大 作 业 报 告

**课程名称：** 计算机图形学

**大作业项目：** 面消隐算法

**实验仪器：** 计算机

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **项目** | **原理阐释** | **完成度与代码规范性** | **独创性** | **合计** |
| **百分比(%)** | **30** | **50** | **20** | **100** |
| 得分 |  |  |  |  |

**系 别： 计算机系**

**专 业： 计算机科学与技术**

**班级姓名： 计科1905/陈浩**

**学 号： 2019010132**

**日 期： 2021/12/4**

**成 绩：**

**指导教师：** 丁 濛

# 实验目的

1. 综合运用课程所讲述的图形学相关原理，实现绘制一个给定的3D模型。

# 基本功能要求

1. 通过文件读取一个3D模型(模型具体为由若干个多边形面片刻画的你的姓氏)，文件存储格式可以自己定义。
2. 实现消隐算法，正确将3D模型按照某种投影方式绘制在设备上(可以是图像，也可以是窗口)。
3. 通过鼠标点击，实现旋转模型的功能。

以上为基本要求，可以根据自身能力水平添加额外的功能。

# 基本原理阐释

整体流程：

* 1. 读取以坐标形式保存的txt文件
  2. 采用Model Transform、View Transform、Perspective Transform将txt文件中的坐标进行输出，并将由坐标构成的图形从三维变为二维；
  3. 采用Viewport Transform将所构成的二维图像移植以窗口的坐标系下，并以该原点为固定点，对图形施加旋转功能；
  4. 采用固定光源和固定相机，对图形采用Z-Buffer算法，利用颜色depth和color depth定义图形的每一个像素，将z值较大的保存其深度值和颜色值，最后输出其颜色；

# 系统设计与实现

主要代码：

* + 1. 读文件：

QFile file("D:/Assignments/Graphics/ZBuffer/data\_ch.txt");

*if*(file.*open*(QIODevice::*ReadOnly* | QIODevice::*Text*)){

QTextStream inStream(*&file*);

*while*(!inStream.atEnd()){

int n;

inStream >> n;

*if*(n == 0)

*break*;

QVector<float> list;

list.reserve(n \* 3);

*for*(int i = 0; i < n; ++i){

int a, b, c;

inStream >> a >> b >> c;

list << a << b << c;

}

patchList.push\_back(Patch3D(*list.data()*, list.size(), QColor( rand() % 255, rand() % 255, rand() % 255) ));

}

}

* + 1. 实现MVP算法：

void **updateEquation**() {

QVector3D v1, v2;

v1 = QVector3D(ptList.at(1) - ptList.at(0));

v2 = QVector3D(ptList.at(2) - ptList.at(1));

*//(QVector3D::crossProduct(v1,* *v2)).*

QVector3D normal = QVector3D::crossProduct(v1, v2);

normal.normalize();

float d = -QVector3D::dotProduct(QVector3D(ptList.at(0)), normal);

equation = QVector4D( normal.x(), normal.y(), normal.z(), d);

}

void **rotate**(double degree = 0., QVector3D axis = QVector3D(0.f, 0.f, 1.f)){

QMatrix4x4 rotation;

rotation.rotate(degree, axis);

*for*(QVector4D& pt : ptList){

pt = rotation.map(pt);

}

updateEquation();

}

void **translate**(QVector3D offset) {

QMatrix4x4 translation;

translation.translate(offset);

*for*(QVector4D& pt : ptList){

pt = translation.map(pt);

}

updateEquation();

}

void **perspective**(*const* QVector3D &pos, *const* QVector3D& direction, *const* float nearPlane){

QPolygonF polygon;

*for*(Patch3D& p : patchList){

p.translate(QVector3D(-pos.x(), -pos.y(), -pos.z() - nearPlane));

}

direction.normalized();

*if*(qAbs(direction.x()) < 1.e-6) {

double angle = qRadiansToDegrees(qAcos(direction.z() / direction.length()));

*if*(direction.y() > 0)

angle = -angle;

*for*(Patch3D& p : patchList) {

p.rotate(-angle,QVector3D(1.f, 0.f, 0.f));

}

QMatrix4x4 temp;

temp.rotate(-angle, QVector3D(1.f,0.f,0.f));

viewDirection = temp.map(direction);

eye = temp.map(pos);

}

*else* *if* (qAbs(direction.y()) < 1.e-6) {

double angle = qRadiansToDegrees(qAcos(direction.z() / direction.length()));

