Homework 10 作业报告 by 76-朱雨田

本次作业要求运用变换矩阵实现角色骨骼动画，并在其上添加布料仿真与人体碰撞的效果。

我实现了作业中的所有要求，实现效果见附录中的 `./gif/arm.gif`，`./gif/belly_dance_girl.gif`，`./gif/arm_with_cloth.gif`，`./gir/belly_dance_girl_with_skirt.gif`。

这次作业是计算机图形学的最后一个小作业了。因为相当多课程的 DDL 任务迫近，本作业完成的实在比较草率，在临近 ddl 的时候才赶出这份报告。

我认为计算机角色动画的学习更应该结合着实际应用侧的游戏引擎、动画软件一起学习，这样才能充分理解其中每个部分的实际意义，也有利于思考"我们接下来该做什么"。在图形学课程结课后，我希望我能去更深入地研究 Unity、Unreal、Blender 等图形学应用侧的软件，做一些漂亮的东西出来，也将图形学课程中学到的知识落地，弄明白我们学到的东西用在了哪里，我们接下来该学什么、该做什么。实际上图形学大作业存在的意义似乎也是让我们更多地接触图形学在工业界的实际应用？

概念理解

变换

为了同时处理平移、旋转、缩放三种变换组合成的仿射变换，计算机图形学中常常用四维的齐次坐标系来表示三维空间的变换，即：将平移、旋转、缩放三种变换都整合进一个四维矩阵中。

这就是 Unity 等游戏引擎中常见的 Transform 组件。其实在接触线性代数前，我对"Transform"的理解是：**描述物体在某一坐标系中位置、旋转、缩放值的一系列物理量。**现在看来这样的描述也是很优美的：当描述清楚物体的位置、旋转、缩放共九个自由变量时，我们就能很轻易地确定一个矩阵，将这一矩阵左乘物体本身在预制下相对物体原点的顶点位置的齐次坐标，就能得到这些顶点的世界坐标，从而真正地对物体做到变换。

用四维矩阵表示物体在坐标系中的参数还有一个优美之处，就是这样的变换是可以复合的。下文中的父子关系就是对这一复合特性的经典运用。

父子关系

计算机动画中常要处理关节型的结构。例如，手与手臂存在位置约束的关系，当手臂旋转时，手也会跟着旋转；同时手还存在自己独立的运动。在绝对坐标系下处理手部自身的运动对动画师来说是很不自然的——更自然的思维是， 在腕部固定的局部坐标系下处理手部的动作。这样，无论身体怎样运动、手臂怎样旋转，同一套手部运动总是可以适用的。

这引出了计算机图形学中常见的父子绑定思想。将手臂作为父对象，将手作为子对象，处理手部的运动时只需在手臂固定的坐标系下进行就行。这样，我们将手部的变换用局部矩阵 $\mathbf{L}$ 表示，将手臂在世界坐标系下的变换用世界矩阵 $\mathbf{W}_{\text{Parent}}$ 表示，可以如下计算得到手部的世界矩阵：

$$\mathbf{W} = \mathbf{W}_{\text{Parent}}\mathbf{L}$$

对于顶点集合 $\{\vec{x}, \dots\}$ 组成的物体，对顶点的变换 $\vec{x}' = \mathbf{W}\vec{x} = \mathbf{W}_{\text{Parent}}(\mathbf{L}\vec{x})$ 可以理解如下：

1. 先做 $\vec{x}_1 = \mathbf{L}\vec{x}$ 变换，在腕部固定的坐标系下，将手部所有的顶点位置做变换，得到新的顶点位置集合 $\{\vec{x}_1, \dots\}$。这一系列顶点是腕部固定的坐标系下手部"做完动作"后得到的物体。

2. 再做 $\vec{x}' = \mathbf{W}_{\text{Parent}}\vec{x}_1$ 变换。这一变换可以理解为将物体 $\{\vec{x}_1, \dots\}$ 组装到父对象所在的世界位置，也就是从局部坐标系向世界坐标系变换。

需要注意，在计算子物体的世界坐标前，父节点的世界变换应该已经确定。好在节点间的父子关系构成的是一棵树，我们可以从根节点开始更新每个节点的世界变换。在框架中，枚举 `joints_` 时得到的就能保证是从根节点向下枚举，因此不必太多考虑这一点，但在自己编写相关程序的时候应务必注意。

