

计算机图形学实验报告（一）

学 号	姓 名	班 级	报告日期
180400715	杨卓宸	1804102	2020/10/16
实验内容	要求： 1. 创建一个单文档程序 2. 能够交互绘制折线 3. 窗口刷新自动重绘。 思路： 左键按下时，确定折线的一个顶点，鼠标移动的过程中，绘制当前的线段，右键按下时，结束折线的绘制。		
实验目的	1. 熟练掌握 MFC 编程方法 2. 能够绘制简单的二维图形（该实验无曲线） 3. 掌握橡皮筋直线的画法 4. 理解掌握图形刷新重绘的意义及方法 5. 掌握交互式绘图的方法		
实验预备知识	1. 绘图工具类： CPen 类：GDI 画笔，用于画线。默认的画笔用于绘制与一个像素等宽的黑色实线。 CPalette 类：GDI 调色板，包含系统可用的色彩信息，是应用程序和彩色输出设备环境(如显示器)的接口。 CRgn 类：GDI 区域，用于设备环境(通常是窗口)内的区域操作，通常和 CDC 类中与裁剪(clipping)有关的成员函数配合使用。 CPoint 类：CPoint 是对 Windows 结构 POINT 的封装，CPoint 表示屏幕上的一个二维点。 2. 绘图常用函数的使用方法： 画矩形 <code>BOOL Rectangle(int x1, int y1, int x2, int y2);</code> <code>BOOL Rectangle(LPCRECT lpRect);</code> 3. 交互式绘图的方法： 通过添加消息函数 <code>onLButtonDown</code> , <code>onMouseMove</code> , <code>onRButtonDown</code> ，通过获取鼠标指针的坐标信息来获取折线的顶点及绘制方向信息。通过修改 <code>onDraw</code> 函数重绘并保持折线。 4. 如何添加消息处理函数： 打开类视图，右键选中视类，点击属性，打开属性窗口，点击消息按钮，在相应的响应处添加消息处理函数。 5. 重绘的意义与方法： 将已完成的图信息保存下来，重新绘制。		

实验过程描述	<p>1. 首先在 view 类头文件中定义：</p> <pre>public: CPoint startPoint; //线段起始点 CPoint oldPoint; //线段中止点 BOOL isDrawing = FALSE; //绘图状态标识 std::vector<std::pair<CPoint, CPoint>> lines; //线段点存储集合 public: afx_msg void OnLButtonDown(UINT nFlags, CPoint point); //鼠标左键按下消息处理函数 afx_msg void OnMouseMove(UINT nFlags, CPoint point); //鼠标移动绘图跟随处理函数 afx_msg void OnRButtonDown(UINT nFlags, CPoint point); //鼠标右键按下消息处理函数</pre> <p>2. 补充添加 3 个消息函数的源码：</p> <pre>void Ctest1View::OnLButtonDown(UINT nFlags, CPoint point) { // TODO: 在此添加消息处理程序代码和/或调用默认值 if (isDrawing) { CDC* pDC = GetDC(); //起点 pDC->MoveTo(startPoint); //划线, 终点 pDC->LineTo(point); ReleaseDC(pDC); lines.push_back(std::make_pair(startPoint, point)); } else { isDrawing = TRUE; } startPoint = point; oldPoint = point; CView::OnLButtonDown(nFlags, point); } void Ctest1View::OnMouseMove(UINT nFlags, CPoint point) { // TODO: 在此添加消息处理程序代码和/或调用默认值 if (isDrawing) { CDC* pDC = GetDC(); //起点 pDC->SetROP2(R2_NOT); pDC->MoveTo(startPoint);</pre>
--------	--

```
pDC->LineTo(oldPoint);
pDC->MoveTo(startPoint);
//划线, 终点
pDC->LineTo(point);
oldPoint = point;
ReleaseDC(pDC);
}
CView::OnMouseMove(nFlags, point);
}

void Ctest1View::OnRButtonDown(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    CPoint pt;
    GetCursorPos(&pt);
    lines.push_back(std::make_pair(startPoint, point));
    isDrawing = FALSE;
    CView::OnRButtonDown(nFlags, point);
}
```

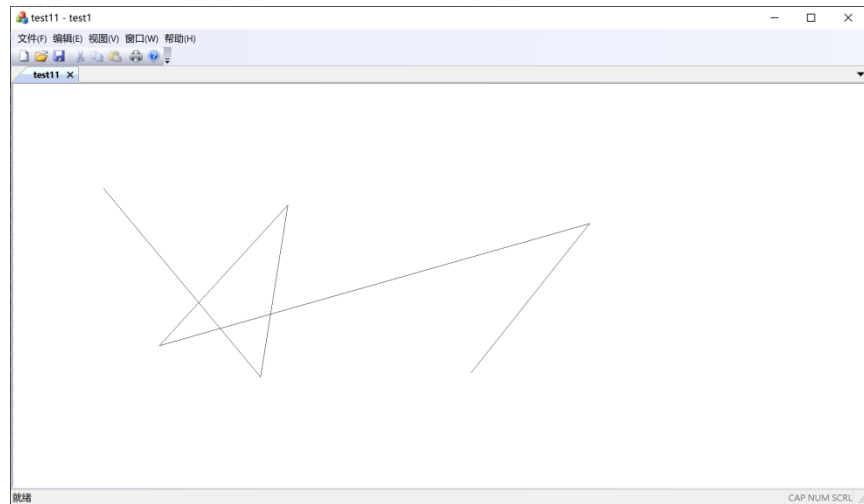
3. 添加 OnDraw 函数中重绘的代码:

```
void Ctest1View::OnDraw(CDC* pDC)
{
    Ctest1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: 在此处为本机数据添加绘制代码
    for (auto line : lines)
    {
        //起点
        pDC->MoveTo(line.first);
        //划线, 终点
        pDC->LineTo(line.second);
    }
}
```

