计算机图形学实验报告

姓　名：　白云鹏

学　号：1004161221

日　期：2018/12/28

实验环境

操作系统	Windows 10
CPU	Intel i5-7360U (4) @ 2.30GHz
内存	8G
GPU	Intel Iris Plus Graphics 640

贝塞尔曲线

实验原理

三次贝塞尔曲线的曲线方程为：

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3$$

直接进行计算即可

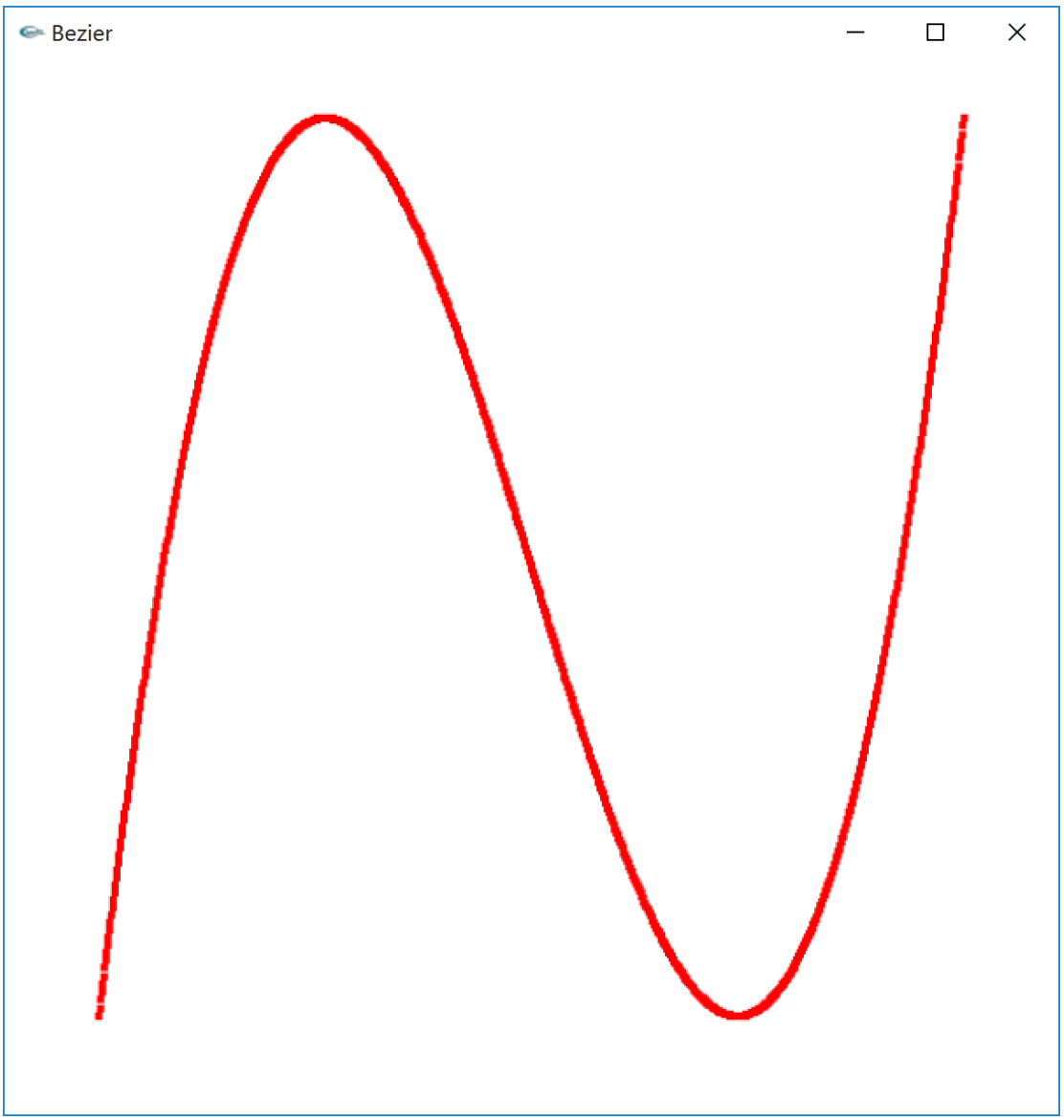
关键代码

贝塞尔曲线的计算：

```
void computeBzePt(GLfloat u, wcPt3D *bezPt, GLint
nCtrlPts, wcPt3D *ctrlPts, GLint *c)
{
    GLint k, n = nCtrlPts - 1;
    GLfloat bezBlendFcn;
    bezPt->z = bezPt->y = bezPt->x = 0.0;
    for (k = 0; k < nCtrlPts; ++k)
    {
        bezBlendFcn = c[k] * pow(u, k) * pow(1 - u, n -
k);
```

```
        bezPt->x += ctrlPts[k].x * bezBlendFcn;
        bezPt->y += ctrlPts[k].y * bezBlendFcn;
        bezPt->z += ctrlPts[k].z * bezBlendFcn;
    }
}
```

实验结果



B 样条曲线

实验原理

三次 B 样条曲线的矩阵形式为：

$$P(t) = [t^3 \quad t^2 \quad t \quad 1] = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

其中P_0, P_1, P_2, P_3为平面上的四个离散点。

用分量形式写出，为：

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3$$

其中：

$$a_0 = \frac{1}{6}(x_0 + 4x_1 + x_2)$$

$$a_1 = \left(-\frac{1}{2}\right)(x_0 - x_2)$$