骨骼与蒙皮

父子关系的处理方式中，每个节点的顶点位置都是固定的，节点所关联的一系列顶点组成的物体本身只能做相似变换。这还不足以满足动画师的需求。动画师们希望顶点的位置可以由多个关节确定。这需要改进“节点”“子部件”的概念，建立“骨骼”与“蒙皮”的新概念。

在骨骼动画中，“子部件”被简化为最简单的“骨骼”，顶点被简化为只剩下一系列互相关联的关联点；剩余的非关节的顶点的位置则更为自由，不再完全受到子部件的约束，而是可以由多个关联的骨骼关节加权求和确定位置。表示为公式则是：

$$\tilde{\mathbf{x}} = \sum_{i}^{n} w_i \mathbf{T}_i \mathbf{B}_i^{-1} \tilde{\mathbf{x}}^0$$

其中 $\mathbf{T}_i$ 为对应关节的世界坐标，$\mathbf{B}_i$ 为“绑定矩阵”，实际上是预设姿态（“T-pose”...）下顶点从关节局部坐标向世界坐标的变换矩阵。与”父子关系“中的理解类似地，$\tilde{\mathbf{x}}^0$ 是顶点在预设姿态下的世界坐标，$\mathbf{B}_i^{-1}\tilde{\mathbf{x}}^0$ 是顶点在预设姿态下关于该顶点的局部坐标，左乘上 $\mathbf{T}_i$ 就能得到将部件”组装“到关节上的效果。在这一基础上对一系列关节加权求和，也就是**对一系列运动做插值**，就能得到受多个关节约束的运动了。

`usda` 文件格式中对”一系列关节“的描述方法是先确定整个模型中顶点最多的约束数，再将所有相关量组成一个一维数组，而不是对每个顶点分别用一个数组描述。这点我在检查了 usda 源文件后才得知，也因此调了很久的 bug。

实现细节

1. OpenUSD 对矩阵乘法的实现存在一个转置关系，因此在计算矩阵相乘时可能会存在不合常规逻辑的现象。这一问题我也调了非常久，在没弄懂 OpenUSD 到底是怎样做矩阵乘法时，我是先将矩阵转换为 Eigen 矩阵，在 Eigen 中按正常逻辑进行左右乘，再转换回 OpenUSD 矩阵的。

```
Eigen::Matrix4f matp = Matrix4f::Zero(), matl = Matrix4f::Zero();
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
    {
        matp(i, j) = parent_->world_transform_[j][i];
        matl(i, j) = local_transform_[j][i];
    }
auto matw = matp * matl;
world_transform_ = GfMatrix4f();
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
        world_transform_[j][i] = matw(i, j);
```

2. 在加上布料仿真的时候，我这里需要修改原有的弹簧质点系统，将写死的固定点解除，并在 `update_dirichlet_bc_vertices()` 函数中也更新一次选择矩阵。

3. 碰撞处理采用的是最简单的加球体的方案。具体流程为枚举所有固定点得到中心；将中心到所有固定点距离的平均值的 2.5 倍作为球体半径；将中心向下移动一个半径得到的位置作为球心。在每个 `step()` 均更新一次球心与半径的位置。2.5 、向下移动一个半径等数值均是试出来的，在交报告时仍有较明显的穿模现象。若要解决穿模现象，我猜测一个可能的方案是需要通过网格法线来确定惩罚力。用球体约束只是很朴素、草率的处理方式，更多的物理约束方式值得在计算机角色动画相关专门课程中进行学习。

```cpp
// only for "belly_dance_girl" model
int fix_n = 0;
double fix_rad = 0;
Vector3d fix_center = Vector3d::Zero();
for (int i = 0; i < n_vertices; i++)
{
    if (dirichlet_bc_mask[i])
    {
        fix_center += X.row(i).transpose();
        fix_n++;
    }
}
fix_center /= fix_n;
for (int i = 0; i < n_vertices; i++)
{
    if (dirichlet_bc_mask[i])
    {
        fix_rad += (X.row(i).transpose() - fix_center).norm();
    }
}
fix_rad /= fix_n;
fix_rad *= 2.5;
fix_center += Vector3d(0, 0, -fix_rad);
sphere_center = fix_center.cast<float>();
sphere_radius = fix_rad;
// std::cout << "center:\n" << sphere_center << std::endl;
// std::cout << "radius: " << sphere_radius << std::endl;
```