

《计算机图形学》系统技术报告

张昊*

(南京大学 计算机科学与技术系, 南京 210093)

摘要: 如果不考虑一个架构完整的图形处理系统, 单独实现某条直线或多边形的输入编辑输出功能是非常简易的, 甚至可以说是无趣的, 而提到设计一个完整的系统就不仅仅是解决某条直线某种算法的问题, 这上升到考察你计划和架构整个系统框架的能力, 这将是一个更大的挑战。《计算机图形学》的这份学期大作业无疑与此类似, 检测学生对某种图形的生成和变换是基础, 更加需要学生去掌握对这些图形显示和编辑功能的控制能力, 从无到有, 从有到完整, 从设计这个系统的每一步体会、反思、提高自己的编程能力。

关键词: 计算机图形学、OpenGL、C++面向对象程序设计

一、引言

经过一个学期的图形学课程学习和图形学大作业的编程, 笔者已经完成《计算机图形学》大作业(下称CG Program)的全部要求。从图形的种类来看, 共有点、直线(或称线段)、多边形、圆、椭圆、曲线、立体图形(笔者选用OpenGL中的Teapot)七种图形, 并共同继承了一个名为Graphics的基类; 从每种图形的功能角度来看, 包括图形的输入、绘制、线条颜色设置、线宽设置、移动、缩放、旋转、填充、裁剪、输出数据、保存至文件等功能; 从与用户交互的角度来看, 有通过鼠标初始图形、弹出式菜单选择功能、键盘按键来移动和旋转图形等交互方式。

二、系统设计

1、模块简介

(1) 图形模块:

Graphics: 是全部图形的基础, 充当图形的基类, 内部的成员函数基本全是虚函数。

Graphics
-graphicsType: string
(for class teapot) rotateX: double rotateY: double
Graphics() ~Graphics() float2str(float f): string COLOR2str(COLOR c): string
virtual setLineWidth(float lw) = 0: void virtual setLineColor(float r, float g, float b) = 0: void virtual Draw() = 0: void virtual Up() = 0: void virtual Down() = 0: void virtual Left() = 0: void virtual Right() = 0: void virtual Increase() = 0: void virtual Decrease() = 0: void virtual Rotate(char) = 0: void virtual Fill(float r, float g, float b) = 0: void virtual Disappear() = 0: void virtual SaveFile() = 0: String

* 作者简介: 姓名张昊, 学号141220141, 电子邮箱为hao_silence@qq.com

Point：是点的类，继承graphics类，包含点的属性以及对点的操作函数。

Point
-pos: COORD -lineWidth: float -lineColor: COLOR
Point() Point(COORD p) Point(float x, float y) ~Point() getPos() const: COORD setPos(COORD p): void getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Line：直线的类，继承graphics类，包含直线的全部属性以及直线绘制、旋转等一系列函数。

Line
-begin: COORD -end: COORD -lineWidth: float -lineColor: COLOR -length: double -n: int angle: double dir[2]: double
Line() Line(COORD s, COORD e) Line(float sx, float sy, float ex, float ey) ~Line() getLinePos() const: SMALL_RECT setLinePos(COORD b, COORD e): void getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Polygon: 多边形的类, 继承graphics类, 包含多边形的顶点等数据以及多边形的具体操作函数。

Polygon
-Vertex: vector<COORD> -lineWidth: float -lineColor: COLOR -length: double* -n: int -angle: double -dir: DIR*
Polygon(); Polygon(int n, COORD c[]); ~Polygon(); getVertex(int n): COORD getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Circle: 圆的类, 继承graphics类, 包含圆心、半径等属性, 实现一些具体的操作函数。

Circle
-center: COORD -radiusa: float -radiusb: float -lineWidth: float -lineColor: COLOR -length: double*
Circle() Circle(COORD c, float r) ~Circle() getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Ellipse: 继承graphics类的椭圆类，类内包括椭圆的图形数据以及各种操作函数。

Ellipse
-center: COORD -a: float -b: float -lineWidth: float -lineColor: COLOR
Ellipse() Ellipse(COORD c, float a, float b) ~Ellipse() getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void Ellipsepot(int x0, int y0, int x, int y): void setPixel(int x, int y): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Curve: 继承graphics类的曲线类，包括Bezier曲线的生成等各种成员函数，具体UML架构如下：

Curve
-Point[4]: COORD -Bezier[11]: COORD -lineWidth: float -lineColor: COLOR
Curve() Curve(int n, COORD c[]) ~Curve() getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void calBezier(): void DrawPoint(int n, COORD c[]): void DrawLine(int n, COORD c[]): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Teapot: 继承自graphics类的类，充当三维图形，包含三维图形的绘制、旋转等功能，如下：

Teapot
-Point[4]: COORD -Bezier[11]: COORD -lineWidth: float -lineColor: COLOR
Curve() Curve(int n, COORD c[]) ~Curve() getLineWidth() const: Float setLineWidth(float p): void getLineColor() const: COLOR setLineColor(float r, float g, float b): void calBezier(): void DrawPoint(int n, COORD c[]): void DrawLine(int n, COORD c[]): void Draw(): void Up(): void Down(): void Left(): void Right(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void Disappear(): void SaveFile(): string

Define: Define模块专门存放宏定义命令，比如弹出式菜单的编号、COLOR的定义等。

Define
Menu No Color Type rand DrawLine

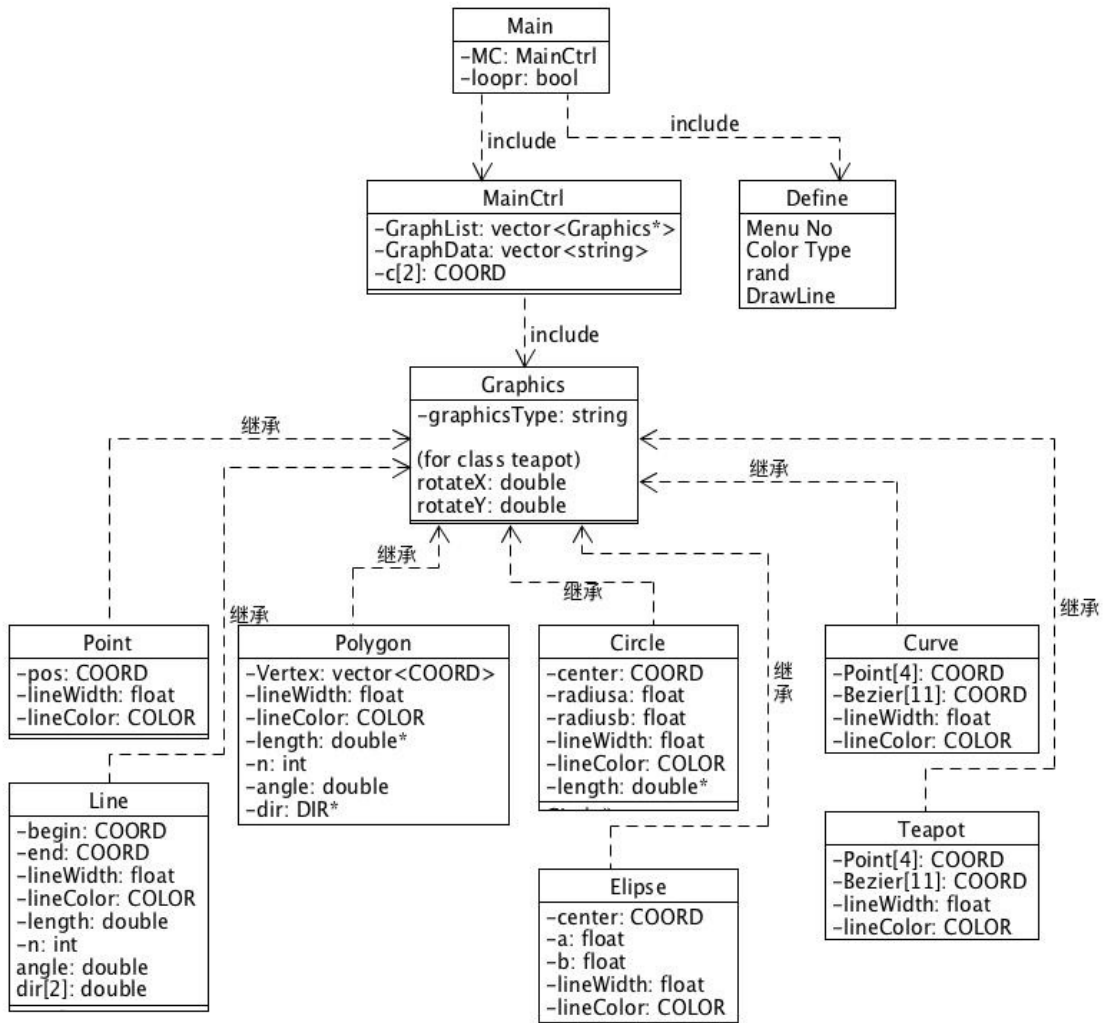
MainCtrl: 是图形的管理类，用List管理所创建的图形，并统一对图形进行编辑变换。