$$a_2 = \frac{1}{2}(x_0 - 2x_1 + x_2)$$

$$a_3 = \left(-\frac{1}{6}\right)(x_0 - 3x_1 + 3x_2 - x_3)$$

$$b_0 = \frac{1}{6}(y_0 + 4y_1 + y_2)$$

$$b_1 = \left(-\frac{1}{2}\right)(y_0 - y_2)$$

$$b_2 = \frac{1}{2}(y_0 - 2y_1 + y_2)$$

$$b_3 = \left(-\frac{1}{6}\right)(y_0 - 3y_1 + 3y_2 - y_3)$$

$(x_0, y_0), (x_1, y_1)(x_2, y_2)(x_3, y_3)$分别为平面四点的坐标。

关键代码

B 样条曲线坐标的计算：

```
void bline(int n)
{
    float f1, f2, f3, f4;
    float dt = 1.0 / n;
    float t;
    glBegin(GL_LINE_STRIP);
    for (int num = 0; num < segmentnum; ++num)
    {
        for (int i = 0; i <= n; ++i)
        {
           //对应着 B 样条曲线函数的表达式
            t = i * dt;
            f1 = (3 * t * t - t * t * t - 3 * t + 1) / 6.0;
            f2 = (3 * t * t * t - 6 * t * t + 4) / 6.0;
            f3 = (-3 * t * t * t + 3 * t * t + 3 * t + 1) / 6.0;
            f4 = t * t * t / 6.0;
            glVertex2f(f1 * vec[num].x + f2 * vec[num + 1].x + f3 *
vec[num + 2].x + f4 * vec[num + 3].x, f1 * vec[num].y + f2 * vec[num
+ 1].y + f3 * vec[num + 2].y + f4 * vec[num + 3].y);
        }
    }
    glEnd();
}
```