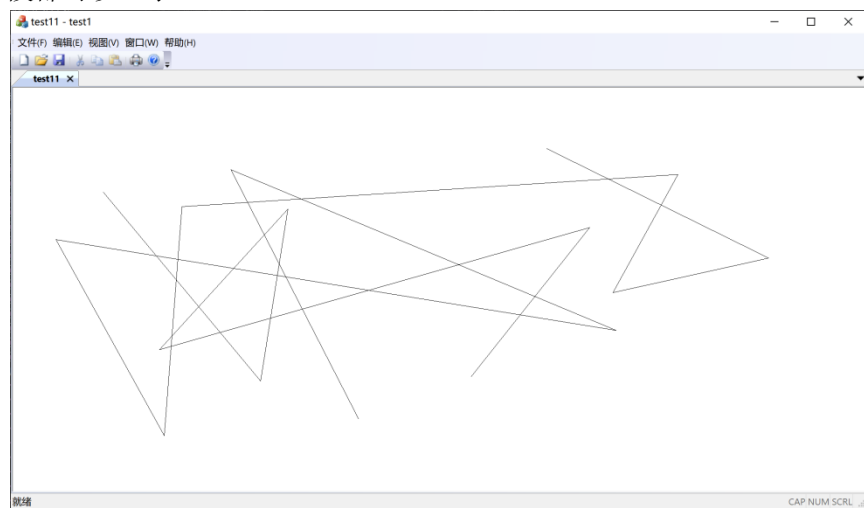
4. 最后进行调试运行。

实验结果

1. 首先按下左键确定起始点，之后依次按下四次左键确定中间四个顶点，最后按下右键结束绘画的同时，将该点作为终点，因此共 6 个顶点。



2. 第一次绘制完成后，可再次绘制下一系列线段。并且可以保持所有线段都可以显示。



3. 窗口拉伸刷新后，线段仍能保持重绘。




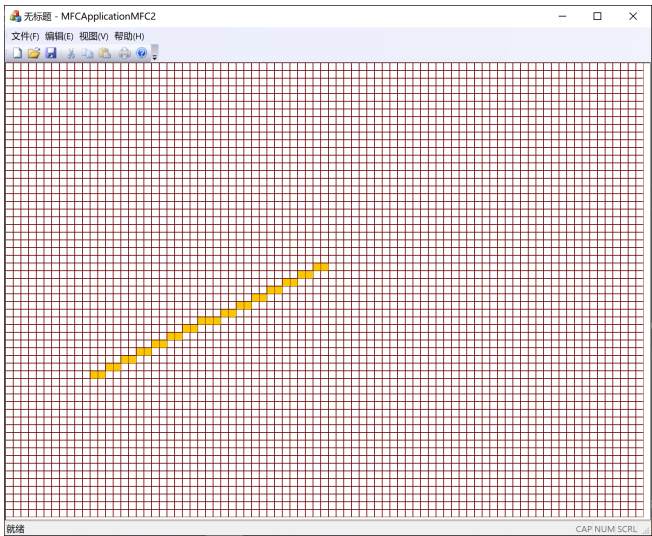
实验当中问题及解决方法	<p>1、起初不熟悉 MFC 的操作：上网查找简单的操作例子后，较为熟悉 MFC 的操作。</p> <p>2、VS2019 默认无法找到类视图，导致无法进行实验：通过不断摸索，发现应该在视图模块里面勾选显示视图类。</p> <p>3、上网查询相关重绘信息后发现晦涩难懂：仔细阅读课程 PPT，最后成功使用一个简单的画折线函数完成重绘。</p> <p>4、右键单击时最后一段折线会消失，因为并没有进行存储，加入存储语句后，最后一段折线可以正常显示。</p> <p>5、鼠标跟随绘图的功能比较困难，通过查找，了解到要通过不断反色处理上条线段来达到跟随效果，最终通过 pDC->SetROP2(R2_NOT)实现。</p>			
成绩（教师打分）	优秀	良好	及格	不及格

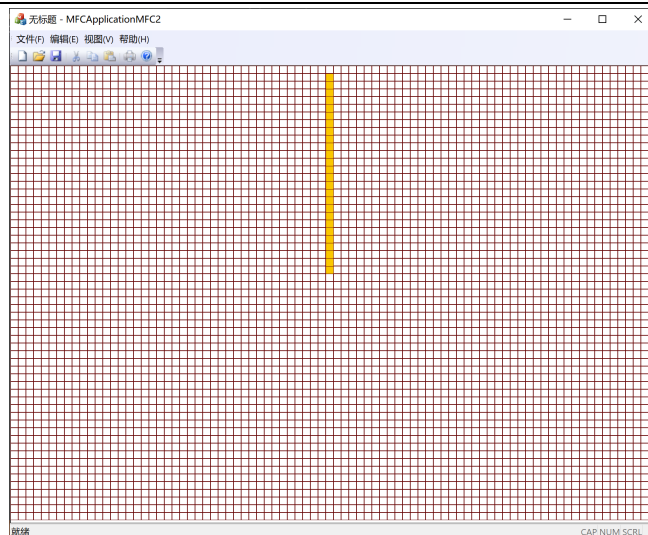
计算机图形学实验报告（二）

学 号	姓 名	班 级	报告日期
180400715	杨卓宸	1804102	2020/10/23
实验内容	实现直线段生成的两种方法： 1) 数值微分法 2) 中点 Bresenham 算法		
实验目的	1. 下列网格上，每一个网格表示一个像素。 2. 用户用鼠标点击两个像素点，当一个像素被点击，将该像素显示出来。 3. 当两个像素都确定后，利用直线段的绘制算法绘制两个像素之间的一条直线段。		
实验预备知识	<p>1. 数值微分法</p> <p>已知过端点 p_0, p_1 的直线段，则直线斜率为 $k = (y_1 - y_0) / (x_1 - x_0)$，要在输出设备上显示该直线段，需要计算出经过该直线段 p_0p_1 上所有像素点的集合。</p> <p>根据几何知识，我们需要将 x 从左端点 x_0 开始，向右端点 x_1 步进，步长 = 1 个像素，将 x 代入直线方程 $y = kx + b$ 计算出相应的坐标即可；x 的取值是离散的，但是计算出来的 y 值不一定是整数，所以需要对 y 进行四舍五入，像素点 $[x, \text{round}(y)]$ 作为当前点的坐标。</p> <p>需要对斜率先进行判断处理，若 $k > 1$ 则以 y 为单位++求 x，$k < 1$ 则以 x 为单位++求 y。</p> <p>2. 中点 Bresenham 算法</p> <p>已知过端点 p_0, p_1 的直线段，定义当前点坐标 (x, y)，定义点偏差判别式为 d，定义直线斜率为 k，$x = x_0$，$y = y_0$，计算 $d = 0.5 - k$，$k = (y_1 - y_0) / (x_1 - x_0)$。绘制点 (x, y)，判断 d 的符号，若 $d < 0$，则 (x, y) 更新为 $(x+1, y+1)$，d 更新为 $d+1-k$，否则 (x, y) 更新为 $(x+1, y)$，d 更新为 $d-k$。如果当前点 x 小于 x_1，重复上述步骤，否则结束。</p> <p>同样需要先判断 k 的大小，若 $k < 1$ 则以 x 为基础计算 y，若 $k > 1$ 则以 y 为基础计算 x。</p>		
实验过程描述	<pre>//数值微分法 void CMFCApplicationMFC2View::drawLine(CPoint startp, CPoint endp, CDC * pDC) { CClientDC dc(this); float y = startp.y, x = startp.x, l, k, dx, dy; dx = endp.x - startp.x; dy = endp.y - startp.y; k = dy / dx; l = dx / dy; if (abs(k)<1) for (int x = startp.x; startp.x < endp.x ? x <= endp.x : x >= endp.x; startp.x < endp.x ? x++ : x--){</pre>		

```
        fillOnePixel(x, round(y), &dc);
        (startp.x < endp.x) ? y += k : y -= k;
    }
    else
        for (int y = startp.y; startp.y < endp.y ? y <= endp.y : y >=
endp.y; startp.y < endp.y ? y++ : y--){
            fillOnePixel(round(x), y, &dc);
            (startp.y < endp.y) ? x += 1 : x -= 1;
        }

//中点 Bresenham 算法
void CMFCApplicationMFC2View::drawLine(CPoint startp, CPoint endp,
CDC * pDC)
{
    CClientDC dc(this);
    float x, y, d, k, l, dx, dy;
    x = startp.x;
    y = startp.y;
    dx = endp.x - startp.x;
    dy = endp.y - startp.y;
    k = dy / dx;
    l = dx / dy;
    if (abs(k) < 1) {
        k > 0 ? d = 0.5 - k : d = k + 0.5;
        for (int x = startp.x; startp.x < endp.x ? x <= endp.x : x >=
endp.x; startp.x < endp.x ? x++ : x--){
            fillOnePixel(x, round(y), &dc);
            if (d < 0){
                startp.y < endp.y ? y++ : y--;
                d += 1 - abs(k);
            }
            else{
                d -= abs(k);
            }
        }
    }
    else {
        l > 0 ? d = 0.5 - l : d = l + 0.5;
        for (int y = startp.y; startp.y < endp.y ? y <= endp.y : y >=
endp.y; startp.y < endp.y ? y++ : y--){
            fillOnePixel(round(x), y, &dc);
            if (d < 0){
                startp.x < endp.x ? x++ : x--;
                d += 1 - abs(l);
            }
        }
    }
}
```

	<pre> } else{ d -= abs(l); } } } } } </pre>
<p>实验结果</p>	<p>1. 数值分析法结果图</p>  <p>第四象限 $k < 1$ 情况</p>  <p>第三象限 $k < 1$ 情况</p>