MainCtrl
-GraphList: vector<Graphics*> -GraphData: vector<string> -c[2]: COORD
MainCtrl(); MainCtrl(string filename): void ReadFromFile(string filename); ~MainCtrl(); SetLineColor(float r, float g, float b): void SetLineWidth(float lw): void DrawGraph(): void UpGraph(): void DownGraph(): void LeftGraph(): void RightGraph(): void Increase(): void Decrease(): void Rotate(char c): void Fill(float r, float g, float b): void CutGraph(): void Disappear(): void SaveToFile(string filename): void str2float(string s): float str2COLOR(string s): COLOR splitStr(string line, string flag): vector<string>

Main: 是整个系统的入口，包含glut的初始化和一些用户与系统的交互函数等。

Main
-MC: MainCtrl -loopr: bool
Init(): void myDisplay(): void processMenuEvents(int option): void createGlutMenu(): void mouse(int button, int state, int x, int y): void timer(int p): void keyboard(unsigned char key, int x, int y): void keyboardSpecial(int key, int x, int y): void main(int argc, char **argv): int

2、模块间联系



三、功能实现

1、图形输入：

(1) point: 点的初始化有三种构造函数, 分别为Point()、Point(float x, float y)、Point(COORD p), 将传入的坐标作为点的坐标位置初始化(默认初始化为(0,0)), 然后线宽和线条颜色分别初始化为1.0和{0.0f, 0.0f, 0.0f};部分代码如下:

```
lineWidth = 1.0f;
COLOR c = {0.0f, 0.0f, 0.0f}; //COLOR为自定义的结构体, 后面会介绍到
lineColor = c;
```

(2) line: 直线同样有三种构造函数, 分别为Line()、Line(COORD b, COORD e)、Line(float sx, float sy, float ex, float ey), 传入的参数当作直线的始终点坐标(默认初始化为(0,0)), 线宽和线条颜色分别初始化为1.0和{0.0f, 0.0f, 0.0f}, 并对直线的方向向量dir和长度length做初始化, length和dir在旋转的时候需要用到; 部分代码如下:

```
length = sqrt( pow((end.Y-begin.Y), 2) + pow((end.X-begin.X), 2) ); //length为直线的长度
dir[0] = (end.X-begin.X) / length; //dir[2]为方向向量
dir[1] = (end.Y-begin.Y) / length;
angle = 30*3.1415926/180.0; //angle表示30度, 设置旋转一次的角度为30度
n = 1; //n为旋转次数的计数
```

(3) polygon: 多边形的构造函数有两种, 分别为Polygon()、Polygon(int n, COORD c[]), 默认为四边形, 且坐标分别为(0,0), (-100, 0), (-100, 100), (0,100)。传入的参数n表示多边形的边数目, c[]数组为各个顶点的元素; 线宽和线条颜色分别初始化为1.0和{0.0f, 0.0f, 0.0f}; 最后, 以第一个顶点为起点, 初始化其余顶点与该顶点的方向向量dir[i]和距离length[i], 初始化的部分代码如下:

```
for(unsigned int i = 0; i < Vertex.size()-1; i++) //每个顶点计算与第一个顶点的距离长度
    length[i]=sqrt(pow((Vertex[i+1].Y-Vertex[0].Y), 2)+pow((Vertex[i+1].X-Vertex[0].X), 2));
for(unsigned int i = 0; i < Vertex.size()-1; i++) //每个顶点与第一个顶点构成的方向向量
{
    dir[i].X = ( Vertex[i+1].X-Vertex[0].X ) / length[i];
    dir[i].Y = ( Vertex[i+1].Y-Vertex[0].Y ) / length[i];
}
```

(4) circle: 圆有两种初始化方法, 第一种是默认初始化, 圆心为(0,0), 半径为0, 第二种构造函数为Circle(COORD c, float r), 其中c为坐标, r为半径。线宽和线条颜色同上。

(5) ellipse: 椭圆也有两种初始化方法: 第一种默认中心和长短轴均为0, 第二种构造函数Ellipse(COORD c, float a, float b)对中心和a、b进行初始化, 两种构造函数对线宽和颜色初始化同上。

(6) curve: 曲线的无参构造函数先不对顶点赋值, 另一个构造函数Curve(int n, COORD c[])的n表示n个控制顶点, COORD数组表示控制顶点的坐标。然后对颜色和线宽初始化。

(7) teapot: 对于要求中的立体图形笔者采用OpenGL的茶壶, 茶壶的默认构造函数初始化茶壶的size为0, 有参数的构造函数Teapot(double s)初始化size为s, 然后需要对茶壶的起始位置、线宽、颜色、是否填充的标志初始化, 部分代码如下:

```
size = s; //表示茶壶的size
lineWidth = 1.0f;
```

```

COLOR c = {0.0f, 0.0f, 0.0f};
lineColor = c;
posX = 150.0; //posX,posY,posZ为茶壶在坐标系中的初始位置
posY = 0.0;
posZ = 0.0;
rotateX = 30; //为了显示立体效果，第一次显示茶壶时偏转一定角度
rotateY = 30;
filled = false; //filled为茶壶是否填充的flag

```

2、图形绘制

(1) point: 点的绘制方法是调用OpenGL中的画点函数来绘制一个点，这是最基础的图形绘制，具体的绘制代码如下：

```

glPointSize(lineWidth); //绘制时设置点的size，即线宽（统一命名为lineWidth）
glColor3f(lineColor.R, lineColor.G, lineColor.B); //设置点的颜色
glBegin(GL_POINT);
glVertex2d(pos.X, pos.Y); //pos如上面图形的输入所述，是点的坐标
glEnd();
glFlush(); //flush缓冲区立即执行

```

(2) line: 直线的显示是判断直线的斜率，然后利用直线生成的Bresenham算法来生成。具体的判断过程分为以下情况：先判断斜率 k 是否存在，若不存在则是一条垂直 x 轴的直线，直接绘制即可；如果存在，再判断 k 是否为0，若 k 为0则说明直线平行于 x 轴也可以直接绘制；继续判断斜率是否是否等于1，这种情况也是很容易绘制直线的，只需从左边的顶点横坐标和纵坐标逐步增加即可， k 为-1的情况与1类似，从左边的顶点 x 逐渐增加 y 逐渐减少绘制；剩余的为一般情况，用Bresenham算法即可，算法细节在此不再赘述。直线的部分生成代码如下所示：

```

/*直线无斜率时*/
glBegin(GL_POINT);
while(y1 <= y2)
    glVertex2d(x1, y1);
    y1++; //x值不变，只需增加y值
glEnd();

/*斜率k为0*/
glBegin(GL_POINT);
while(x1 <= x2) //确定较小的x值
    glVertex2d(x1, y1);
    x1++; //y值不变，增加x值
glEnd();

/*斜率k为正负1*/
glBegin(GL_POINT);
while(x1 <= x2)
{
    glVertex2d(x1, y1);
    x1++;
}