实验结果

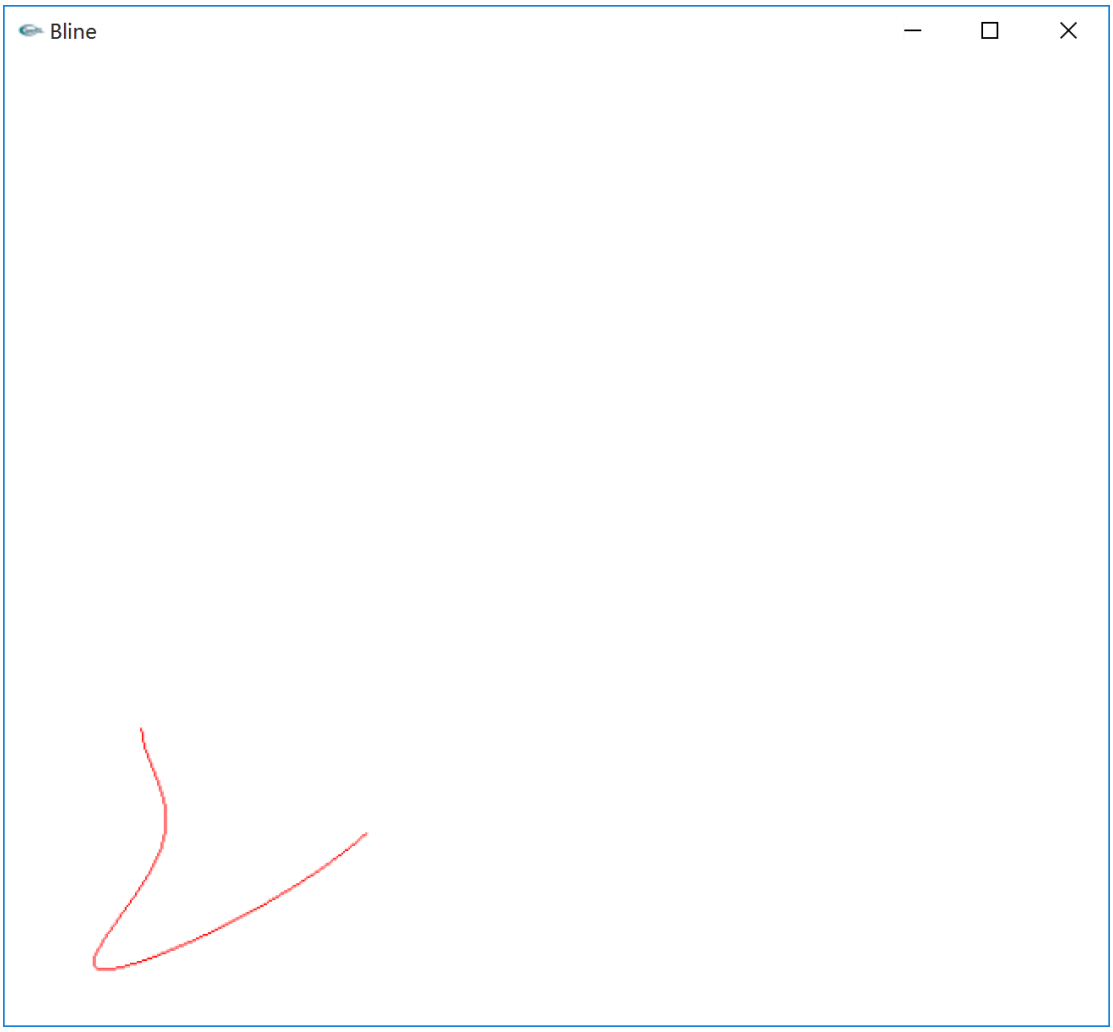


图 1 B 样条曲线

可动时钟

实验原理

时钟的绘制比较容易，主要是各个点坐标的计算，需要注意的是在计算的时候，中心点为(0,0)，而在计算坐标的时候需要对其角度进行换算，即$\frac{\pi}{2}-$

$angle$，才能得到真正的角度。原理上比较容易，首先是对表盘进行绘制，然后绘制其刻度，在根据当前系统时间绘制指针，最后每隔一定时间间隔就对其进行重新绘制即可。

关键代码

表盘的绘制：

```
void getBase()
{
    double midx = len / 500;
    glLineWidth(3);
    glColor3f(1, 0, 0);
    //绘制外面的大圆
    glBegin(GL_LINE_LOOP);
    double gap = 2 * pi / 1000;
    for (double start = 0; start < 2 * pi; start += gap)
    {
        double nowx = r * cos(start);
        double nowy = r * sin(start);
        glVertex3d(nowx, nowy, 5);
    }
    glEnd();

   //绘制内层小圆
    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    for (double start = 0; start < 2 * pi; start += gap)
    {
        double nowx = lowr * cos(start);
        double nowy = lowr * sin(start);
        glVertex3d(nowx, nowy, 5);
```

```
    }
    glEnd();

    glPointSize(5);
    glColor3f(0, 0, 0);
   //绘制中心点
    glBegin(GL_POINTS);
    glVertex3f(0, 0, 5);
    glEnd();
}
```