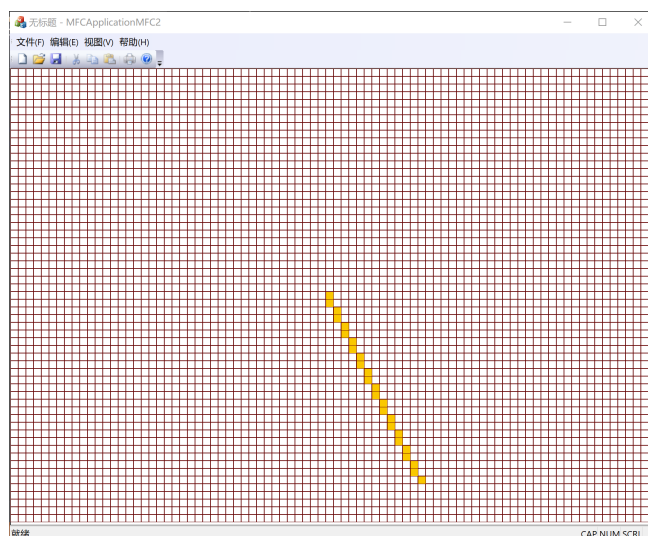


竖直情况

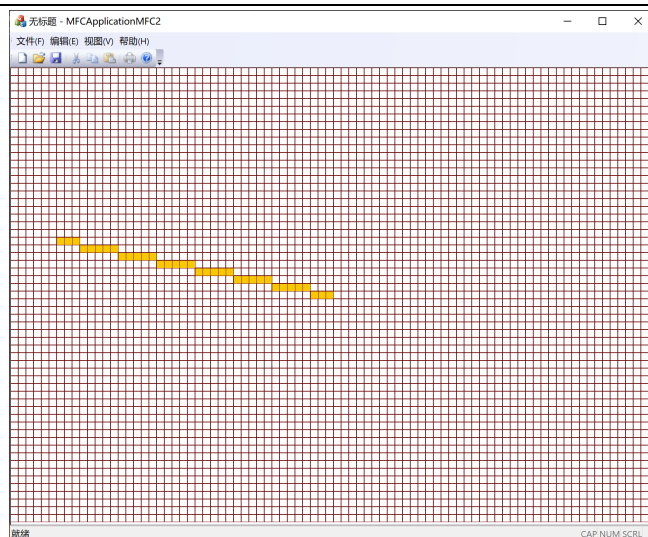


第二象限 $k > 1$ 情况

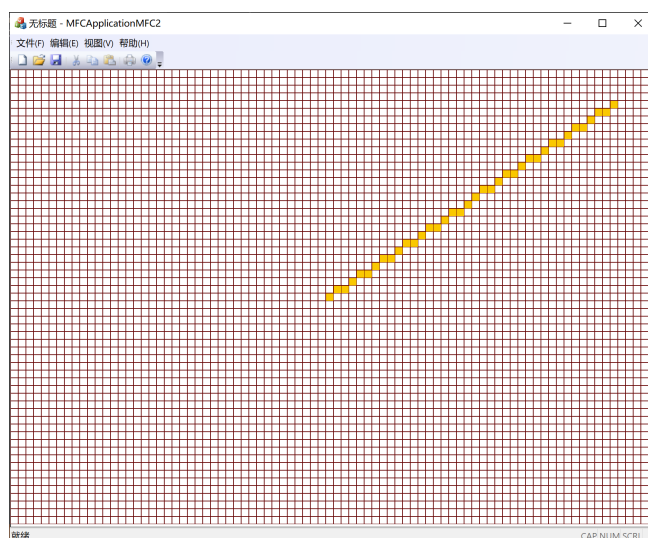
2. 中点 Bresenham 算法结果图



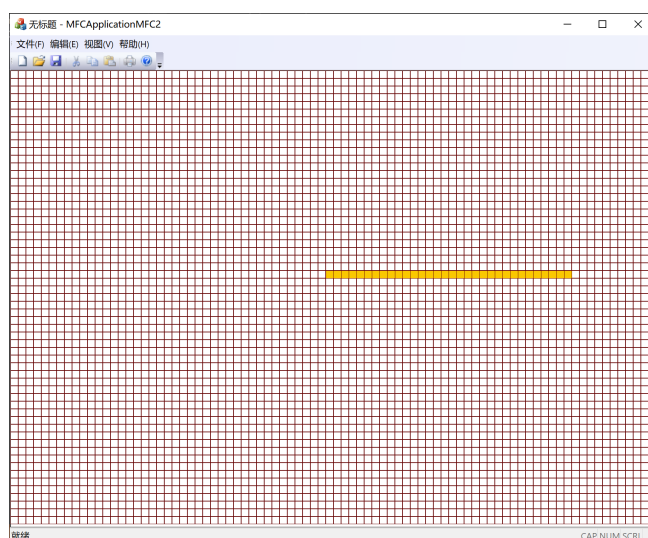
第四象限 $k > 1$ 情况



第二象限 $k < 1$ 情况



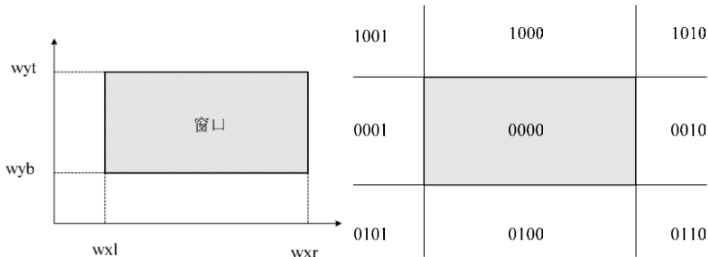
第一象限 $k < 1$ 情况



水平情况

实验当中问题及解决方法	<p>1、x 为离散值，y 的值计算出来不一定是整数，所以 y 取 $\text{round}(y)$。</p> <p>2、由于 xy 点分布在 4 个象限，所以需要判断 xy 的正负来进行加减操作。</p> <p>3、由于斜率不同，计算的基础 dx 或 dy 不同，所以还需要判断 $k>1/k<1$，进行计算。</p> <p>4、多段代码是类似并且重复的，所以多调用三目运算符可以节约大量重复代码的起始判断。</p>			
成绩（教师打分）	优秀	良好	及格	不及格

计算机图形学实验报告（三）

学 号	姓 名	班 级	报告日期
180400715	杨卓宸	1804102	2020/10/29
实验内容	利用 Cohen-Sutherland 直线裁剪算法实现直线段的剪切 实现直线段裁剪的中点分割算法		
实验目的	1. 建立一个 MFC 单文档程序，在窗口中间部位绘制一个矩形用来模拟窗口。 2. 通过鼠标交互绘制直线段（橡皮筋直线，当按两次鼠标左键确定一条直线段）。 3. 当按下鼠标右键时利用直线段裁剪算法对绘制的直线段进行裁剪，保留并显示裁剪之后的直线段。		
实验预备知识	<p>Cohen-Sutherland 直线裁剪算法：</p> <p>假设裁剪窗口是标准矩形，由上（$y=wyt$）、下（$y=wyb$）、左（$x=wxl$）、右（$x=wxr$）四条边组成，如下图所示。</p> <p>Cohen-Sutherland 直线裁剪算法：</p> <p>假设裁剪窗口是标准矩形，由上（$y=wyt$）、下（$y=wyb$）、左（$x=wxl$）、右（$x=wxr$）四条边组成，如下图所示。延长窗口四条边形成 9 个区域。根据被裁剪直线的任一端点 $P(x, y)$ 所处的窗口区域位置，可以赋予一组 4 位二进制区域码 C4C3C2C1。</p>  <p>编码定义规则：</p> <p>第一位 C1：若端点位于窗口之左侧，即 $X < W_{xl}$，则 $C1=1$，否则 $C1=0$。</p> <p>第二位 C2：若端点位于窗口之右侧，即 $X > W_{xr}$，则 $C2=1$，否则 $C2=0$。</p> <p>第三位 C3：若端点位于窗口之下侧，即 $Y < W_{yb}$，则 $C3=1$，否则 $C3=0$。</p> <p>第四位 C4：若端点位于窗口之上侧，即 $Y > W_{yt}$，则 $C4=1$，否则 $C4=0$。</p> <p>裁剪步骤：</p> <ol style="list-style-type: none"> 若直线的两个端点的区域编码都为 0，即 $RC1 RC2=0$（二者按位相或的结果为 0，即 $RC1=0$ 且 $RC2=0$），说明直线两端点都在窗口内，应“简取”。 若直线的两个端点的区域编码都不为 0，即 $RC1\&RC2\neq 0$（二者按位相与的结果不为 0，即 $RC1\neq 0$ 且 $RC2\neq 0$），说明直线的两个端点都在窗口外，应“简弃”。 若直线既不满足“简取”也不满足“简弃”的条件，直线段必然与窗口相交，需要计算直线与窗口边界的交点。交点将直线分为两段，其中一段完全位于窗口外，可“简弃”。对另一段赋予交点处的区域编码， 		

