HW9: 基于C++四元数类的GrassHopper动画制作

> 85
>
> PB15111679
>
> 庄涛
>
> 2018/04/28

0. 要求

1. 使用`C++`实现给定物体（例如Mesh）的两个不同空间姿态(pose)间的平滑过渡（效果参考：作业9-`demo.gif`，具体`C#`与`C++`数据接口部分参考作业8相关资料）；
2. 使用四元数平滑过渡解决旋转问题；

- 参考作业4给的`MiniMeshFrame`框架中的`ArcBall`类（四元数类）；
- 了解和实现四元数插值类；
- 通过百度或google能找到很多有关四元数与旋转的有关技术论文；

3. 利用`GrassHopper`制作并导出动画 **（gif/MP4）**。

1. 四元数类`Quat`

```
class Quat
{
public:
    double x, y, z, w;

    Quat();
    Quat(double _x, double _y, double _z, double _w);
    Quat(Vector3d norm, double theta);
    ~Quat();

    inline Vector3d norm(){/*...*/}
    inline static Quat InValid(){/*...*/};
    inline double theta(){/*...*/}

    Quat operator ()(double t);
    Quat & operator =(Quat & q);
};
```

四元数插值就是对`theta`进行线性插值。

这个类只做了一些简单的方法，我把核心的函数放在了另一个类`Transform`中。

2. 变换类 `Transform`

```
class Transform
{
public:
    Transform();
    Transform(Quat & q);
    Transform(Quat & q, Vector4d & T);
    Transform(TF_M & t);
    Transform(vector<Vector4d> & p, vector<Vector4d> & Ap);
    ~Transform();

    Transform R();
    Vector4d T();
    Quat Q();
    TF_M & M();

    void setR(Quat & q);
    void setT(Vector4d T);
    bool set(vector<Vector4d> & p, vector<Vector4d> & Ap);

    Transform operator ()(double t);
    Vector4d operator *(Vector4d & p);

    static Transform Invalid();
private:
    TF_M _M;
    static void gauss(MatrixXd & m);
    static Vector4d solve(MatrixXd & m, double sign);
};
```

这里有4个核心函数，如下

函数	说明
void setR(Quat & q)	将四元数转换为旋转矩阵
`Quat Q()`	获取变换中的旋转矩阵部分，并将其转化为四元数。
`bool set(vector<Vector4d> &p, vector<Vector4d> &Ap)`	通过3个点求得变换矩阵，最关键部分。用到了私有成员函数`void gauss(MatrixXd & m)`和`Vector4d solve(MatrixXd & m, double sign)`
`Transform operator ()(double t)`	变换矩阵插值

2.1 旋转矩阵与四元数

两者的关系如下

$$R_{left} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2+q_2^2-q_1^2-q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_0q_2+q_1q_3) & 2(q_2q_3-q_0q_1) & q_0^2+q_3^2-q_1^2-q_2^2 \end{bmatrix}$$

由此可得

$$\begin{cases} |q_0| = \frac{1}{2}\sqrt{1+T_{11}+T_{22}+T_{33}} \\ |q_1| = \frac{1}{2}\sqrt{1+T_{11}-T_{22}-T_{33}} \\ |q_2| = \frac{1}{2}\sqrt{1-T_{11}+T_{22}-T_{33}} \\ |q_3| = \frac{1}{2}\sqrt{1-T_{11}-T_{22}+T_{33}} \end{cases}$$

$$\begin{cases} sign(q_1) = sign(q_0)sign(T_{23}-T_{32}) \\ sign(q_2) = sign(q_0)sign(T_{31}-T_{13}) \\ sign(q_3) = sign(q_0)sign(T_{12}-T_{21}) \end{cases}$$

根据上述公式即可进行旋转矩阵与四元数之间的转换。

2.2 变换矩阵插值

这次作业涉及的变换是**旋转+平移**，因此插值就只考虑这两者。

- **平移**的插值是简单的线性插值。
- **旋转**的插值是将旋转矩阵转换为四元数，对四元数进行插值，再将其转换为旋转矩阵。

2.3 求变换矩阵

2.3.1 求法

这次实验最难的部分就是**求变换矩阵**。理论上只需3对不共线的点就可以确定一个变换矩阵。

设变换矩阵为

$$TF=\begin{bmatrix}A&B&C&D\\E&F&G&H\\I&J&K&L\\0&0&0&1\end{bmatrix}=\begin{bmatrix}A&B&C&0\\E&F&G&0\\I&J&K&0\\0&0&0&1\end{bmatrix}*\begin{bmatrix}1&0&0&D\\0&1&0&H\\0&0&1&L\\0&0&0&1\end{bmatrix}=R*T$$

设三对点为

$$\begin{aligned}p_1&=(px_1,py_1,pz_1,1),Ap_1=(Apx_1,Apy_2,Apz_3,1)\\p_2&=(px_2,py_2,pz_2,1),Ap_2=(Apx_2,Apy_2,Apz_2,1)\\p_3&=(px_3,py_3,pz_3,1),Ap_3=(Apx_3,Apy_3,Apz_3,1)\end{aligned}$$