刻度的绘制：

```
void getLine()
{
    double gap = 2 * pi / 60;
    int i = 0;
    glColor3f(0, 1, 0);
    for (double start = 0; start < 2 * pi; start += gap,
++i)
    {
        double startx = r * cos(start);
        double starty = r * sin(start);
        double endx = (r - shortlinesz) * cos(start);
        double endy = (r - shortlinesz) * sin(start);
         //如果是5的倍数则需要绘制长线条
        if (i % 5 == 0)
        {
            endx = (r - longlinesz) * cos(start);
            endy = (r - longlinesz) * sin(start);
        }
        glBegin(GL_LINES);
        glVertex3d(startx, starty, 5);
```

```
        glVertex3d(endx, endy, 5);
        glEnd();
    }
}
```

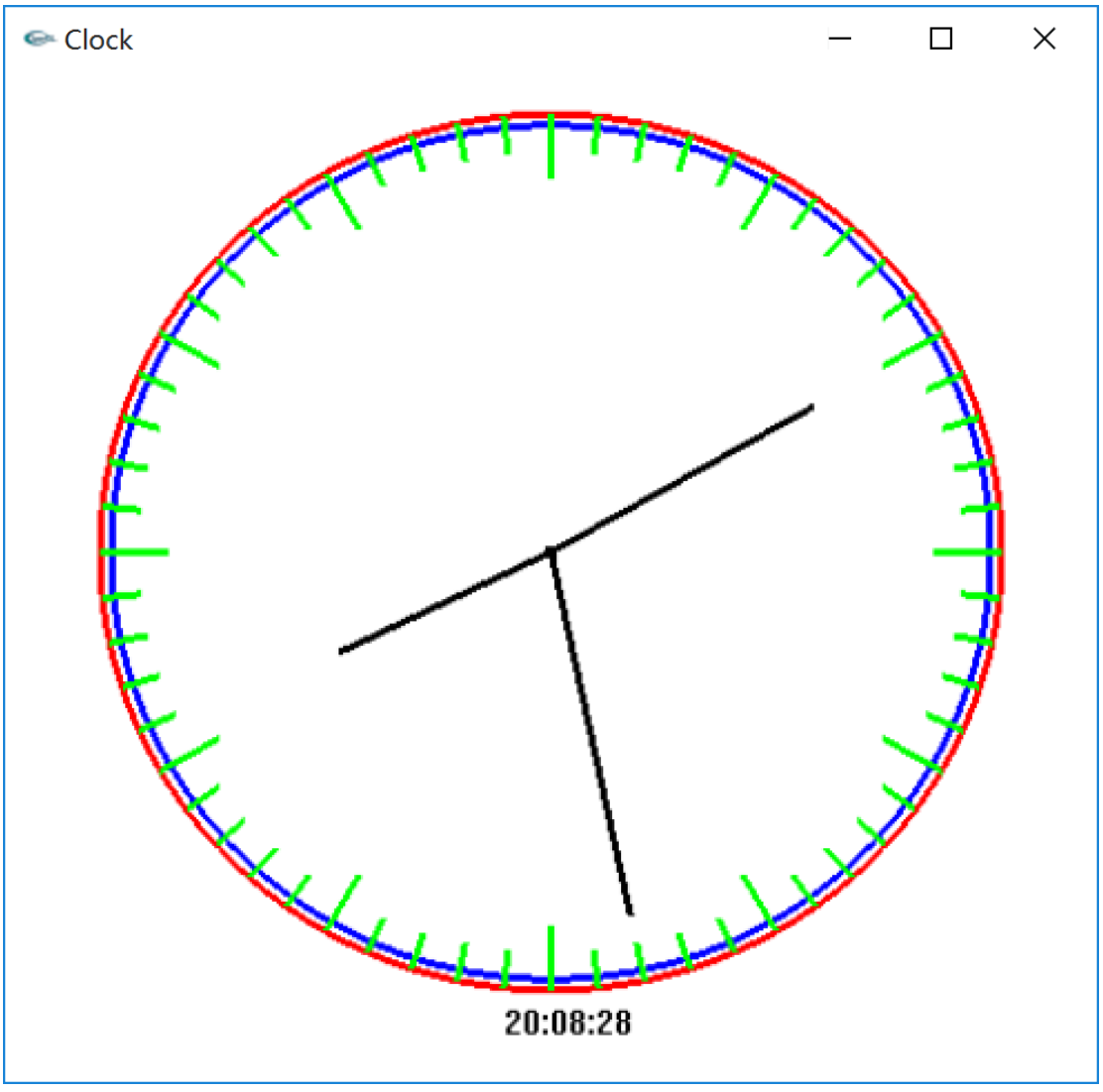
指针的绘制：

```
void getPointer()
{
   //获取当前系统时间
    time_t tt;
    tt = time(NULL);
    tm *t = localtime(&tt);
    int hr = t->tm_hour;
    double second = t->tm_sec;
    double minute = t->tm_min;
    hr = hr % 12;
    double hour = hr;
    //计算三个指针的角度
    double hourangle = pi / 2 - (2 * pi * hour / 12 + pi
/ 6 * (minute / 60));
    double minuteangle = pi / 2 - (minute * 2 * pi / 60 +
2 * pi / 60 * (second / 60));
    double secondangle = pi / 2 - (second * 2 * pi / 60);
    glLineWidth(3.0);
    glColor3f(0, 0, 0);
    //绘制三个指针
    glBegin(GL_LINES);
    glVertex3f(0, 0, 5);
    glVertex3f(hoursz * cos(hourangle), hoursz *
sin(hourangle), 5);
    glEnd();

    glBegin(GL_LINES);
    glVertex3f(0, 0, 5);
```

```
    glVertex3f(minutesz * cos(minuteangle), hoursz *
sin(minuteangle), 5);
    glEnd();
    glBegin(GL_LINES);
    glVertex3f(0, 0, 5);
    glVertex3f(secondsz * cos(secondangle), secondsz *
sin(secondangle), 5);
    glEnd();
}
```