	<p>再次测试，再次求交，直至确定完全位于窗口内的直线段为止。</p> <p>4. 实现时，一般按固定顺序左（$x=wx_l$）、右（$x=wx_r$）、下（$y=wy_b$）、上（$y=wy_t$）求解窗口与直线的交点。</p> <p>延长窗口四条边形成 9 个区域。根据被裁剪直线的任一端点 $P(x, y)$ 所处的窗口区域位置，可以赋予一组 4 位二进制区域码 $C4C3C2C1$。</p> <p>编码定义规则：</p> <p>第一位 $C1$：若端点位于窗口之左侧，即 $X < Wx_l$，则 $C1=1$，否则 $C1=0$。</p> <p>第二位 $C2$：若端点位于窗口之右侧，即 $X > Wx_r$，则 $C2=1$，否则 $C2=0$。</p> <p>第三位 $C3$：若端点位于窗口之下侧，即 $Y < Wy_b$，则 $C3=1$，否则 $C3=0$。</p> <p>第四位 $C4$：若端点位于窗口之上侧，即 $Y > Wy_t$，则 $C4=1$，否则 $C4=0$。</p> <p>裁剪步骤：</p> <ol style="list-style-type: none"> 1. 若直线的两个端点的区域编码都为 0，即 $RC1 RC2=0$（二者按位相或的结果为 0，即 $RC1=0$ 且 $RC2=0$），说明直线两端点都在窗口内，应“简取”。 2. 若直线的两个端点的区域编码都不为 0，即 $RC1\&RC2\neq 0$（二者按位相与的结果不为 0，即 $RC1\neq 0$ 且 $RC2\neq 0$，即直线位于窗外的同一侧，说明直线的两个端点都在窗口外，应“简弃”。 3. 若直线既不满足“简取”也不满足“简弃”的条件，直线段必然与窗口相交，需要计算直线与窗口边界的交点。交点将直线分为两段，其中一段完全位于窗口外，可“简弃”。对另一段赋予交点处的区域编码，再次测试，再次求交，直至确定完全位于窗口内的直线段为止。 4. 实现时，一般按固定顺序左（$x=wx_l$）、右（$x=wx_r$）、下（$y=wy_b$）、上（$y=wy_t$）求解窗口与直线的交点。
实验过程描述	<p>1. Cohen-Sutherland 算法</p> <pre>void CTestView::CohenSutherland()//Cohen—Sutherland 算法 { BOOL isChanged; double x, y; RC0 = codePoint(Pointx[0], Pointy[0]); RC1 = codePoint(Pointx[1], Pointy[1]); while (TRUE) { isChanged = FALSE; if (0 == (RC0 RC1)) { return; } else if (0 != (RC0 & RC1)) { Pointx[0] = Pointy[0] = Pointx[1] = Pointy[1] = 0; return; } else { </pre>

```
if (0 == RCO)
//如果 P0 点在窗口内，交换 P0 和 P1, 保证 p0 点在窗口外
{
    //交换点的坐标值
    double cPointx, cPointy;
    cPointx = Pointx[0]; cPointy = Pointy[0];
    Pointx[0] = Pointx[1]; Pointy[0] = Pointy[1];
    Pointx[1] = cPointx; Pointy[1] = cPointy;
    //交换点的编码值
    unsigned int TRC;
    TRC = RCO; RCO = RC1; RC1 = TRC;
}
//按左、右、下、上的顺序裁剪
if (RCO & 1)//P0 点位于窗口的左侧
{
    x = wxl;//求交点 y
    y = Pointy[0] + (Pointy[1] - Pointy[0])*(x -
Pointx[0]) / (Pointx[1] - Pointx[0]);
    Pointx[0] = x; Pointy[0] = y;
    isChanged = TRUE;
    RCO = codePoint(Pointx[0], Pointy[0]); RC1 =
codePoint(Pointx[1], Pointy[1]);
}
if (RCO & 2)//P0 点位于窗口的右侧
{
    x = wxr;//求交点 y
    y = Pointy[0] + (Pointy[1] - Pointy[0])*(x -
Pointx[0]) / (Pointx[1] - Pointx[0]);
    Pointx[0] = x; Pointy[0] = y;
    isChanged = TRUE;
    RCO = codePoint(Pointx[0], Pointy[0]); RC1 =
codePoint(Pointx[1], Pointy[1]);
}
if (RCO & 4)//P0 点位于窗口的下侧
{
    y = wyb;//求交点 x
    x = Pointx[0] + (Pointx[1] - Pointx[0])*(y -
Pointy[0]) / (Pointy[1] - Pointy[0]);
    Pointx[0] = x; Pointy[0] = y;
    isChanged = TRUE;
    RCO = codePoint(Pointx[0], Pointy[0]); RC1 =
codePoint(Pointx[1], Pointy[1]);
}
if (RCO & 8)//P0 点位于窗口的上侧
```

```
{
    y = wyt; //求交点 x
    x = Pointx[0] + (Pointx[1] - Pointx[0]) * (y -
Pointy[0]) / (Pointy[1] - Pointy[0]);
    Pointx[0] = x; Pointy[0] = y;
    isChanged = TRUE;
    RC0 = codePoint(Pointx[0], Pointy[0]); RC1 =
codePoint(Pointx[1], Pointy[1]);
}
if (FALSE == isChanged)
{
    return;
}
}
```




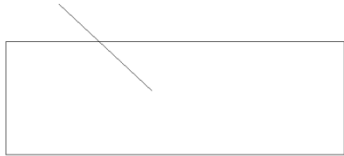
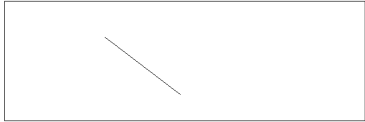
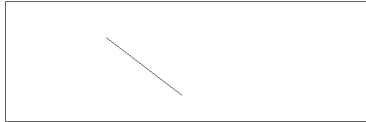


2. 直线段裁剪的中点分割算法

```
void CTestView::midLineClip()
{
    RC0 = codePoint(Pointx[0], Pointy[0]);
    RC1 = codePoint(Pointx[1], Pointy[1]);
    double p1_x, p1_y, p2_x, p2_y;
    p1_x = Pointx[0];
    p1_y = Pointy[0];
    p2_x = Pointx[1];
    p2_y = Pointy[1];
    if (0 == (RC0 | RC1))
    {
        return;
    }
    //存在一个点在裁剪框内
    else if (0 != (RC0 & RC1))
    {
        Pointx[0] = Pointy[0] = Pointx[1] = Pointy[1] = 0;
        return;
    }
    else
    {
        if (RC0 == 0)
            findIntersection(p1_x, p1_y, p2_x, p2_y);
    }
}
```

```
        else if (RC1 == 0)
            findIntersection(p1_x, p1_y, p2_x, p2_y);
        else
        {
            Pointx[0] = (p1_x + p2_x) / 2;
            Pointy[0] = (p1_y + p2_y) / 2;
            midLineClip();
        }
    }
}

//找出 p1 和 p2 之间边界的交点
double CTestView::findIntersection(double p1_x, double p1_y, double
p2_x, double p2_y)
{
    double p_x, p_y;
    p_x = (p1_x + p2_x) / 2;
    p_y = (p1_y + p2_y) / 2;
    if (distance(p_x, p_y, p1_x, p1_y) < 1.5)
        return p_x;
    RC0 = codePoint(Pointx[0], Pointy[0]);
    RC1 = codePoint(Pointx[1], Pointy[1]);
    RC = codePoint(p_x, p_y);
    if (RC0 == 0)
        if (p_x == 0)
            return findIntersection(p_x, p_y, p2_x, p2_y);
        else
            return findIntersection(p1_x, p1_y, p_x, p_y);
    if (RC1 == 0)
        if (p_x == 0)
            return findIntersection(p_x, p_y, p1_x, p1_y);
        else
            return findIntersection(p2_x, p2_y, p_x, p_y);
    return p_x;
}

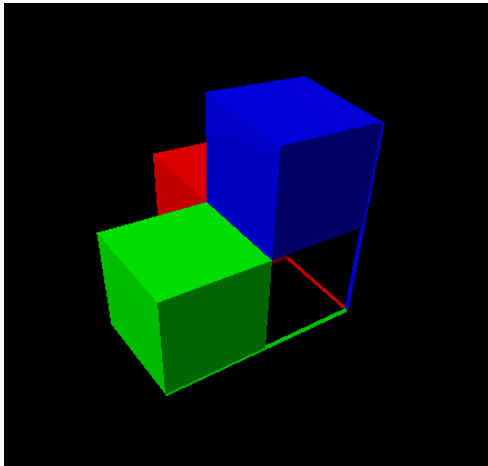
double distance(double p1_x, double p1_y, double p2_x, double p2_y)
{
    double result;
    result = sqrt((p1_x - p2_x) * (p1_x - p2_x) + (p1_y - p2_y) * (p1_y
- p2_y));
    return result;
}
```