则满足的关系是

$$\begin{cases}TF*\begin{bmatrix}p_1^T&p_2^T&p_3^T\end{bmatrix}_{4X3}=\begin{bmatrix}Ap_1^T&Ap_2^T&Ap_3^T\end{bmatrix}_{4X3}\\R^T*R=I\\|R|=1\end{cases}$$

12个未知量，式1只有9个不共线的等式。如果加入一个不共面的点，那就有12个不共线的等式，可以求解了。

但问题是，有些mesh就只有3个点，更常见的是有些mesh所有点都共面，那就没法得到额外的一个不共面的点了。因此只能通过新增额外的限制条件。

旋转矩阵R是正交矩阵，故满足式2，旋转矩阵的行列式为1，即式3。

但以上公式组合起来并不是线性方程组，难以求解，需要特殊处理。

我的求法如下

$$\begin{cases}\begin{bmatrix}A&B&C&D\end{bmatrix}*\begin{bmatrix}p_1^T&p_2^T&p_3^T\end{bmatrix}_{4X3}=\begin{bmatrix}Apx_1&Apx_2&Apx_3\end{bmatrix}_{4X3}\\A^2+B^2+C^2=1\end{cases}$$

式1求得A、B、C与D的关系，代入式2求得D，处理的是一元二次方程，存在符号选择，两个根不同，通过尝试的方法可知道正确的根。得到D后可求得A、B、C

同样的方法可求得E、F、G、H和I、J、K、L

最后验证

$$\begin{cases}R^T*R=I\\|R|=1\end{cases}$$

不满足则换一种符号选择方案，失败的原因是分开的求法求得的旋转是**瑕旋转**，行列式为`-1`。

总共存在3次符号选择，则有**8种选择方案**。以上步骤最多重复8次可求得解。

2.3.2 分析

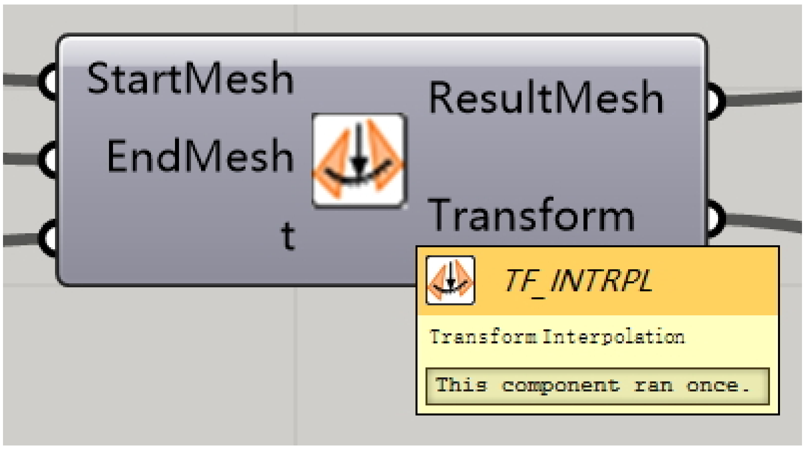
我的方法在算法复杂度上等同于高斯消元法，符号的尝试只是在多次求解二元一次方程。

而矩阵的维度是固定的。所以整个步骤对问题来说是常数时间，虽然方法可能并不是最佳的，但至少可行。

这次作业要求的是从两个`Mesh`出发来插值，而不是输入一个`Mesh`和一个变换矩阵`TF`。

一开始觉得很奇怪，徒增了很多麻烦，但想想还是很合理的，因为对用户来说，得到两个`mesh`是容易的，得到变换矩阵`TF`是比较麻烦的。

3. 插件



插件在`Transform/Euclidean`组里。图标的意思是有两个形状一样的mesh，其间有一个刻度，还有一个用于调节的刻度针。表达的意思就是在两个mesh之间插值。`TF_INTRPL` 就是 `Transform Interpolation`的意思。