实验结果



椭圆

实验原理

取$F(x,y)= b^2x^2+a^2y^2-a^2b^2$。假设当前点$p(x_i,y_i)$，下一个可能的点为$\mathrm{p}_1(x_i+1,y_i),p_2(x_i+1,y_i-1)$。那么若$d_i=F(x_i+1,y_i-0.5)<0$，则表示像素点$p_1,p_2$的中点在椭圆内，可取$p_1$作为下一个点，否则就取$p_2$作为下一个点。

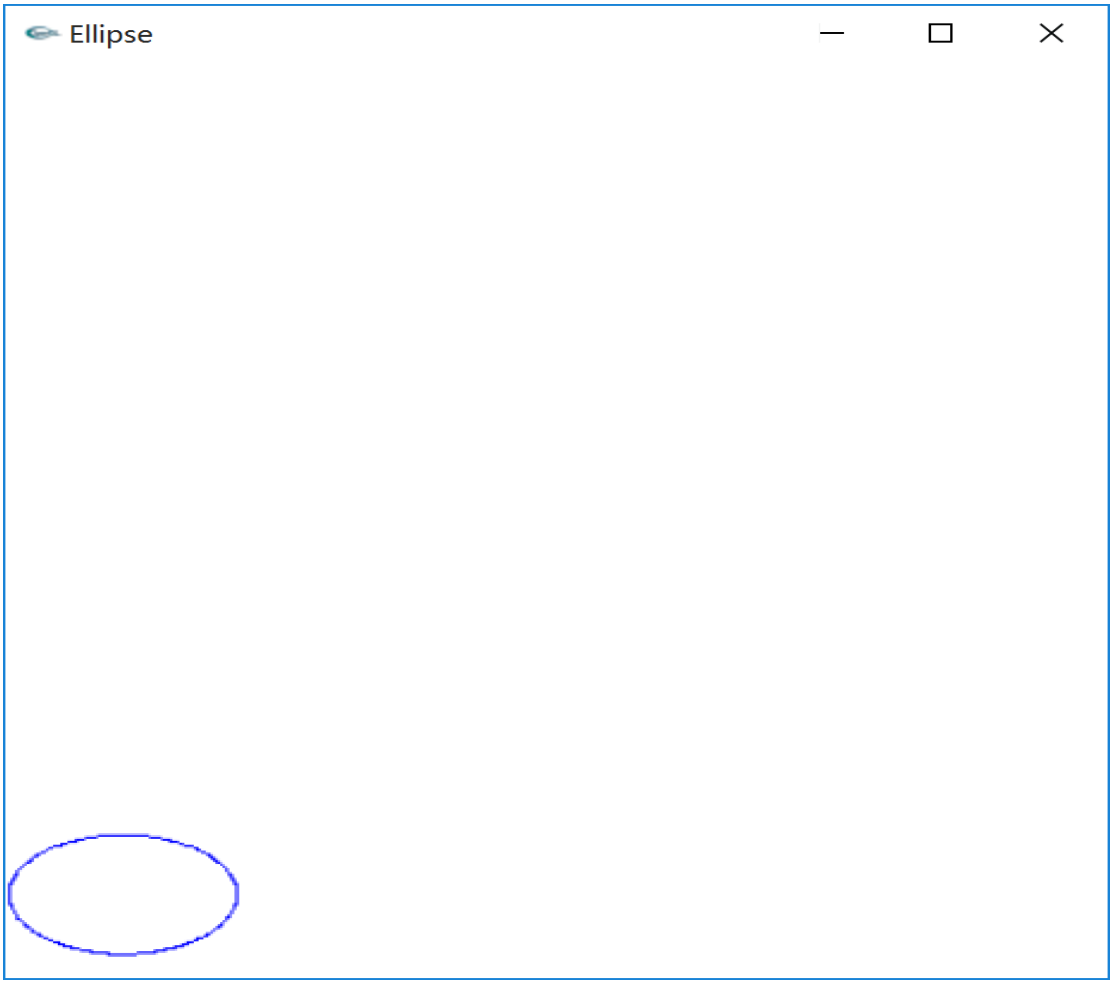
关键代码

椭圆的绘制，这里只用画$\frac{1}{4}$的椭圆，剩下的可用四路对称就可以得到整个圆。

```
void drawEllipse(int x0, int y0, int a, int b)
{
    double aa = a * a;
    double bb = b * b;
    double d = bb + aa * (0.25 - b);
    int x = 0, y = b;
    duichen(x0, y0, x, y);
    //椭圆的上半部分
    while (bb * (x + 1) < aa * (y - 0.5))
    {
        if (d < 0)
        {
            d += bb * (2 * x + 3);
        }
        else
        {
            d += (bb * (2 * x + 3) + aa * ((-2) * y +
2));