```

```
        y1++;    //k = -1时变为y1- -;
    }
    /*k = -1 省略*/

    /* k的绝对值大于1 */
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int temp1 = 2 * dx;
    int temp2 = 2 * (dx - dy);
    int p = 2 * dx - dy;
    if(y1 > y2)
    {
        swap(x1, x2);
        swap(y1, y2);
    }

    glBegin(GL_POINTS);
    glVertex2d(x1, y1);
    while(y1 <= y2)
    {
        y1++;
        if(p < 0)
            p += temp1;
        else
        {
            if(flag_k == 1)
                x1++;
            else
                x1--;
            p += temp2;
        }
        glVertex2f(x1, y1);
    }
    glEnd();
    glFlush();

    /* k的绝对值小于1 */
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int temp1 = 2 * dy;
    int temp2 = 2 * (dy - dx);
    int p = 2 * dy - dx;
    if(x1 > x2)
        swap(x1, x2);
```

```

        swap(y1, y2);
    glBegin(GL_POINT);
    glVertex2d(x1, y1);
    while(x1 < x2)
    {
        x1++;
        if(p < 0)
            p += temp1;
        else
        {
            if(flag_k == 1)
                y1++;
            else
                y1--;
            p += temp2;
        }
        glVertex2d(x1, y1);
    }
}

```

(3) polygon: 多边形的显示可以利用已经完成的直线生成算法, 将多边形顶点数组中的两个相邻的顶点连接成线, 便可以生成多边形, 注意要注意数组首尾两个顶点的连接, 形成封闭的多边形。具体代码如下:

```

for(auto i = Vertex.begin(); i != Vertex.end()-1; i++)
{
    COORD b = { (*i).X, (*i).Y};
    COORD e = { (*(i+1)).X, (*(i+1)).Y};
    Line L(b, e);
    L.setLineColor(lineColor.R, lineColor.G, lineColor.B);
    L.setLineWidth(lineWidth);
    L.Draw();
}
COORD b = { (*Vertex.begin()).X, (*Vertex.begin()).Y};    //第一个顶点
COORD e = { (*(Vertex.end()-1)).X, (*(Vertex.end()-1)).Y}; //最后一个顶点
Line L(b,e);
L.setLineColor(lineColor.R, lineColor.G, lineColor.B);
L.setLineWidth(lineWidth);
L.Draw(); //形成封闭的多边形

```

(4) ellipse: 椭圆的生成算法原理是利用椭圆的四路对称特性, 只需根据中点椭圆生成算法绘制出第一象限的椭圆曲线, 即可根据对称性原理绘制出其他部分。那第一象限的部分应该如何绘制呢? 按照斜率为1的直线将其分为两段, 每段利用中点椭圆生成算法分别增加y值和x值, 即可绘制, 具体细节不再赘述, 有以下代码加以解释:

```

double d = sqb + sqx*(0.25 - b);
int x = 0;
int y = b;
Elipsepot(x0, y0, x, y); //此函数的功能是绘制与(x0,y0)对称的四个点

```

```

// 1 绘制第一象限曲线的靠近y轴的部分
while (sqb*(x + 1) < sqa*(y - 0.5))
{
    if (d < 0)
        d += sqb*(2 * x + 3);
    else
    {
        d += (sqb*(2 * x + 3) + sqa*((-2)*y + 2));
        --y;    //以y轴为变量
    }
    ++x;
    Ellipsepot(x0, y0, x, y); //每确定一个点便按照四路对称画其他点
}
d = (b * (x + 0.5)) * 2 + (a * (y - 1)) * 2 - (a * b) * 2;
// 2绘制第一象限曲线的靠近x轴的第二部分
while (y > 0)
{
    if (d < 0)
    {
        d += sqb * (2 * x + 2) + sqa * ((-2) * y + 3);
        ++x;    //以x轴为增量
    }
    else
        d += sqa * ((-2) * y + 3);

    --y;
    Ellipsepot(x0, y0, x, y);
}

```

(5) circle: 圆可以看作特殊的椭圆（长轴等于短轴的椭圆），事实上笔者也是这么处理的，在圆的定义中将半径看作radiusa和radiusb，这样做是因为可以利用椭圆的绘制方法来生成圆。利用定义圆的属性变量初始化一个椭圆，然后设置相关的属性，就可以调用椭圆的Draw()函数来绘制圆。部分代码如下：

```

Ellipse circle(center, radiusa, radiusb); //为满足椭圆设为radiusa和radiusb，两者相等
circle.setLineWidth(lineWidth);           //设置线宽
circle.setLineColor(lineColor.R, lineColor.G, lineColor.B); //设置线条颜色
circle.Draw(); //利用椭圆的Draw()函数

```

(6) curve: 曲线的生成笔者利用课上孙老师讲过的Bezier曲线生成算法来绘制，先绘制曲线的控制顶点，然后根据控制顶点来计算曲线的“折点”，然后根据多边折线的各顶点唯一确定一条Bezier曲线。画点和连线的函数比较简单，在此不解释了，重点解释如何确定多边折线的顶点。

```

float a0,a1,a2,a3,b0,b1,b2,b3;
a0=Point[0].X;
a1=-3*Point[0].X+3*Point[1].X;
a2=3*Point[0].X-6*Point[1].X+3*Point[2].X;
a3=-Point[0].X+3*Point[1].X-3*Point[2].X+Point[3].X;
b0=Point[0].Y;

```

```

b1=-3*Point[0].Y+3*Point[1].Y;
b2=3*Point[0].Y-6*Point[1].Y+3*Point[2].Y;
b3=-Point[0].Y+3*Point[1].Y-3*Point[2].Y+Point[3].Y;

float t = 0;
float dt = 0.01;

for(int i = 0; t<1.1; t+=0.1, i++)
{
    Bezier[i].X = a0+a1*t+a2*t*t+a3*t*t*t;
    Bezier[i].Y = b0+b1*t+b2*t*t+b3*t*t*t;
}

```

(7) teapot: 茶壶的生成是调用OpenGL的函数glutWireTeapot(double size)来绘制的, 前面在teapot的初始化的时候说到, 为了使更好的显示茶壶的立体感, 特意调整了茶壶初始的角度(rotateX和rotateY), 并且对茶壶的初始位置也做了调整(posX、posY和posZ)。所以在绘制茶壶时需要先对这些属性进行设置如下:

```

/* 对颜色和线宽的设置同其他图形, 在此略 */
glPushMatrix();           //防止修改其他图形的属性
glRotatef(rotateY,0,1,0); //是想在物体的初始状态时旋转一定角度
glRotatef(rotateX,1,0,0);
glTranslated(posX,posY,posZ);
if(filled)                //判断图形是否为填充状态
    glutSolidTeapot(size);
else
    glutWireTeapot(size);
glPopMatrix();

```

3、图形颜色和线宽:

正如初始化时对所有图形设置线宽和颜色属性, 笔者设置了对这些属性的调整函数setLineWidth(float lw)和setLineColor(float r, float g, float b)。在每种图形的Draw()函数中绘制之前设置brush的颜色即可, 如glPointSize(lineWidth)和glColor3f(lineColor.R, lineColor.G, lineColor.B)。

4、图形平移: 分为up、down、left、right四个方向的平移

(1) point: 点平移时只需要按照平移的方向改变pos的坐标。若向上平移则pos.Y += 40; 若向下平移则pos.Y -= 40; 向左平移则pos.X -= 40; 向右则pos.X += 40。

(2) line: 直线的平移和点的类似, 与之不同的是直线移动需要调整begin和end两点的坐标。上下平移则分别增减begin.Y和end.Y; 左右平移则分别增减beginX和end.X。

