程序的正确性。初始代码用正确的文件名自动调用Mesh::draw。一定要用m_mesh.currentVertices，而不是m_mesh.bindVertices来渲染。

与以前作业中的网格不同，这些网格不提供每个顶点的法向，因为它们不是解析计算得到的。取而代之，我们将对每个三角形动态生成一个法线,方法是在渲染循环中对边求叉积。别忘了规范化你的法向。注意你的模型是如何“以面的方式出现”的：由于法线突然改变，相邻面之间的光照是不连续的。

## 4.3 文件输入: 附加权重 (5分)

最后要加载的是“联接权重”。初始代码自动用正确的文件名调用Mesh::loadAttachments。"联接”文件格式（.attach）很简单。它包含多行文本，每行是在网络中的一个结点。每一行包含所有关节，-1，字段用空格分隔。每个字段是一个浮点数，表示结点附着到第（i+1）个关节的强度。第0个关节(根关节)的权重，假定为零。

你的代码应该填充网格的attachments字段。我们推荐使用初始代码的数据结构，其中m_mesh.attachments是vector< vector< float > >数据类型。内部向量包含每个关节的一个权重，而外部向量的大小等于结点的数量。

## 4.4 实现SSD (40% of grade)

最后，将实现SSD。首先计算混合权重所需的所有变换，然后使用它们更新网格的结点。

### 4.4.1 计算变换

如上所述，我们必须计算绑定姿态世界到关节变换（一次）并且对世界变换来实现姿态角动画（每次骨架改变时）。初始代码在代码中适当的位置自动调用computeBindWorldToJointTransforms和updateCurrentJointToWorldTransforms。

CopyWorktotoJooTimeTrimes应设置每个关Joint的ButWordtoJoin变换矩阵。你应使用类似于渲染骨架的递归算法。小心矩阵乘法的顺序。仔细考虑哪个空间是输入，哪个空间是输出。

每当骨架改变时，updateCurrentJointToWorldTransforms被调用。您的实现应该更新每个Joint的currentJointToWorldTransform矩阵，此实现与computeBindWorldtoJointTransforms非常类似。于您的computeBindWorldtoJointTransforms的实现。但是同样，要小心你些哪些空间之间映射。一个方便的调试方法是，如果骨架没有改变（即，没有动任何滑块），则任何Joint的关节变换绑定世界应该是当前关节世界变换的逆。

### 4.4.2 变换网格

对于作业的最后一部分，你将根据骨架和附加权重变形网格。由于填充了所有适当的数据结构，所以你的实现应该是直接的。每当滑块改变时，初始代码会调用SkeletalModel::updateMesh。你的代码应该根据骨架的当前姿态来更新m_mesh.currentVertices中每个结点的当前位置，方法是混合在m_mesh.bindVertices中绑定位置结点位置的变换。

如果正确实现SSD，则你的解决方案应与示例解一致。您可以随意更改角色的外观，并扩展代码以将多个角色摆放在一起构成一个有趣的场景。

# 5 附加分

与以前的作业一样，这个作业的附加分也将被列为容易、中等和困难。这些分类只不过是对它们价值的粗略估计。实际分值将取决于实现的质量。例如，一个中等难度的不好实现可能比一个简单的很好实现得分低。我们努力做到公正。

## 5.1 容易

- 通过存储多个骨架、网格和附件权重使你的代码能处理多个角色，从而更通用。你还可选地实现，在相同角色的多个实例间重用数据。
- 使用OpenGL纹理映射来更有趣地渲染模型的各部分。OpenGL红宝书的第9章有些内容，它提供了大量可以随意使用的示例代码。
- 使用OpenGL模拟阴影或地板上的模型反射。尽管，通用地实现这些操作是相当复杂的，但是当您只是假设平面接收阴影时，就比较容易了。OpenGL红皮书第14章中描述了实现此点的一种方法。
- 对于SSD的实现，使用伪色彩显示结点权重。例如，给每个关节分配具有不同色调的颜色，并且根据指定的权重计算相应的关节颜色的加权平均,来模型中的每个顶点着色。
- 有很多技巧可以让模型看起来更有趣。请放飞你的思维，实现我们没有列出的扩展，我们会给你一个适当附加分。

## 5.2 中等难度

- 用通用圆柱体和旋转面建立自己的模型，并用SSD来摆放。
- 实现通过模型显示来直观的操纵关节变量。例如，如果肘关节是活动的，用户应能点击arm并拖拉它来设置角度（而不是使用滑块）。如果想看此接口的示例，可看看MAYA。
- 通过使用插值样条来控制关节变量，来实现动画函数。此方法被称为关键帧。可以允许来自文本文件的输入，也可以实现某种接口，允许用户交互地修改曲线。

## 5.3 困难

- 实现姿态空间变形。此方法是可替代SSD，通常能得到更高质量的结果。
- 实现逆运动学，它可求得给定位置约束下的关节角度。例如，当拖动模型的手时，肘部自然伸展。代码应允许通过绘图窗口对模型进行交互操作。
- 实现基于网格的逆运动学，它允许模型在没有任何基础骨架的情形下摆一个姿势。

# 6 提交指示

和之前的作业一样，需要编写README.txt,pdf以回答下面问题:

- 如何编译和运行你的代码?
- 是否与其他同学讨论与合作？如果是，请告诉我们你与谁交流过，以及你收到或者给予了什么样的帮助。
- 是否有特别有助于你完成作业的参考资料（书籍，论文，网站等）？请提供一份清单。
- 代码有何已知问题吗？如果有，请提供一份列表，可能的话，描述一下你认为原因是什么，以及如果给你更多的时间，你怎样改进。这非常重要，因为如果你帮助老师理解所发生的事情，我们很有可能给更多附加分。
- 你是否做了附加题？如果是，说明如何使用你做的附加分程序。如果涉及大量的工作，请描述你做了什么，并且如何做的。
- 对此作业,你想分享些什么意见？

以单个压缩文件提交你的作业(.tar.gz or .zip).包括:

- 源码
- 执行程序
- 必要的附加文件 (OBJs, 纹理)
- README文件
- PNG, JPEG,GIF格式的结果图像