实验结果	<div>1. 直线与边框有两个交点</div> <div></div> <div>2. 直线与边框有一个交点</div> <div></div> <div>3. 直线在边框内部</div> <div></div> <div>4. 直线在边框外部</div> <div></div>			
实验当中问题及解决方法	<p>问题：利用鼠标交互式画完直线后，按下鼠标右键进行剪切却没有任何变化。</p> <p>解决：没有先对之前画的直线进行清空，之前的直线将新的直线段覆盖了，因此先对原来直线段进行清空，利用一个简单的方法，即用白色的画笔重新绘制一遍将原来的覆盖掉。</p>			
成绩（教师打分）	优秀	良好	及格	不及格

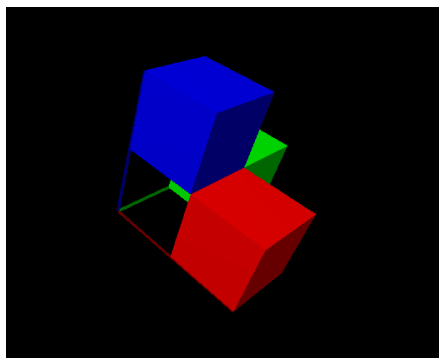
计算机图形学实验报告（四）

学 号	姓 名	班 级	报告日期
180400715	杨卓宸	1804102	2020/11/6
实验内容	<p>(1) 绘制三个立方体，位置如图所示。要求绘制出坐标系的 3 个坐标轴。</p> <p>(2) 三个立方体一起沿着坐标轴(x 或 y 或 z)旋转起来。</p> <p>说明：旋转可以通过设置时钟事件来实现，提供的程序里已经实现旋转的功能。打开菜单： 编辑-rotate about x 等即可旋转，可以参考其实现。</p> <p>(3) 三个立方体分别沿 X 轴，Y 轴和 Z 轴旋转。</p>		
实验目的	<p>1. 熟练掌握 OpenGL 编程的基本方法。</p> <p>2. 学会图形绘制，使用基本图元：点、线、多边形来绘制三维图 形和场景。</p> <p>3. 变换操作，从不同的角度和距离观察物体。</p>		
实验预备知识	<p>1. OpenGL 提供了 3 个函数库： 基本库 实用库 辅助库 基本库是 OpenGL 的核心函数库，提供了 115 个 函数，函数以 “gl” 为前缀。</p> <p>主要功能包括物体描述、平移、旋转、缩放、 光照、纹理、材质、像素、位图、文字处理等。</p> <p>2. OpenGL 中的变换 OpenGL 中的变换矩阵： <code>glRotatef(2, 2, 3); //变换矩阵</code> <code>M draw_triangle(); //定义点 P</code> OpenGL 中的多种变换（几何变换、投影变换等） 是由矩阵的乘积实现的。 矩阵操作模式： <code>void glMatrixMode(GLenum mode);</code> 参数取值 GL_PROJECTION 表示该语句后面的变换是针对投影变换的。</p> <p>3. OpenGL 中的矩阵堆栈 矩阵堆栈的使用： <code>void glPushMatrix(): 复制栈顶矩阵并压入栈。</code> <code>void glPopMatrix(): 移除栈顶矩阵。</code></p>		

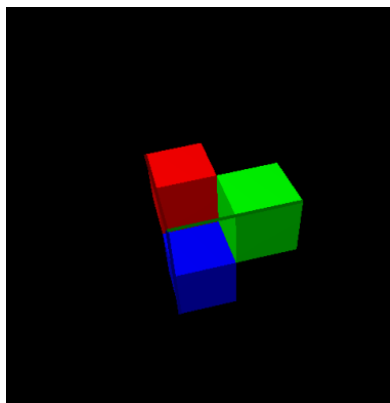
实验过程描述	<p>1. 仔细阅读实验材料，结合 ppt，理解明白每个函数的作用。</p> <p>2. 补充 RenderScene() 函数：</p> <p>画出坐标系：</p> <pre>glLineWidth(4.0f); glBegin(GL_LINES); glColor3f(0, 0, 1); glVertex3f(0, 0, 0); glVertex3f(2, 0, 0); glColor3f(1, 0, 0); glVertex3f(0, 0, 0); glVertex3f(0, 2, 0); glColor3f(0, 1, 0); glVertex3f(0, 0, 0); glVertex3f(0, 0, 2); glEnd();</pre> <p>通过判断不同的按键信息来画方块：</p> <p>首先是三个立方体看做一个整体的情况：</p> <p>if (m_rotate_axis == XX) //如果想要按 x 轴旋转</p> <pre>{ glRotatef(m_angle, 1, 0, 0); }</pre> <p>else if (m_rotate_axis == YY) //如果想要按 y 轴旋转</p> <pre>{ glRotatef(m_angle, 0, 1, 0); }</pre> <p>else if (m_rotate_axis == ZZ) //如果想要按 z 轴旋转</p> <pre>{ glRotatef(m_angle, 0, 0, 1); }</pre> <p>else if (m_rotate_axis == NOROTATE) //默认状态，直接按照一定角度画出三个立方体即可</p> <pre>{ glRotatef(315, 0, 1, 0); glRotatef(40, 0, 1, 1); glRotatef(310, 0, 1, -1); glRotatef(350, 1, 1, 0); glRotatef(355, 0, 0, 1); }</pre> <p>假设按下的是想要三个立方体分别旋转，则需要一步一步地画三个立方体，达到不让三个立方体朝一个方向转的目的：</p> <pre>glPushMatrix(); glTranslatef(0, 1, 0); glColor3f(1, 0, 0); if (m_rotate_axis == CYLINDER)</pre>
--------	--

	<pre>{ glRotatef(m_angle, 0, 1, 0); } draw_cube(); glPopMatrix(); glPushMatrix(); glTranslatef(1, 0, 0); glColor3f(0, 0, 1); if (m_rotate_axis == CYLINDER) { glRotatef(m_angle, 1, 0, 0); } draw_cube(); glPopMatrix(); glPushMatrix(); glTranslatef(0, 0, 1); glColor3f(0, 1, 0); if (m_rotate_axis == CYLINDER) { glRotatef(m_angle, 0, 0, 1); } draw_cube(); glPopMatrix();</pre> <p>3. 不断调试，通过调整角度使效果最佳。</p>
实验结果	<p>1. 直接运行程序画出的立方体：</p> 