*if*(direction.x() < 0 )

angle = -angle;

*for*(Patch3D& p : patchList) {

p.rotate(-angle,QVector3D(0.f, 1.f, 0.f));

}

QMatrix4x4 temp;

temp.rotate(-angle, QVector3D(0.f,1.f,0.f));

viewDirection = temp.map(direction);

eye = temp.map(pos);

}

*else* {

double angle = qAcos(direction.z() / direction.length());

*if*(direction.x() < 0 )

angle = -angle;

*if*(QVector2D(direction.x(), direction.y()).length() > 1e-6){

double theta = -qRadiansToDegrees(qAsin(direction.y() / QVector2D(direction.x(), direction.y()).length()));

*if*(direction.y() < 0)

theta = -theta;

*for*(Patch3D p : patchList) {

p.rotate(theta, QVector3D(0.f,0.f,1.f));

}

QMatrix4x4 temp;

temp.rotate(theta, QVector3D(0.f,0.f,1.f));

viewDirection = temp.map(direction);

eye = temp.map(pos);

}

*for*(Patch3D& p : patchList) {

p.rotate(-qRadiansToDegrees(angle),QVector3D(0.f, 1.f, 0.f));

}

QMatrix4x4 temp;

temp.rotate(-qRadiansToDegrees(angle), QVector3D(0.f,1.f,0.f));

viewDirection = temp.map(direction);

eye = temp.map(pos);

}

3.实现Z-Buffer算法：

void **ZBuffer**(QImage &image, *const* QVector<Patch3D>& patchList, QVector4D eye){

QVector<QPolygonF> planarList;

planarList.reserve(patchList.size());

QVector<int> patchIdList;

*for*(int i = 0; i < patchList.size(); ++i){

*const* Patch3D& patch = patchList.at(i);

#ifdef PERSPECTIVE

patchIdList << i;

planarList << patch.getPerspectiveProjection(QVector3D(eye));

#else

*if*(!patch.isOrthoXOY()) {

patchIdList << i;

planarList << patch.getXOYOrthoProjection();

}

#endif

}

QVector<float> zbuffer;

zbuffer.resize(800 \* 600);

zbuffer.fill(1000000.);

*for*(int i = 0; i < planarList.size(); ++i){

*const* QPolygonF& p = planarList.at(i);

QRectF boundingBox = p.boundingRect();

*for*(int x = boundingBox.left(); x < boundingBox.right(); ++x) {

*if*(x <= 0 || x >=800 )

*continue*;

*for*(int y = boundingBox.top() ; y < boundingBox.bottom(); ++y){

*if*(y <= 0 || y >= 600)

*continue*;

unsigned char \*scanLine = image.scanLine(y);

*if*(p.containsPoint(QPointF(x,y), Qt::*OddEvenFill*)) {

double z;

*if*(eye.w() == 0)

z = patchList.at(patchIdList.at(i)).getZValue(QPointF(x, y));

*else*

z = patchList.at(patchIdList.at(i)).getZValue(QVector3D(eye), QVector3D(x - eye.x(), y - eye.y(), - eye.z()).normalized());

*if*(z < zbuffer[y \* 800 + x]) {

zbuffer[y \* 800 + x] = z;

QColor c = patchList.at(patchIdList.at(i)).getColor();

scanLine[x \* 3 + 0] = c.red();

scanLine[x \* 3 + 1] = c.green();

scanLine[x \* 3 + 2] = c.blue();