(3) polygon: 多边形的平移是需要增减顶点数组中每个顶点的坐标值, 部分代码如下:

```

for(auto i = Vertex.begin(); i != Vertex.end(); i++)
{
    (*i).Y += 40; //up
    (*i).Y -= 40; //down
    (*i).X -= 40; //left
    (*i).X += 40; //right
}

```

(4) ellipse: 平移椭圆只需要移动椭圆中心的坐标。上移则center.Y += 40; 下移则center.Y -= 40;左移则center.X -= 40;右移则center.X += 40;

(5) circle: 与椭圆相同, 都是移动中心的相应坐标, 不再赘述。

(6) curve: 曲线移动的时候是移动其控制顶点的坐标, 与移动多边形类似。(笔者有考虑过只移动求得的Bezier曲线折点的坐标, 但这种方法只是暂时的获得效果, 不能持久地保存结果)

(7) teapot: 因为初始化茶壶时有用三个变量posX、posY、posZ设定其位置, 所以移动时只需要改变相应的坐标即可, 和以上图形移动相似。

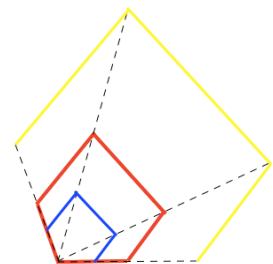
5、图形缩放:

(1) point: 点无缩放, 略。

(2) line: 笔者对直线的缩放采用的是成倍伸长或缩短。若要求伸长, 则每次长度增加为原来的两倍; 若要求缩短, 则每次长度减短为原来的一半。具体的代码如下所示:

```
/*伸长时直线尾端点坐标的变化*/
end.X += end.X - begin.X;
end.Y += end.Y - begin.Y;
/*缩短时尾端坐标的变化*/
end.X -= (end.X - begin.X) * 0.5;
end.Y -= (end.Y - begin.Y) * 0.5;
/*变化之后需要更新直线的长度和单位方向向量*/
length = sqrt( pow((end.Y-begin.Y), 2) + pow((end.X-begin.X), 2) );
dir[0] = (end.X-begin.X) / length;
dir[1] = (end.Y-begin.Y) / length;
```

(3) polygon: 笔者设计的多边形的缩放算法原理与直线类似。上面提到多边形初始化时记录了各顶点到第一个顶点的长度和向量, 把除第一个顶点的其余各点与第一个顶点连接构成n-1条直线, 分别按照直线的缩放方法计算即可, 代码不再粘贴, 只是将n-1条直线循环做与直线类似的缩放处理即可。算法原理的图示如有图所示, 红色为原始多边形, 黄色为放大的多边形, 蓝色为缩小的多边形。



(4) ellipse: 椭圆的缩放是放大或缩小其长短轴; 但是要注意a和b必须等比例的缩放, 否则椭圆的“高矮胖瘦”会变化, 笔者采用的是b以20为单位变化, a以a/b * 20变化, a%b的细节不予考虑。

(5) circle: 圆的缩放较为简单, 只需增减圆的半径即可, 不多解释。

(6) curve: 曲线的缩放是类比与多边形的缩放, 笔者将曲线的控制顶点看作多边形的顶点, 先对这些“多边形的顶点”(即曲线的控制顶点)缩放, 然后重新计算Bezier的控制点即可实现曲线的缩放。代码如下:

```
class::Polygon c(4, Point); //曲线的控制顶点按照多边形处理
c.Increase(); //多边形的缩放模式, 这里是放大, 若缩小调用Decrease()
for(int i = 0; i < 4; i++)
```

```
Point[i] = c.getVertex(i); //将放大后的点坐标赋值给曲线的控制顶点
```

(7) teapot: 因为茶壶的绘制函数是`glutWireTeapot(size)`, 其中`size`为茶壶的大小, 所以缩放时只需增减`size`即可, 在此笔者缩放时等比例缩放: `size = size * 2`或`size = size * 0.5`。

6、图形旋转:

(1) point: 点无旋转, 略。

(2) line: 在初始化时有提到旋转时需要用到`dir`和`angle`等变量, 此时派上用场。`dir`表示直线的方向向量, 旋转时在此基础上再加30度(旋转一次按30度), 即

```
float temp[2] = {dir[0], dir[1]}
dir[0] = cos(angle)*temp[0] - sin(angle)*temp[1];
dir[1] = sin(angle)*temp[0] + cos(angle)*temp[1];
```

然后将直线的尾端点`end`的坐标按`dir`的方向旋转即可:

```
end.X = begin.X + length * dir[0];
end.Y = begin.Y + length * dir[1];
```

自此, 完成直线的旋转。

(3) polygon: 多边形的旋转利用了直线的旋转基理, 是以第一个顶点为旋转基点, 将除第一个顶点的每个顶点与第一个顶点当作一条直线旋转, 然后重新绘制多边形即可。代码与直线相似, 只不过是加了一层循环:

```
DIR *temp = new DIR [Vertex.size()-1];
for(unsigned int i = 0; i < Vertex.size() - 1; i++)
    /* 记录原来的dir */
for(unsigned int i = 0; i < Vertex.size()-1; i++)
{
    /* 更新dir */
    /* 更新Vertex[i+1]的坐标 */
}
```

(4) ellipse: 椭圆的旋转过程笔者进行了简化, 只有水平和垂直的变化, 在代码实现上是交换`a`和`b`的值。

(5) circle: 因为圆是中心对称图形, 所以旋转没有效果的变化。

(6) curve: 曲线的旋转是通过多边形的变化实现的, 通过将曲线的控制点看作是控制多边形, 然后对其旋转, 再重新计算Bezier曲线的控制点, 即可得到旋转后的曲线。

```
class::Polygon tempP(4, Point); //将曲线的控制顶点看作多边形处理
tempP.Rotate(' '); //旋转, 传入的char参数是要在teapot旋转
```

时要用

```
for(int i = 0; i < 4; i++) //曲线的控制点得到旋转后的坐标
{
    Point[i] = tempP.getVertex(i);
}
```

(7) teapot: 茶壶初始化时为了更好显示茶壶的立体效果, 已经将其旋转了一定角度。此时的旋转只需更改`rotateX`、`rotateY`和`rotateZ`三者中相应的变量就可以实现旋转。对于茶壶的旋转方式笔者实现了三种方式: 第一种是与其他图形一样的要求旋转, 此时的默认旋转方式是绕`x`轴旋转:

`rotateX=10`; 第二种方式是通过键盘按键形式, 通过`a`、`s`、`d`、`w`四个按键控制茶壶绕着`x`轴或者`y`轴旋转: `rotateX`和`rotateY`的变化; 第三种方式是按下`r`键, 茶壶保持一种旋转方向一直旋转, 通过按`e`键停止, 这种效果通过定时器实现(后面详细解释), 此时的旋转方向和默认的旋转方向相同。具

体代码如下：

```
switch (ch)
{
case 'a':
    rotateY-=10; break;
case 's':
    rotateX-=10; break;
case 'd':
    rotateY+=10; break;
case 'w':
    rotateX+=10; break;
default:
    rotateX-=10; break;
}
```

7、图形裁剪：

对于图形的裁剪，不论是哪种图形，笔者是通过绘制一个裁剪矩形来裁剪图形的，在裁剪框的内容被裁剪掉，其余部分保留。具体的实现方式是通过鼠标按下左键移动记录起点和终点两个坐标，并以这两个坐标确定一个裁剪矩形，然后对矩形内部的内容用背景色填充。

```
int left = (c[0].X > c[1].X) ? c[1].X : c[0].X; // 确定左上角和右下角的坐标
int right = (c[0].X > c[1].X) ? c[0].X : c[1].X;
int top = (c[0].Y > c[1].Y) ? c[0].Y : c[1].Y;
int bottom = (c[0].Y > c[1].Y) ? c[1].Y : c[0].Y;
COORD p[] = {{left,top}, {left, bottom}, {right, bottom}, {right, top}};
class::Polygon poly(4,p); //用矩形初始化
poly.Fill(1,1,1); //用白色填充裁剪矩形
```