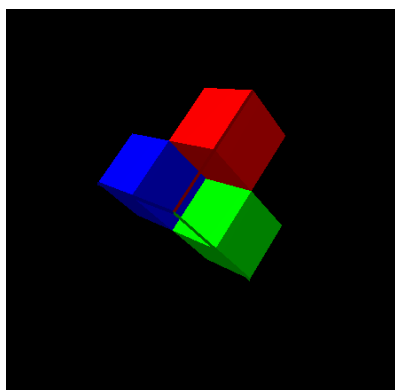
2. 绕 x 轴旋转：



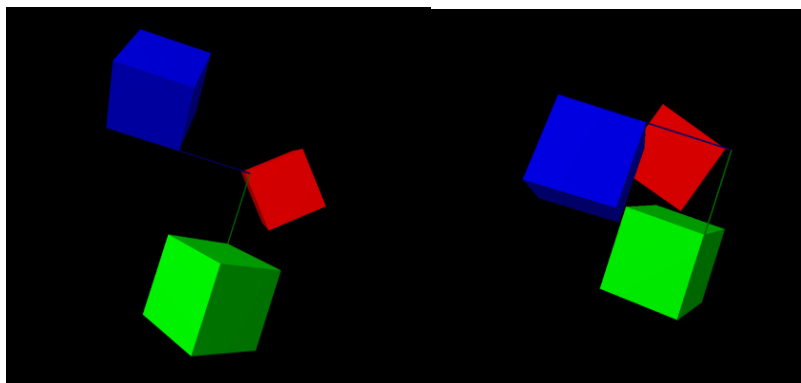
3. 绕 y 轴旋转

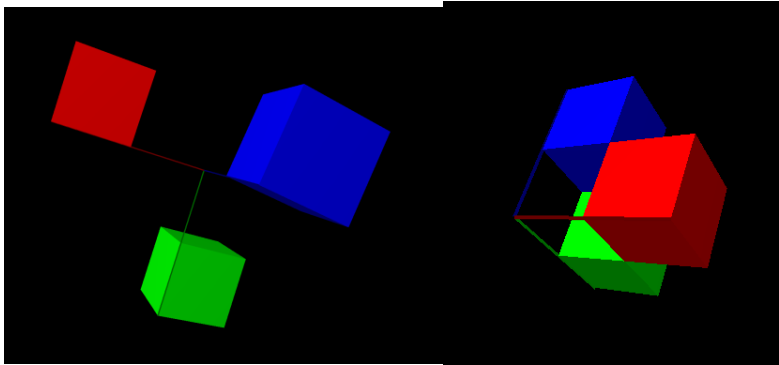


4. 绕 z 轴旋转



5. 绕 xyz 三轴旋转

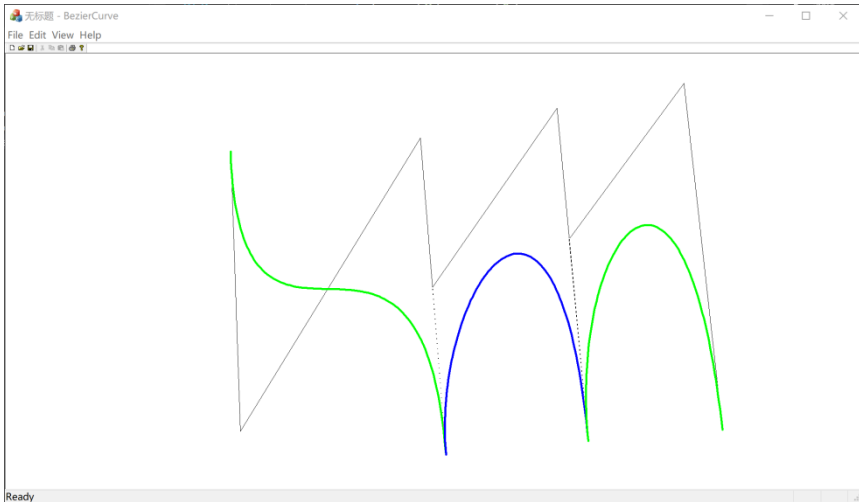
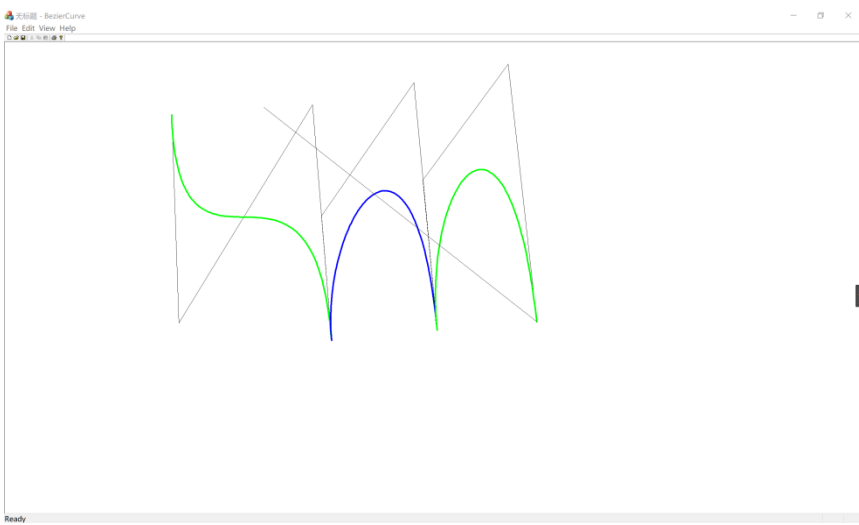


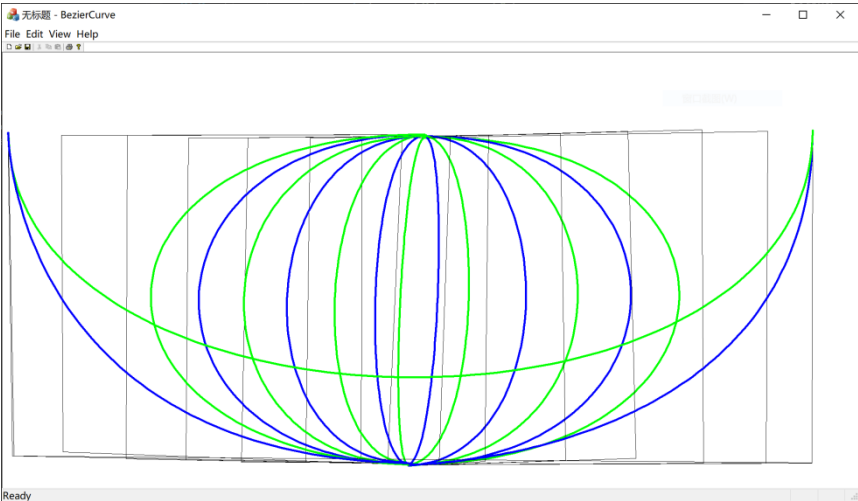
	6. 停止： 			
实验当中问题及解决方法	<p>1. 问题： 起初没有仔细阅读材料，在原有的基础上再画两个立方体即可，但是期初没有想到直接调用画立方体的函数，而是通过寻找立方体每个点的坐标一个面一个面地画。</p> <p>解决：直接调用 <code>draw_cube()</code> 函数。</p> <p>2. 问题： 没有很好的理解题目要求 3。</p> <p>解决： 每个方块都要旋转但不是一起朝一个方向旋转，而是沿各自的轴转。通过判断按下的事件来控制旋转。</p>			
成绩（教师打分）	优秀	良好	及格	不及格

计算机图形学实验报告（五）

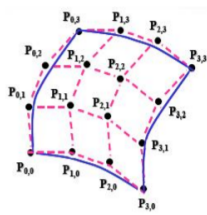
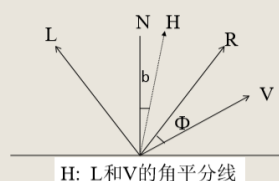
学 号	姓 名	班 级	报告日期
180400715	杨卓宸	1804102	2020/11/18
实验内容	建立一个 MFC 单文档程序，完成以下平面曲线的设计功能： (1) 实现 Bezier 曲线的交互绘制，即用鼠标点击给出控制点的位置，定义 3 次 Bezier 曲线，并绘制出控制点、控制多边形和 Bezier 曲线。 (2) 能够绘制多段 Bezier 曲线并进行拼接，使得拼接达到 1 阶几何连续 (G1 连续)。 (3) 利用实现的上述功能，设计出一个有点复杂度的平面图形，例如手的轮廓、一个动物的轮廓等。		
实验目的	要求： (1) Bezier 曲线上点的计算要求使用 de Casteljau 算法。 (2) 不同段的 Bezier 曲线用不同的颜色显示。 (3) 为了达到两段曲线的 G1 拼接，要求后一段曲线的第一个控制点和前一段曲线的最后一个控制点重合，并且后一段曲线的第 2 个控制点、前一段曲线的倒数第 2 个控制点、前一段曲线的最后一个控制点（即后一段曲线的第一个控制点）在同一条直线上。 可以在前一段曲线设计完成后，画出曲线上最后一个点处的切线 L，用户绘制后一段曲线的第 2 个控制点时需绘制在直线 L 附近，程序自动将用户绘制的控制点投影到切线 L 上。		
实验预备知识	<p>De Casteljau 算法的基本观点是选择在 AB 中的一个点 C，C 将 AB 分为 $u:1-u$ (A 到 C 的距离与 AB 之间的距离之比是 u)，让我们找到决定 C 在哪里的方法。</p> <p>从 A 到 B 的向量是 $B-A$。因为 u 是在 0 和 1 之间的比率，点 C 位于 $u(B-A)$。将 A 的位置加以考虑，点 C 为 $A+u(B-A)=(1-u)A+uB$。因此，对于给定的 u，$(1-u)A+uB$ 是在 A 和 B 之间的点 C，将 AB 分为 $u:1-u$ 的两段</p> <p>De Casteljau 算法的想法如下。假设我们想要找到 $C(u)$，u 在 $[0, 1]$ 中。由第一个多段线 $00-01-02-03 \dots -0n$ 开始，利用上面的法则找到在线段上的点 $1i$，$1i$ 在 $0i$ 到 $0(i+1)$ 的连线上并且将这段线分为 $u:1-u$ 的两部分。依次地，我们可以得到 n 个点 $10, 11, 12, \dots, 1(n-1)$，他们定义了一个新的多段线 (polyline)，一共有 $n-1$ 段。</p> <p>新点由 $1i$ 进行标记，再次利用上面的规则我们可以得到第二个多段线，具有 $n-1$ 个点 $(20, 21, \dots, 2(n-2))$ 和 $n-2$ 条边。从这个多段线开始，进行第三次，得到新的多段线，由 $n-2$ 个点 $30, 31, \dots, 3(n-3)$ 和 $n-3$ 条边组成。重复这个过程 n 次得到一个点 $n0$，Casteljau 已经证明在曲线上的点 $C(u)$ 对应 u。</p> <p>这是 de Casteljau 算法的几何解释，是在曲线设计中最漂亮结果之一。</p> <p>de Casteljau 算法伪代码： Input: array $P[0:n]$ of $n+1$ points and real number u in $[0, 1]$ Output: point on curve, $C(u)$</p>		