}

}

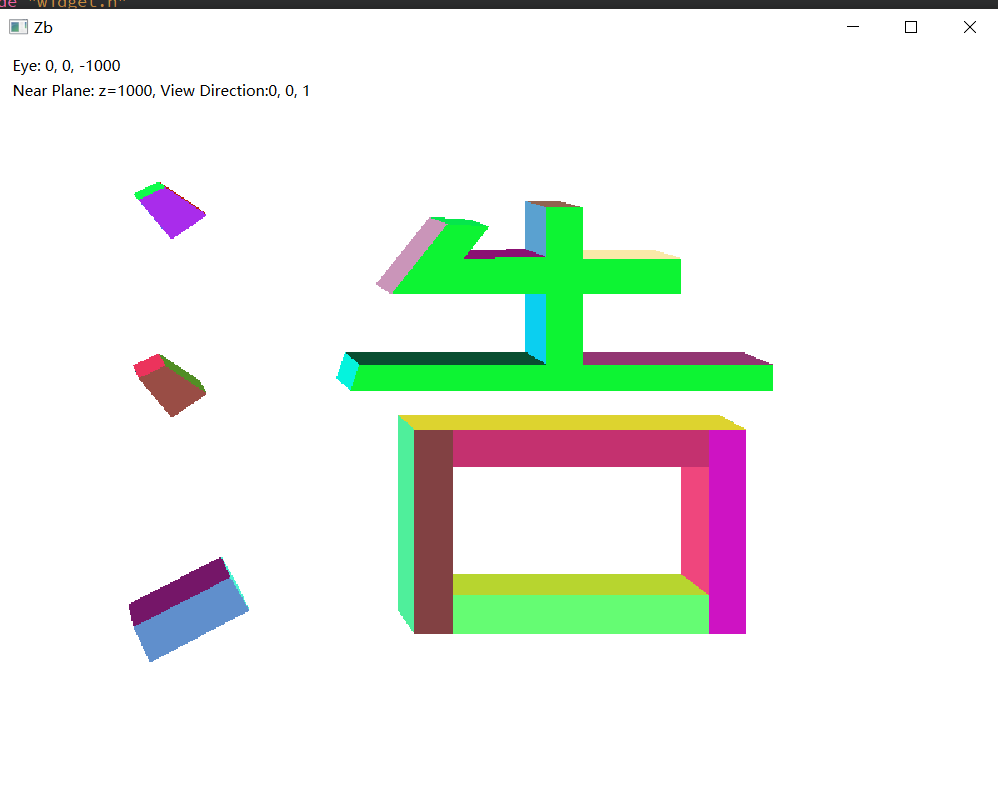
}

}

}

}

实验结果：



# 结果分析与讨论

对于本学期开设的图形学课程，我很有幸能够选到，并跟着丁老师的步骤，一步步了解图形学的发展，并入门简单的图形学。

正是这门图形学课，让我真正知道了自己将来想要研究的方向。在完成图形学课程作业途中，我也有幸接触了闫令琪老师的Games101课程，跟着闫老师的Games101，并结合图形学课所介绍的其他知识，让我对图形学有了一定的认识，并且这是大学期间我第一次真正的对一门学科感兴趣。

本门课程学习完毕，也只是图形学的入门。

该课程主要介绍了：

1. .如何画出一条直线或一个圆：可以采用DDA算法或Brensenham算法，通过对直线和圆的基本表达式的变换，以及确定中点和交点的位置，便可以确定直线或圆上分布的各个像素的位置，并将其依次连接就能构成连续的线段和圆；
2. . 如何对一个指定区域进行填充：可以采用种子填充算法或Fill-Flood算法，其中可以用一个包围盒bounding-box将构建的图形框选，采用右手定则对包围盒里的各个点和图形的各个顶点进行叉乘，若结果非负，则表明该点在图形内部，应该对其填充颜色；
3. . 如何绘制一个Bezier曲线：Bezier曲线是依靠将各个线段等分的点的线性分布所构成的；其中构成二阶Beizer曲线需要三个点构成两条线段，在线段上选取两个共点(将两条线段按比例等分)，相连构成一条新的线段，并在新线段上再采用等分选取共点，该点便是Bezier曲线上的一点；而三阶Bizer曲线是由两条Bezier曲线的共点构成，以此类推；
4. . 如何将立体图形从三维平面转变到二维平面：采用MVP Transform和Viewport Transform；首先用Model T确立图形的变换位置，再用View T更换摄像机的位置(一般是移动到世界坐标原点，再对准物体)，再用Perspective T将图形投影到二维平面(最后需要进行正交计算)，最后用Viewport T将图形移动到笛卡尔坐标系的原点处；

接着可以采用Z-Buffer算法，记录图形上各个像素的颜色和深度值，将深度值较大(根据坐标系规定)的颜色保存到摄像机的近平面处，即可距离相机不同距离的图形颜色和遮挡状态；

后续还可以采用MSAA或SSAA进行对图形的抗锯齿，也能够引入光照模型计算ADS分布，实现物体的镜面反射、漫反射和环境光构成，并进行着色或进行纹理的贴图等；

通过对本门课程的初步学习，让我对图形学有了基本的认知和了解，并且也让我产生了对图形学的极大兴趣，十分感谢丁老师能够开设图形学这门选修课，希望今后能够进一步的学习图形学理论和应用。