另外需要说明的是，因为裁剪是不针对于某个图形的性质，所以笔者没有将裁剪设计到每种图形的类中，而是将其放在对图形的管理类MainCtrl中，关于MainCtrl后面说明。

8、图形填充：

(1) point: 点的填充无意义，略

(2) line: 直线的填充也无意义，略。之所以将没有意义的图形也列举在这里，是因为在基类Graphics中将些成员函数定义为纯虚函数，派生类中必须实现，只是以空函数体实现而已。

(3) polygon: 多边形的填充是利用OpenGL的函数glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)来实现，笔者自己想不到一种更加原创的实现方法，对于规则的多边形可以利用数学方法计算来实现填充，但是这里的多边形是不规则的，所以调用了已有的函数。具体的代码如下：

```
glColor3f(r,g,b); //设置填充的颜色
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glBegin(GL_POLYGON);
for(auto i = Vertex.begin(); i != Vertex.end(); i++)
    glVertex2f((*i).X, (*i).Y); //说明填充图形的顶点
glEnd();
```


(4) circle: 圆的填充原理为以圆心为圆心, 半径由0逐渐增大至原有半径画圆填充, 为了一定程度上加快填充速度, 可以加增大线宽和循环变量。具体的实现代码如下:

```
for(int i= 0; i < radiusa-3; i=i+5)
{
    Circle c(center, i);
    c.setLineColor(r,g,b);
    c.setLineWidth(5);
    c.Draw();
}
```

(5) ellipse: 椭圆的填充算法原理为中心和长轴不变, 短轴由0逐渐增加至b, 画相应的椭圆最终实现填充。但是这样的效率有点低, 所以将每次的线宽增大, 然后循环变量也随之增大, 一定程度上加快了填充速度。另外代码中的 $i < b - 4$ 没有增加至b是为了考虑填充的效果。具体的代码如下:

```
int i= 0;
while(i < b - 4)
{
    Ellipse e(center, a-5, i);
    e.setLineColor(r,g,b1);
    e.setLineWidth(5);
    e.Draw();
    if(i < 5)
        i++;
    else
        i += 5;
}
```

(6) curve: 曲线的填充没有意义, 略。

(7) teapot: 茶壶的填充似乎也没有意义, 所以如果茶壶需要填充, 笔者调用OpenGL已有的函数 `glutSolidTeapot(size)` 将其绘画成一个solid teapot (默认为wire teapot)。

9、图形输出:

(1) point: 点的输出内容为点的横纵坐标 + 线宽 + 线条颜色。格式如下:

```
string space = " ";
string p = float2str(pos.X) + space + float2str(pos.Y) + space;
string s = "point " + p + float2str(lineWidth) + space + COLOR2str(lineColor);
```

(2) line: 直线的输出内容为起点的横纵坐标 + 终点的横纵坐标 + 线宽 + 线条颜色。格式如下:

```
string space = " ";
string p = float2str(begin.X) + space + float2str(begin.Y) + space
+ float2str(end.X) + space + float2str(end.Y) + space;
string s = "line " + p + float2str(lineWidth) + space + COLOR2str(lineColor);
```

(3) polygon: 多边形输出的内容是各个顶点的坐标、线宽和线条颜色, 部分代码如下:

```
string space = " ";
string p;
for(auto i = Vertex.begin(); i != Vertex.end(); i++)
```

```

        p += float2str((*i).X) + space + float2str((*i).Y) + space;
        string s = "polygon " + p + float2str(lineWidth) + space + COLOR2str(lineColor);

```

(4) circle: 圆要输出的内容是圆心坐标、半径和线宽、颜色，格式与上面类似。

(5) ellipse: 椭圆的输出内容是中心坐标、长轴、短轴、线宽和颜色，格式与上面类似。

(6) curve: 曲线输出的内容是控制顶点的坐标和线宽、颜色，只允许四个控制顶点，格式如

下：

```

        string space = " ";
        string p1 = float2str(Point[0].X) + space + float2str(Point[0].Y) + space;
        string p2 = float2str(Point[1].X) + space + float2str(Point[1].Y) + space;
        string p3 = float2str(Point[2].X) + space + float2str(Point[2].Y) + space;
        string p4 = float2str(Point[3].X) + space + float2str(Point[3].Y) + space;
        string s = "curve " + p1 + p2 + p3 + p4 + float2str(lineWidth) + space +

```

```

COLOR2str(lineColor);

```

(7) teapot: 茶壶需要显示的信息是茶壶的size、茶壶的线宽和颜色。格式同上，信息之间用空格隔开。

10、图形保存：

图形的保存是指将图形数据保存至文件，保存的图形数据格式同上面的输出格式相同，具体的代码如下：

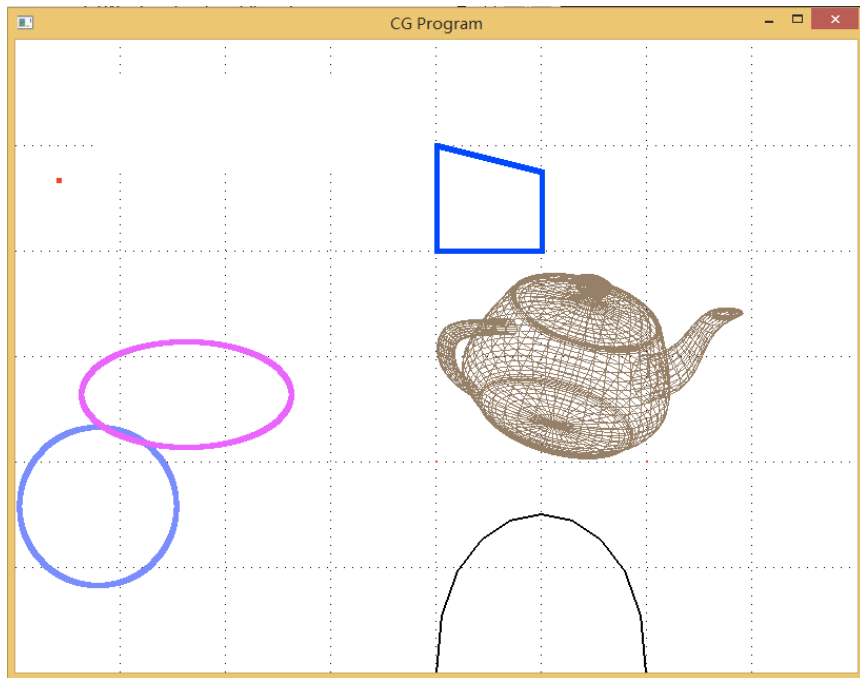
```

        of stream file(filename);
        for(auto i = GraphList.begin(); i != GraphList.end(); i++)
            file<<(*i)->SaveFile()<<endl; //格式与图形数据格式相同
        file.close();
        cout<<"#CGP# 图形数据保存成功！";

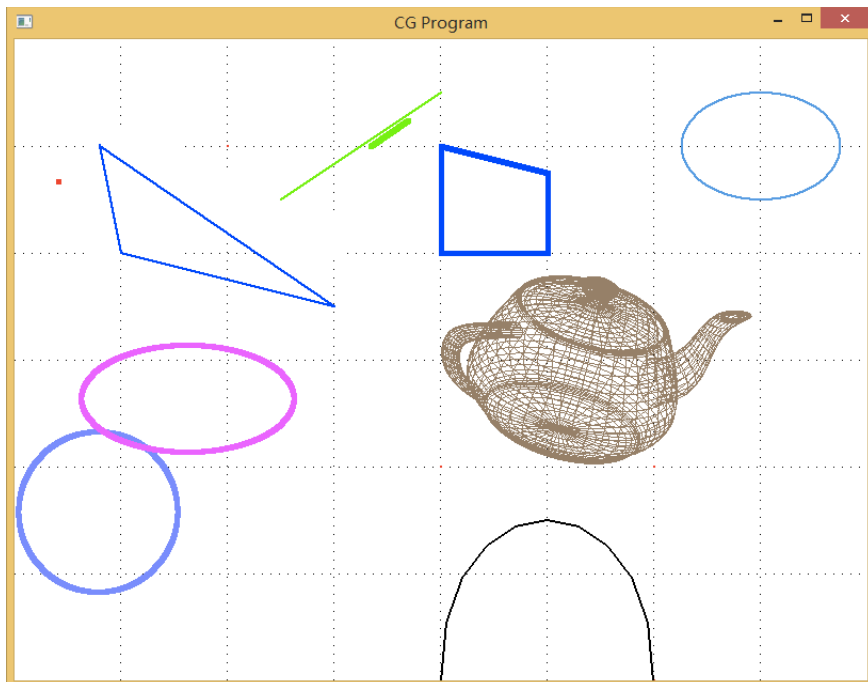
```