	<pre> Working: point array Q[0:n] for i := 0 to n do Q[i] := P[i]; // save input for k := 1 to n do for i := 0 to n - k do Q[i] := (1 - u)Q[i] + u Q[i + 1]; return Q[0]; </pre>
实验过程描述	<pre> // 重绘所有直线和贝塞尔曲线 int n = 0; for (auto thisPoint = startPoint.begin(); thisPoint < startPoint.end(); ++thisPoint) { n++; // 重绘直线 if (thisPoint == startPoint.begin()) { pDC->MoveTo(*thisPoint); continue; } pDC->LineTo(*thisPoint); pDC->MoveTo(*thisPoint); // 重绘曲线 if (n % 3 == 1 && n != 1) { std::vector<CPoint> p(thisPoint - 3, thisPoint + 1); CPen penCil; // 第奇数条曲线，画红线 if (n / 3 % 2) { penCil.CreatePen(PS_SOLID, 5, RGB(0, 255, 0)); } // 第偶数条曲线，画蓝线 else { penCil.CreatePen(PS_SOLID, 5, RGB(0, 0, 255)); } DrawBezierCurve(p, penCil); } } // 绘制贝塞尔曲线 void CBezierCurveView::DrawBezierCurve(std::vector<CPoint>& p, CPen& thisPen) { CClientDC dc(this); dc.SelectObject(&thisPen); </pre>

	<pre>dc.MoveTo(p.front()); for (double i = 0.0; i < 1.0; i += 0.01) { double points[4] = { (1 - i) * (1 - i) * (1 - i), 3 * i * (1 - i) * (1 - i), 3 * i * i * (1 - i), i * i * i }; double p_x = points[0] * p[0].x + points[1] * p[1].x + points[2] * p[2].x + points[3] * p[3].x; double p_y = points[0] * p[0].y + points[1] * p[1].y + points[2] * p[2].y + points[3] * p[3].y; dc.LineTo(p_x, p_y); dc.MoveTo(p_x, p_y); } dc.LineTo(p.back()); }</pre>
实验结果	<p>1. 可实现左键点击三次绘出曲线，并且奇数次曲线为绿色，偶数次曲线为蓝色。</p>  <p>2. 窗口变动后，曲线仍可以重绘。</p> 

	<p>3. 使用 Bezier 曲线绘制出的灯笼形对称图案。</p> 			
实验当中问题及解决方法	<p>1、Point 点的存储对我造成了一定的困难。一开始有些无从下手。最终选用了 vector 的 CPoint 类数组存储，方便很多，并且可以随时调用。</p> <p>2、鼠标左键绘图和右键暂停，我采用了实验一的解决办法，添加相应函数进行处理，也算是对之前学过的知识的总结。为了应用程序更完美的实现，我亦添加了窗口重绘功能，也采用实验一相同的处理办法。</p> <p>3、Bezier 曲线的一阶连续，我使用了点集进行存储关键点，并通过 De Casteljau 算法计算得到。</p>			
成绩（教师打分）	优秀	良好	及格	不及格

计算机图形学实验报告（六）

学 号	姓 名	班 级	报告日期
180400715	杨卓宸	1804102	2020/11/24
实验内容	光照模型实验		
实验目的	绘制一个 Bezier 曲面，利用 Phong 光照模型生成曲面的明暗效果。		
实验预备知识	<p>1. Glut 库： 本程序使用 glut 库，简化程序界面和交互操作的编写，glut 的文件夹 glutdlls37beta。 将 glut 的 dll 文件拷贝到系统目录（system32）或者程序的 debug/release 目录下程序才能正常运行。</p> <p>2. Bezier 曲面： 设 $P_{ij} (0 \leq i \leq n, 0 \leq j \leq m)$ 为 $(n+1) \times (m+1)$ 个空间点列，则 $m \times n$ 次张量积形式的 Bezier 曲面定义为：</p> $P(u, v) = \sum_{i=0}^m \sum_{j=0}^n P_{ij} B_{i,m}(u) B_{j,n}(v) \quad u, v \in [0, 1]$  <p>其中 $B_{i,m}(u) = C_m^i u^i (1-u)^{m-i}$, $B_{j,n}(v) = C_n^j v^j (1-v)^{n-j}$ 是 Bernstein 基函数。依次用线段连接点列中相邻两点所形成的空间网格，称之为特征网格。</p> <p>Bezier 曲面的矩阵表示式是：</p> $P(u, v) = \begin{bmatrix} B_{0,n}(u) & B_{1,n}(u) & \cdots & B_{m,n}(u) \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0m} \\ P_{10} & P_{11} & \cdots & P_{1m} \\ \cdots & \cdots & \cdots & \cdots \\ P_{n0} & P_{n1} & \cdots & P_{nm} \end{bmatrix} \begin{bmatrix} B_{0,m}(v) \\ B_{1,m}(v) \\ \cdots \\ B_{n,m}(v) \end{bmatrix}$ <p>在一般实际应用中，n、m 不大于 4。</p> <p>3. Phong 光照模型的实现：</p> <p>对物体表面上的每个点 P，均需计算光线的反射方向。为了减少计算量，假设：</p> <ul style="list-style-type: none"> 光源在无穷远处，L 为常向量 视点在无穷远处，V 为常向量 $(H \cdot N)$ 近似 $(R \cdot V)$，H 为 L 与 V 的平分向量  <ul style="list-style-type: none"> 对所有的点总共只需计算一次 H 的值，节省了计算时间 		

实验过程描述	<pre>1. surfacepoint() 计算 Bezier 曲线上的点 计算 Bezier 曲面上一些采样点，把这些点连成一个三角形网格 M. void surfacepoint(double u, double v, double& x, double& y, double& z) { double ax0 = -0.5; double ay0 = 0.1; double az0 = -0.2; double bx0 = 0.1; double by0 = 0.4; double bz0 = 0.1; double cx0 = 0.5; double cy0 = 0.1; double cz0 = 0.3; double ax1 = -0.5; double ay1 = -0.2; double az1 = 0.2; double bx1 = 0.2; double by1 = 0.2; double bz1 = 0.3; double cx1 = 0.5; double cy1 = -0.2; double cz1 = 0.2; double ax2 = -0.5; double ay2 = -0.6; double az2 = 0.3; double bx2 = 0.1; double by2 = -0.4; double bz2 = 0.5; double cx2 = 0.5; double cy2 = -0.3; double cz2 = 0.1; double u1 = (1 - u) * (1 - u); double u2 = 2 * (1 - u) * u; double u3 = u * u; double ax = u1 * ax0 + u2 * ax1 + u3 * ax2; double ay = u1 * ay0 + u2 * ay1 + u3 * ay2; double az = u1 * az0 + u2 * az1 + u3 * az2; double bx = u1 * bx0 + u2 * bx1 + u3 * bx2; double by = u1 * by0 + u2 * by1 + u3 * by2; double bz = u1 * bz0 + u2 * bz1 + u3 * bz2; double cx = u1 * cx0 + u2 * cx1 + u3 * cx2; double cy = u1 * cy0 + u2 * cy1 + u3 * cy2;</pre>
--------	--

```

double cz = u1 * cz0 + u2 * cz1 + u3 * cz2;

double v1 = (1 - v) * (1 - v);
double v2 = 2 * (1 - v) * v;
double v3 = v * v;
x = v1 * ax + v2 * bx + v3 * cx;
y = v1 * ay + v2 * by + v3 * cy;
z = v1 * az + v2 * bz + v3 * cz;
}