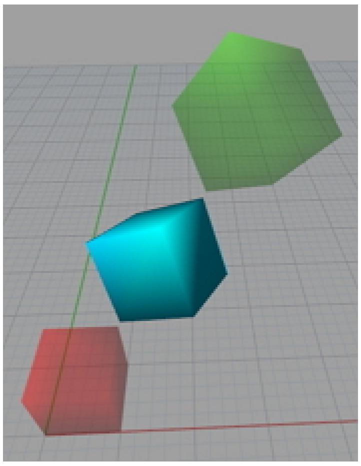
startMesh和endMesh要求形状不变，形状不同的话会有错误信息提示。

t的范围是0-1内，否则报错。可以利用这点作为插件的开关。

4. demo

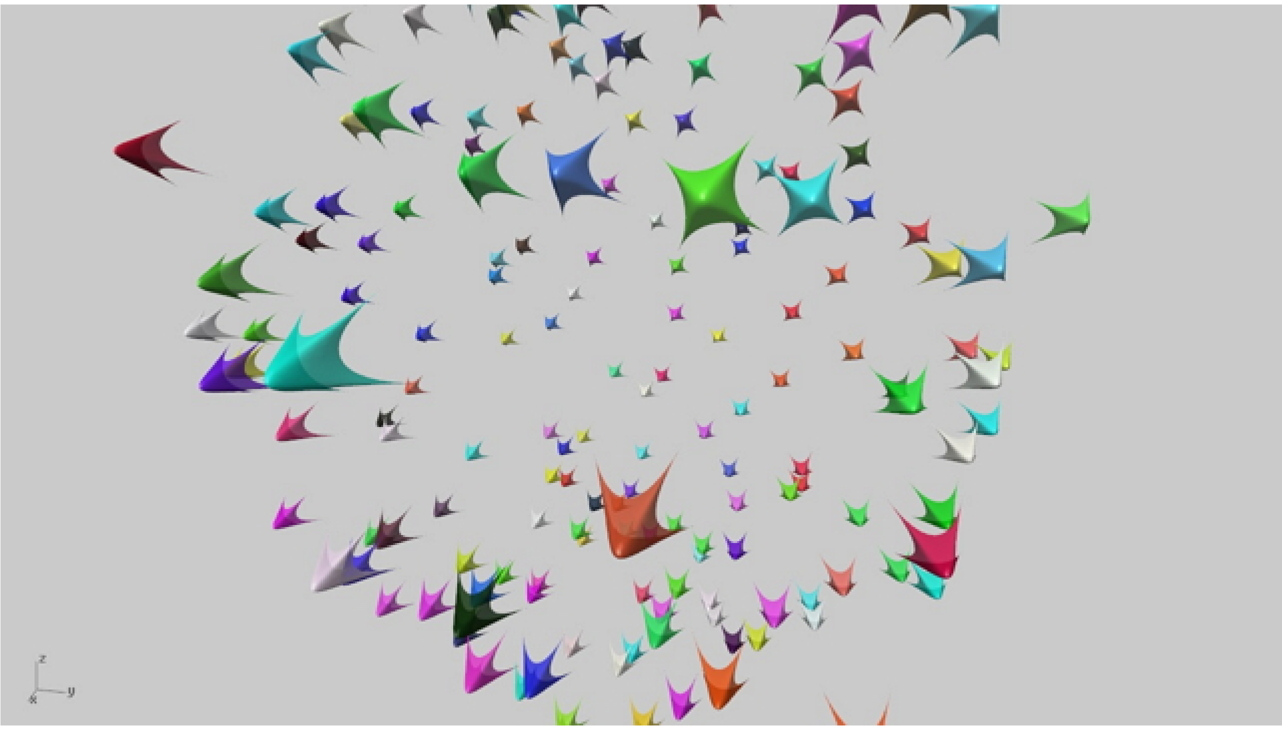
> demo都在`demo.gh`中

4.1 简单的demo1



对应文件是`demo1.gif`。一个正方体做了旋转和平移变换。

4.2 demo2



这个demo总共有**两个过程**。

- 第一个过程是将mesh变换到**球点**上，旋转**180度**。
- 第二个过程是将球点上的mesh变换到 **x+300** 的位置，旋转**90度**，对应关系打散。

镜头在设定好的曲线上移动。

对应的文件是`demo2.mkv`。

720p，30 fps，bgm：`Take me hand`

5.精度问题

精度问题很严重，求根，对0和1的判断都要对浮点数进行特殊处理，要引入一个极小量，如`0.000001`。

一开始是使用float类型，后来全都改成了double类型，问题好了很多。

即使做了一定的处理之后，求解过程还是存在由精度产生的问题，最终导致变换矩阵求解失败。因此我的做法是多次随机选取不同的3对点进行尝试，通过点对的重新选择来应付偶然产生的精度问题。

6. 使用方法

6.1 编译

> 可以跳过这一步骤，因为压缩包里已经含有`TF_INTRPL.dll`和`TF_INTRPL.gha`了

解决方案目录是`TF_INTRPL`

编译后

- `C++`的`TF_INTRPL.dll`文件在`TF_INTRPL\x64\Realease`中
- `C#`的`TF_INTRPL.gha`文件在`TF_INTRPL\TF_INTRPL_CS\bin\x64\Realease`中

6.2 文件放置

- 将`TF_INTRPL.gha`放在`C:\Users\Username\AppData\Roaming\Grasshopper\Libraries`中
- 将第三方插件`MeshTools.gha`、`MeshEdit.gha`和`ghpython.gha`也放在`C:\Users\Username\AppData\Roaming\Grasshopper\Libraries`中
- 将`TF_INTRPL.dll`所在目录加入到系统环境变量`Path`中。

> 我的电脑-->属性-->高级系统设置-->环境变量-->系统变量-->Path

- 打开`Grasshoper`，然后再打开`TF_INTRPL.gh`，就能看到各插件的使用例子了。报告中使用的例子就来自该文件。插件在`Transform/Euclidean`组里。