四、系统测试

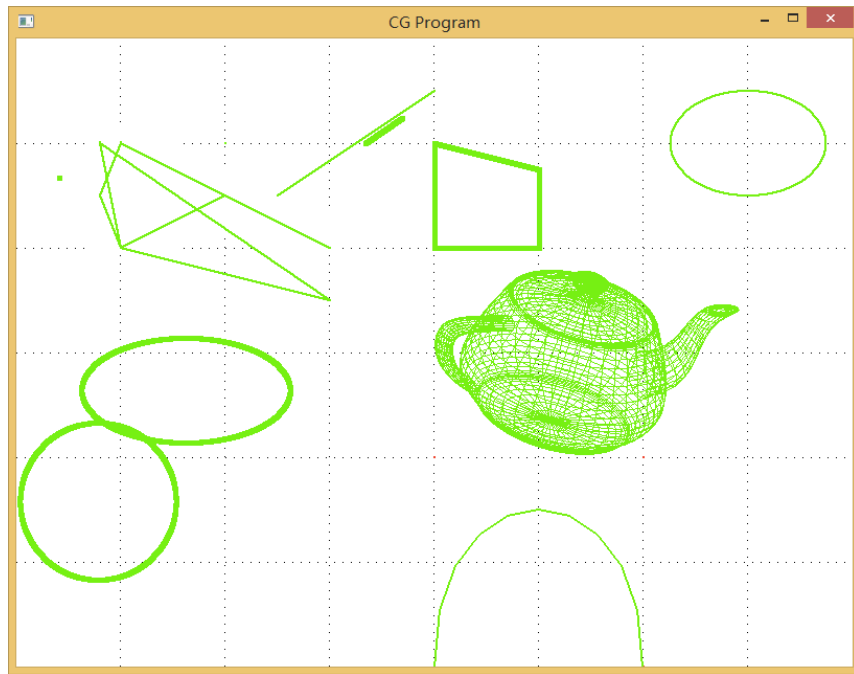
1、开始：（绘制已经初始化的图形）



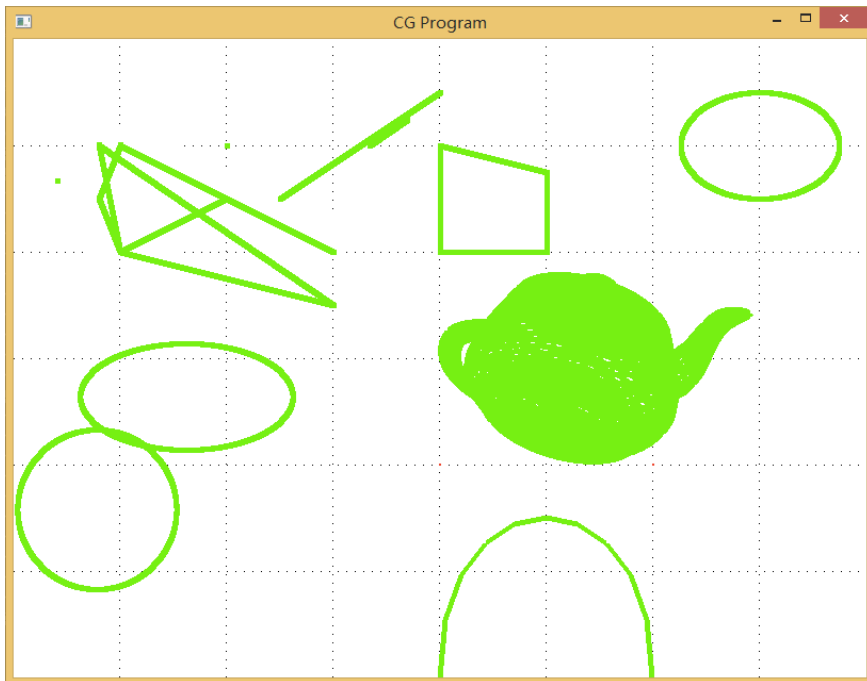
2、新建图形：



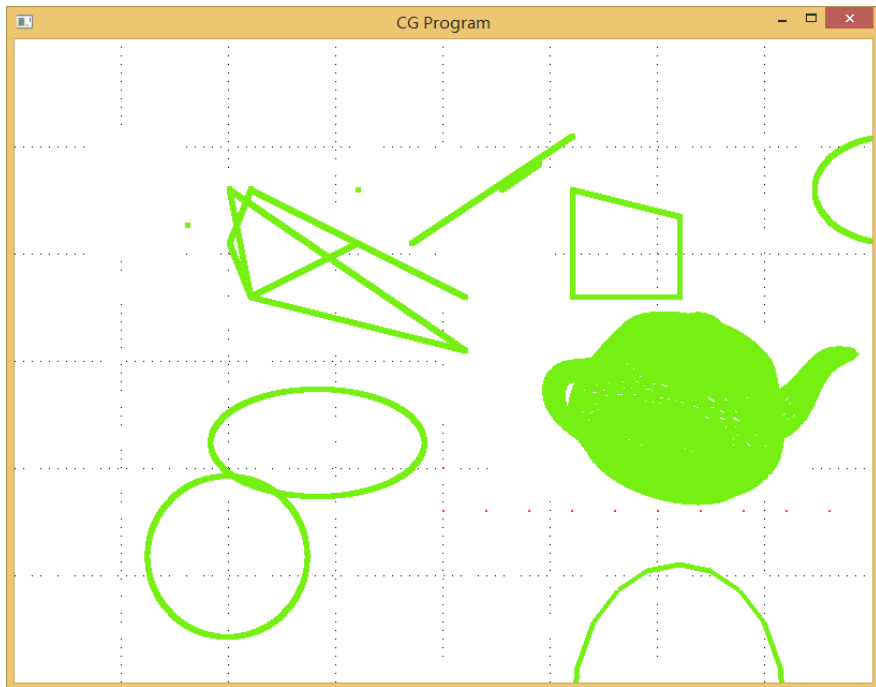
3、设置颜色：（此处设置为绿色，有多种颜色供选）



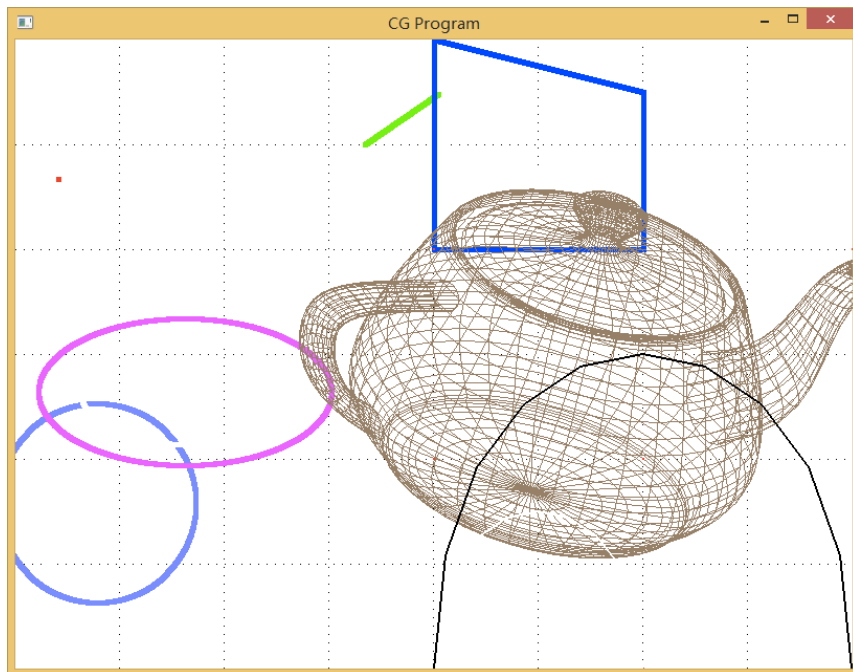
4、设置线宽：（有多种型号的线宽）



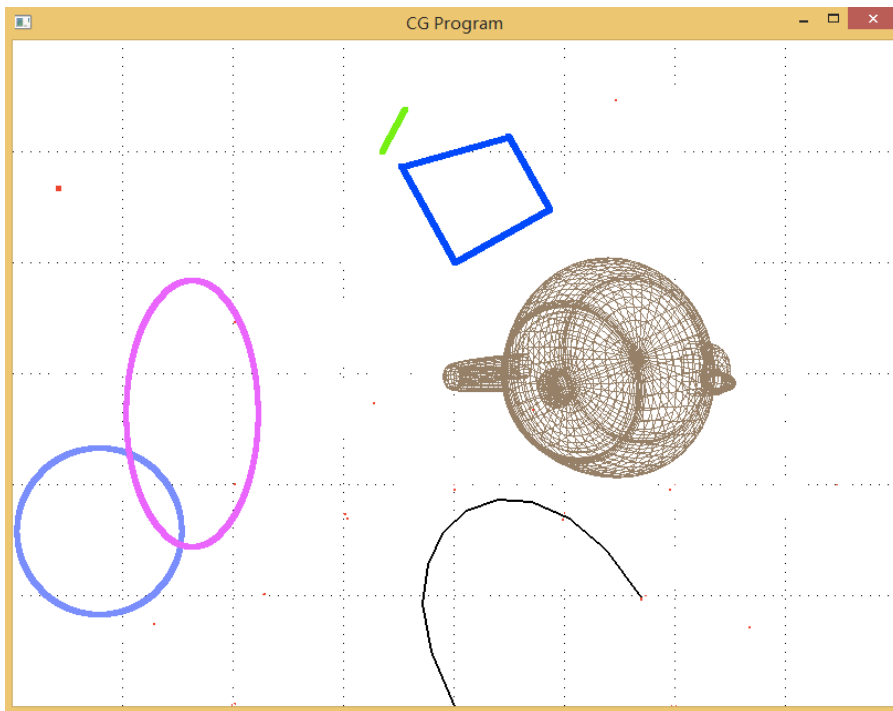
5、移动：（此处为右移）



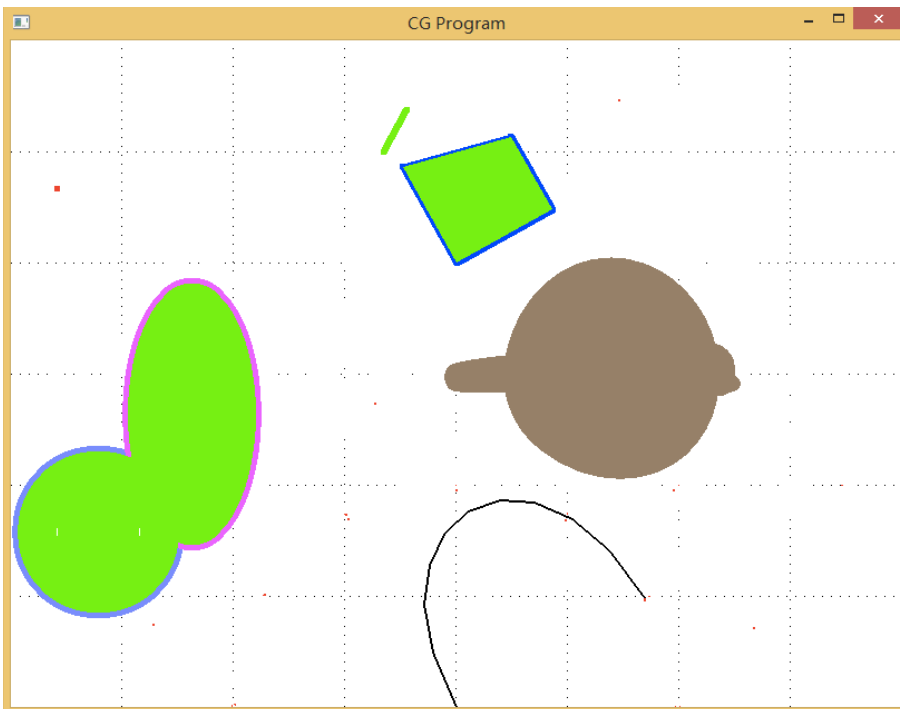
6、缩放图形（此处为放大，缩小略）



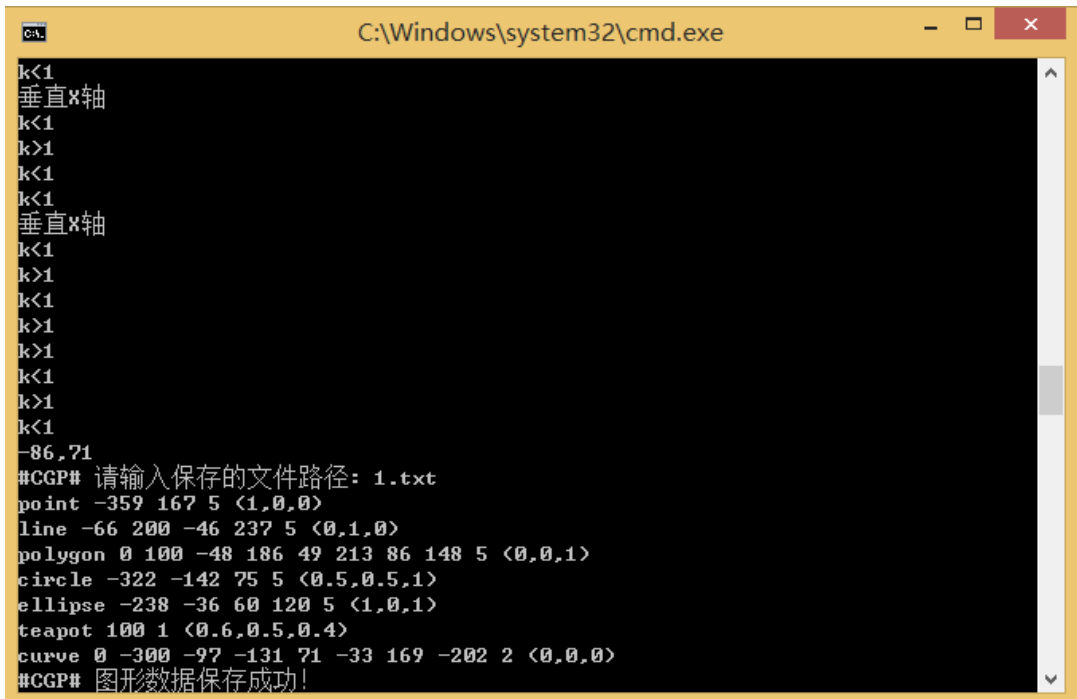
7、旋转图形：



8、填充图形：



9、保存图形：



```

C:\Windows\system32\cmd.exe
k<1
垂直x轴
k<1
k>1
k<1
k<1
垂直x轴
k<1
k>1
k<1
k>1
k>1
k<1
k>1
k<1
-86,71
#CGP# 请输入保存的文件路径: 1.txt
point -359 167 5 <1,0,0>
line -66 200 -46 237 5 <0,1,0>
polygon 0 100 -48 186 49 213 86 148 5 <0,0,1>
circle -322 -142 75 5 <0.5,0.5,1>
ellipse -238 -36 60 120 5 <1,0,1>
teapot 100 1 <0.6,0.5,0.4>
curve 0 -300 -97 -131 71 -33 169 -202 2 <0,0,0>
#CGP# 图形数据保存成功!