```

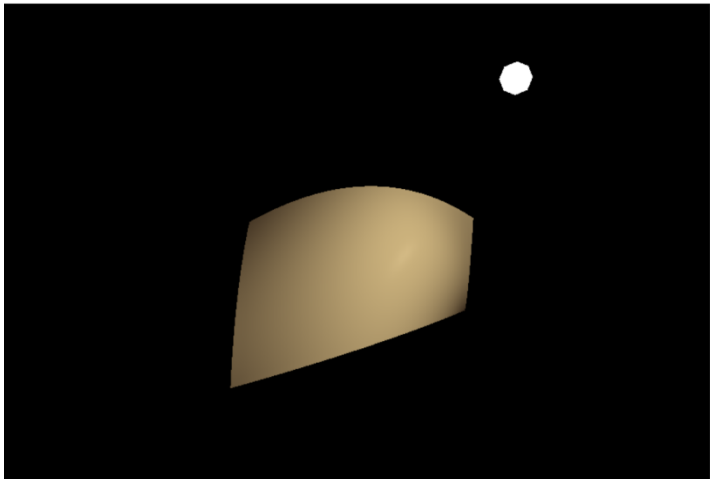
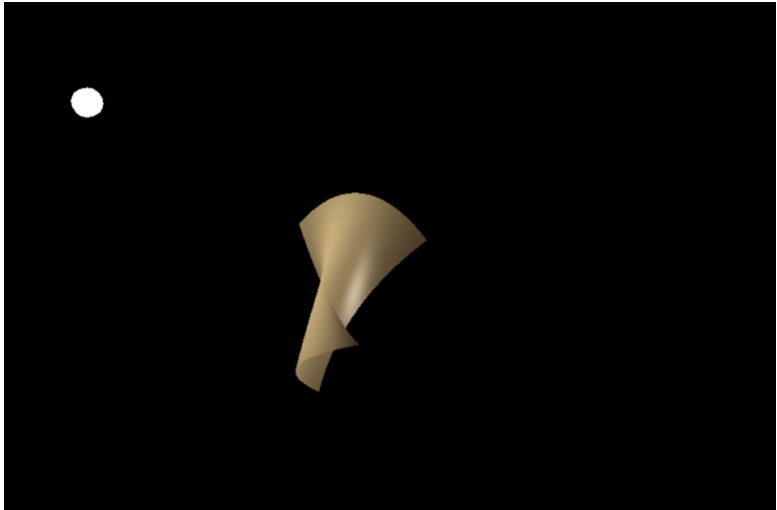
2. phong() 计算顶点的颜色

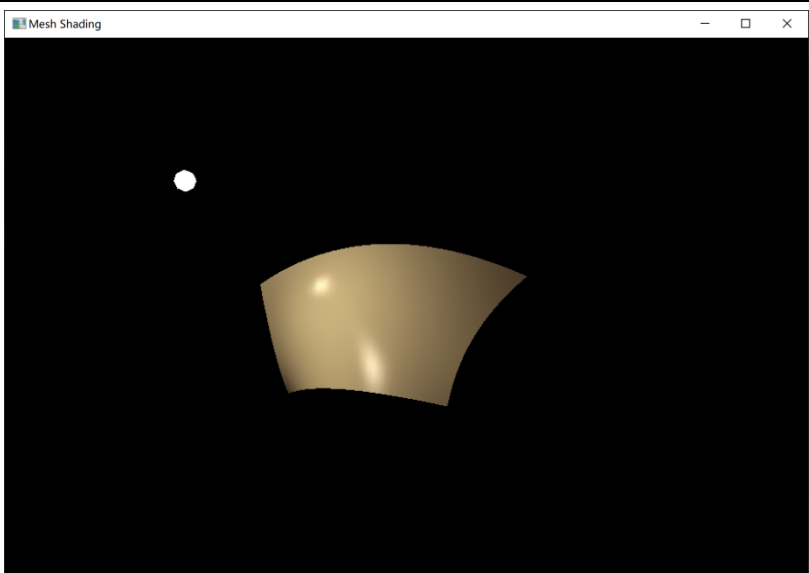
利用 Phong 光照模型计算网格 M 上所有顶点的颜色，绘制该网格曲面。网格曲面的绘制过程为依次绘制网格曲面的每一个三角形。

```

void phong()
{
    //每一个顶点计算一个颜色
    GLfloat  SZ[3]; //入射光向量
    GLfloat  RZ[3]; //镜面反射光向量
    GLfloat  VZ[3]; //人眼的视角向量
    //每一个顶点计算一个颜色
    for (int i = 0; i < NUMV; i++) {
        //计算每个网格的视角向量，-----GLfloat
vertices[MAXNUMV][3]; //存放网格的顶点
        VZ[0] = vertices[i][0] - xcam; //xcam、ycam、zcam是人眼的坐标
        VZ[1] = vertices[i][1] - ycam;
        VZ[2] = vertices[i][2] - zcam;
        Normalize(VZ); //单位化
        //计算光源的入向量，-----GLfloat
light_position0[] = { 0.5, 0.5, 1.0, 1.0 }; //光源的位置
        SZ[0] = vertices[i][0] - light_position0[0];
        SZ[1] = vertices[i][1] - light_position0[1];
        SZ[2] = vertices[i][2] - light_position0[2];
        Normalize(SZ); //单位化
        //计算  $n \cdot s$ 
        GLfloat cos1 = SZ[0] * normals[i][0] + SZ[1] * normals[i][1] + SZ[2] * normals[i][2];
        //计算反射光的向量  $r = 2 (n \cdot s)n - s$ ; ppt 上的公式
        RZ[0] = 2 * (cos1) * normals[i][0] - SZ[0];
        RZ[1] = 2 * (cos1) * normals[i][1] - SZ[1];
        RZ[2] = 2 * (cos1) * normals[i][2] - SZ[2];
        Normalize(RZ); //单位化
        //计算反射光与人眼之间的夹角的cos值:  $r \cdot v$ 
        GLfloat cos2 = RZ[0] * VZ[0] + RZ[1] * VZ[1] + RZ[2] * VZ[2];
    }
}

```

	<pre> colors[i][0] = ambient[0] * light_ambient[0] + diffuse[0] * light_diffuse0[0] * cos1 + specular[0] * light_specular0[0] * pow(cos2, shininess); colors[i][1] = ambient[1] * light_ambient[1] + diffuse[1] * light_diffuse0[1] * cos1 + specular[1] * light_specular0[1] * pow(cos2, shininess); colors[i][2] = ambient[2] * light_ambient[2] + diffuse[2] * light_diffuse0[2] * cos1 + specular[2] * light_specular0[2] * pow(cos2, shininess); } }</pre> <p>3. DrawFace() 绘制曲面 绘制一个三角形时，要指定每一个顶点的颜色（顶点颜色即采用 Phong 模型计算出的颜色）。 指定一个顶点的颜色 glColor3f(r, g, b)。</p>
实验结果	<p>直接运行结果：</p>  <p>利用鼠标旋转结果：</p> 

				
实验当中问题及解决方法	<p>问题：起初直接运行已有代码，程序报错显示各种头文件无法找到。</p> <p>解决方法：没有仔细查看题目详解后面的提示，只是单纯的上网查找原因，经过一些简单的修改仍未成功。之后发现要配置库，配置好之后还是不能正常运行，之后通过同学的建议尝试更改了 Windows SDK 版本问题得到解决。</p>			
成绩（教师打分）	优秀	良好	及格	不及格