**Graphics 期末大作业报告**

1. **算法思路**
2. 变形部分

在迭代一中，虽然那个算法也适应任意的模型的变形转换，但是在迭代一中的变形是最为简单基本的，两个模型间点的对应按照点排列的前后顺序来一一对应的，这样在变形的过程中会使得模型的表面不保持，虽然有用三角片来改进变形过程的效果，但是还是觉得不太合适。

在提交后，受到同学讲解说到的用球面与模型上的点一一对应的启发，发现由球面变为模型间的点是可以保持模型的面不破坏的。只要将模型的每个点收缩或拉长为距中心一样长的距离，就会形成一个球面。这样的变形过程，面可以不受到破坏。由此想到，**模型1变为模型2可以不可以先将模型1变为一个球，然后再从一个球变为模型2。**虽然模型1对应球和模型2对应的球是不一样的，但只要保证这两个球的大小一样，那两个球是没有什么区别的。所以先可以变为模型一的球后然后再换成模型2对应的球。

当加入纹理和材质后，变形的过程可以渲染原来的点对应的纹理坐标，但是模型一对应的球突然变为模型二对应的球的过程会比较突然，因为表面的纹理会突然换掉。因此加入了一个缓冲的变换。先逐步减少模型一对应的球的纹理的渲染，使其变为一个纯色的球，然后模型二对应的球从一个纯色的球逐步增加纹理的渲染，渲染完成后再慢慢变成模型二。这样减缓了两个球转变的突变的效果。

1. 纹理与材质渲染

对于纹理和材质的导入部分，没有太多复杂的算法。就是读取obj文件，找到材质库，根据材质库的信息格式，加载纹理图案和材质信息到相关的类中。对于这些数据的使用都在obj f中，f对应的信息是 顶点index/纹理index/法向量index。 作业中实现了对于无压缩的tga格式和bmp格式的纹理贴图文件的解析，可以自动识别tga和bmp格式的图片。

1. 粒子特效

实现一个粒子产生器系统，使其变形完成后能够从物体表面飞出不同颜色，不同速度，不同加速度的粒子。粒子可以用很小的四边形面片表示。为了更好的粒子效果，给粒子的小四边形贴上纹理。粒子拥有着速度，加速度，位置，生命值，消散的速度等特征。

题目中要求粒子从模型的表面发出，则可以传入模型的点的集合用来初始化粒子的位置。为了使粒子能够四面八发的发射，则一个点速度的方向应该和模型中心的点到这个点的向量的正负值一致。这样粒子就会向外发射。还有粒子可以会根据fade 的大小来设置生命减少的速度，当生命为0时，粒子初始点与原来的速度重新发射出来。因此可以在类中记录下初始位置和速度。

1. **算法描述和实现**

作业的代码是面向对象编程的，主要有两大类，obj类和particle类，就是关于物体的加载渲染和粒子系统的。

1. 变形部分

在obj类中有一个成员向量sphere\_vs 来存储该模型对应的球的顶点的值。在obj的顶点加载时，对每个顶点进行计算，将其x,y,z的值进行相同倍数的缩放，使其到原点的距离为1，这样sphere\_vs存储的就是模型obj对应的球的表面的点。

Obj类的对应的两个变形函数

void transfer\_to\_sphere(int step,int steps);//变形,变为球

void sphere\_to\_transfer(int step, int steps);//变形，球变为物体

step表示是第几步变形，steps表示的是一共几步变形.

使用线性插值发，可以计算出在step步要渲染模型的点对应的位置。例如：

Vertex src = m\_vs[m\_fs[i].m\_vds[j].m\_nVI];

Vertex dest = Sphere\_vs[m\_fs[i].m\_vds[j].m\_nVI];

middle.fX = src.fX - (src.fX - dest.fX) / steps\*step;

middle.fY = src.fY - (src.fY - dest.fY) / steps\*step;

middle.fZ = src.fZ - (src.fZ - dest.fZ) / steps\*step;

glVertex3dv((double\*)&(middle))

最后，当模型已经变成了球后，逐步减少对纹理的渲染

int tmp = pow(2, step - steps);

if ((i%tmp)==0&&m\_fs[i].m\_vds[j].m\_bVt)

{

glTexCoord3dv((double\*)&(m\_vts[m\_fs[i].m\_vds[j].m\_nVtI]));

}

每增加一步，tmp就增加一倍，通过i%tmp==0，将渲染的点的纹理减少一半。

1. 纹理和材质渲染

根据obj文件，读取顶点v，法线vn，纹理vt，还有面f 等坐标，根据 mtllib 读取材质文件，加载对应tga格式的纹理文件，绑定对应的纹理。作业中对应创建了MTL, Tga类，用来存储对应的材质纹理信息。而对于bmp格式的纹理可以直接调用LoadImage函数进行解析。

1. 粒子特效

粒子的相关属性

struct particle

{

bool active; // Active (Yes/No)

float life; // Particle Life

float fade; // Fade Speed

float r; // Red Value

float g; // Green Value

float b; // Blue Value

float init\_x; // init X Position

float init\_y; // init Y Position

float init\_z; //init Z Position

float init\_xi; // X speed

float init\_yi; // Y speed

float init\_zi; // Z speed

float x; // X

float y; // Y

float z; // Z

float xi; // X speed

float yi; // Y speed

float zi; // Z speed

float xg; // X

float yg; // Y

float zg; // Z

}

位置的初始化：

particles[loop].x = m\_vs[loop].fX;

particles[loop].y = m\_vs[loop].fY;

particles[loop].z = m\_vs[loop].fZ

保证粒子能向外发射

particles[loop].xi=rand()\* (particles[loop].init\_x - x\_avg) /( abs(particles[loop].init\_x) \*slowdown);

particles[loop].yi=rand()\* (particles[loop].init\_y - y\_avg) /( abs(particles[loop].init\_y) \*slowdown);

particles[loop].zi = rand()\*(particles[loop].init\_z - z\_avg) / (abs(particles[loop].init\_z) \* slowdown);颜色和贴图：

glColor4f(particles[loop].r, particles[loop].g, particles[loop].b,1);

glBegin(GL\_TRIANGLE\_STRIP); glTexCoord2d(1, 1); glVertex3f(x + 0.01f, y + 0.01f, z); // Top Right

glTexCoord2d(0, 1); glVertex3f(x - 0.01f, y + 0.01f, z); // Top Left

glTexCoord2d(1, 0); glVertex3f(x + 0.01f, y - 0.01f, z); // Bottom Right

glTexCoord2d(0, 0); glVertex3f(x - 0.01f, y - 0.01f, z); // Bottom Left

glEnd();

1. **编程环境：**

Win10+vs2013，使用glut库

1. **配置说明：**

将glut 库添加在vs的文件夹下（详情按照助教发的教程），无其它额外配置。

1. **程序运行说明：**

程序一进去会是一个渲染好的3d；

“w”键： 物体向前移动

“s ”键： 物体向后移动

“a”键： 物体向左移动

“d”键： 物体向右移动

“r” 键： 物体恢复默认位置

“鼠标左”键： 按住可旋转物体，改变视角

“鼠标中”键： 按住前后移动可缩放

“1”键：物体变换为另一个加载好的3d模形，变形结束后发散出粒子

1. **实验结果和分析：**

实现了视角的变换和变形的动画以及粒子效果，变形的效果相对于迭代一有所改善，但还可以继续改进。作业中只能加载tga和tmp格式的纹理图片，之后可以扩展更多的格式。