            --y;
        }
```

```
        ++x;
        duichen(x0, y0, x, y);
    }
    d = (b * (x + 0.5)) * 2 + (a * (y - 1)) * 2 - (a * b)
* 2;
    //椭圆的下半部分
    while (y > 0)
    {
        if (d < 0)
        {
            d += bb * (2 * x + 2) + aa * ((-2) * y + 3);
            ++x;
        }
        else
        {
            d += aa * ((-2) * y + 3);
        }
        --y;
        duichen(x0,y0,x,y);
    }
}
```

实验结果



实验心得

在本次实验中，补充了前几次实验中遗漏的椭圆绘制，然后完成了 B 样条曲线以及贝塞尔曲线的绘制，这两个算法的原理比较简单但是在实现上仍然有一些困难，主要是在计算的时候可能会出现计算的失误导致最后曲线绘制出现错误。在时钟的绘制中，由于开始的时候对于坐标系仍然有一些混乱导致在绘图的时候钟表呈现中心对称，在将时间固定为一个值后不断对其进行调整，最后才得到了其真正角度的换算方法。在绘制时钟的时候思路比较简单，在绘制表盘的圆形的时候是将其圆划分为多个点然后使用 LINE_LOOP 进行绘制的，在调试的过程中由于对系统时间结构体的不熟悉导致强转类型出错程序崩溃。