```

五、其他的代码说明：

1、关于三维图形茶壶的旋转

笔者对于三维图形的旋转方面实现了按r键可以让茶壶一直保持旋转状态的功能，按e键可以停止旋转。具体的实现原理是，在键盘响应事件中增加对r键的响应。当检测到按下r键时，将一个循环标志loopr置为true，并且OpenGL提供的函数glutTimerFunc(200,timer,0)。意思为每间隔200ms执行一遍timer函数。而timer函数的函数体内容如下：

```

for(auto i = MC.GraphList.begin(); i != MC.GraphList.end(); i++)
{
    if((*i)->graphicsType == "teapot")
    {
        (*i)->Disappear();
        (*i)->rotateX -= 5;
        (*i)->Draw();
    }
}
if (loopr)
    glutTimerFunc(200,timer,0);

```

意思很明显，当进入timer函数后，先执行一次旋转，然后判断loopr是否为真，因为之前已经置为true，所以再此调用glutTimerFunc(200,timer,0)，进入timer函数，以此循环，实现旋转的状态。

如何停止呢？当按下‘e’键时，将loopr置为false即可。

2、裁剪图形时如何记录裁剪矩形的坐标？

利用鼠标事件的响应，当检测到鼠标按下时，记录此时的坐标赋给MainCtrl的c[0]；当检测到鼠标抬起时，再次记录此时的坐标并赋给MainCtrl的c[1]。完成矩形坐标的记录，具体代码如下：

```
if(button == GLUT_LEFT_BUTTON)
{
    if(state == GLUT_DOWN)
    {
        //系统坐标系与相对坐标系的转换
        MC.c[0].X = x-400;
        MC.c[0].Y = -(y-300);
    }
    else if(state == GLUT_UP)
    {
        MC.c[1].X = x-400;
        MC.c[1].Y = -(y-300);
        MC.CutGraph();        //实现裁剪功能的函数
    }
    cout<<x-400<<','<<-(y-300)<<endl;
}
```

3、点击右键弹出命令菜单

笔者利用OpenGL提供的createGlutMenu()函数进行了弹出式菜单的设计，方便用户与该系统的交互，具体实现方式为：在createGlutMenu()函数中添加mennu，然后将其对应的响应事件链接到processMenuEvents()函数中，菜单的部分代码如下：

```
/* 如下是createGlutMenu()中主菜单的部分代码 */
int menu;
menu = glutCreateMenu(processMenuEvents);
glutAddMenuEntry("开始", 开始); //添加一个菜单，并且第二个参数是相应的事件入口
glutAddSubMenu("新建", subMenu1); //添加一个字菜单，subMenu1是内嵌的一个菜单
glutAddSubMenu("颜色", subMenu2);
glutAddSubMenu("线宽", subMenu5);
glutAddSubMenu("平移", subMenu4);
glutAddSubMenu("缩放", subMenu3);
glutAddMenuEntry("旋转", 旋转);
glutAddSubMenu("填充", subMenu6);
glutAddMenuEntry("清空", 清空);
glutAddMenuEntry("保存", 保存);
```

在processMenuEvents()函数中对相应的每种事件加以处理，得到自己的效果就好，最后将菜单的弹出选项与鼠标右键相连，调用glutAttachMenu(GLUT_RIGHT_BUTTON)实现。

4、通过键盘的上下左右键实现图形的平移。

笔者利用keyboardSpecial()函数来响应特殊按键的事件，因为keyboard()函数只能够响应有ASCII码的键盘响应事件，对于特殊的无ASCII码的按键需要用keyboardSpecial()响应。原理很简单，不再赘述，下面是响应的部分代码：

```
switch(key)
{
    case GLUT_KEY_LEFT:
        MC.LeftGraph(); break;
    case GLUT_KEY_UP:
        MC.UpGraph(); break;
    case GLUT_KEY_RIGHT:
        MC.RightGraph(); break;
    case GLUT_KEY_DOWN:
        MC.DownGraph(); break;
}
```

六、系统说明：

1、开发环境：

本次图形学大作业（CG Program）实用的开发工具是VisualStudio 2012，作为Microsoft公司的重要产品，VisualStudio被认为是目前最好的软件开发工具之一，受到软件开发人员的青睐，包括微软在内的许多软件公司都把VisualStudio作为直接产品的开发平台。作为一种程序设计语言，VisualStudio不仅支持传统的软件开发办法，更重要的是它能支持面向对象、可视化的开发风格。因此VisualStudio更应该称作是一个集成开发工具，它提供了软件代码自动生成和可视化的资源编辑功能，在使用VisualStudio开发应用程序的过程中，系统还为我们生成了大量的各种类型的文件。

2、系统的安装与配置：

首先，配置OpenGL的glut和glew工具包。

在OpenGL的官网下载glutdlls37beta.zip，解压这个zip包，会发现其中包含如下几个文件：

glut.h：头文件，复制到D:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\include\gl目录下(需要自己来新建gl目录)

glut.lib,glut32.lib：静态链接库，复制到D:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\lib目录下

glut.dll,glut32.dll：动态链接库，复制到C:\Windows\System32目录下，64位操作系统的，还需要复制到C:\Windows\SysWOW64目录下。

至此，opengl初始环境搭建完成，就可以运行CG Program了。

结束语

这次的CG Program不同于以往的课程设计，它作业量大，历时很长，并且只有主要的目标，没有细节上的要求，完全要由自己去找资料解决在开发过程中遇到的一些困难。

首先，我们做的是个图形管理系统，这使我们对Visual Studio和OpenGL这些非常重要的开发工具有了更进一步的了解，比如如何绘制图形、实现图形的变幻、如何建立菜单并为他们设计事件响应程序等等。

其次，我觉得对这学期学的其他课程也有帮助，例如，这次我用C++的继承机制继承Graphics基类，很好的巩固了高级程序设计课程的知识，对C++这门语言也有了更深刻的理解和认知。

总之，这次历时一个学期的Program让我们学到了很多知识和方法，使我们受益匪浅，在以后的学习工作中我们仍会向这次一样认真地完成接下来的任务。

致谢 在此，笔者向对《计算机图形学》课程教学的孙教授表示衷心感谢，对负责此项作业的两助教表示感谢，以及对像这次大作业提供帮助和支持的同学、博主表示感谢。

References:

- [1] GLUT工具包的下载与安装: <http://www.xiaobao1993.com/31.html>
- [2] OpenGL入门教程: <http://blog.csdn.net/xsckernel/article/details/50158329>
- [3] 中点圆生成算法、中点椭圆生成算法: <http://blog.csdn.net/u012866328/article/details/52607439>
- [4] GLUT菜单: http://blog.csdn.net/xie_zi/article/details/1963383
- [5] Bezier曲线的生成: <http://www.cnblogs.com/opengl/p/3789244.html>
- [6] OpenGL鼠标、键盘事件的应用: http://blog.csdn.net/dream_whui/article/details/39647169
- [7] 《计算机图形学教程》. 孙正兴主编. 机械工业出版社. 2006.8