智能系统的设计与仿真

作业一

姓名：石若川　学号：2111381　专业：智能科学与技术

1 实验内容

- 绘制坐标轴和正弦曲线，绘图大小随窗口大小而变化
- 鼠标左键点击窗口，在鼠标点击位置显示十字光标，并显示当前位置在用户自定义坐标系下的坐标
- 鼠标右键点击窗口，清除十字光标与坐标

2 代码分析

2.1 设定绘图范围

首先在 OnDraw 函数中，利用 rect 获取矩形窗口的大小。利用 SelectObject 限定左上角四分之一矩形为绘图区域。定义黑色画刷 NewBrush 绘制黑色背景矩形。利用 SetWindowsOrg 设置逻辑坐标系的坐标原点。

```
CRect rect; //获取窗口大小
GetClientRect(&rect);

// 限制绘图区域为左上四分之一矩形
CRgn rgn;
rgn.CreateRectRgn(0, 0, rect.Width() / 2, rect.Height() / 2);
pDC->SelectObject(rgn);

CBrush NewBrush;
CBrush* pOldBrush;
NewBrush.CreateSolidBrush(RGB(0, 0, 0));
pOldBrush = pDC->SelectObject(&NewBrush);
pDC->Rectangle(rect);
pDC->SelectObject(pOldBrush);

//设置逻辑坐标系原点
CPoint ptLeft(rect.left, rect.top);
CPoint ptRight(rect.right, rect.bottom);
pDC->SetWindowOrg(0, -rect.Height() / 4);
```

2.2 绘制坐标轴与刻度

首先定义白色画笔 NewPen，将绘图区域的水平方向中轴线作为 x 轴，y 轴的水平位置距离原点 10 个像素。在绘制坐标轴的过程中，每隔一个单位长度绘制刻度并显示刻度值，并在坐标轴末端位置绘制箭头。

```
CPen NewPen;
CPen* pOldPen;
NewPen.CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
pOldPen = pDC->SelectObject(&NewPen);

// 绘制横纵坐标
pDC->MoveTo(0, 0);
pDC->LineTo(rect.Width() / 2, 0);
pDC->MoveTo(10, -rect.Height() / 4);
pDC->LineTo(10, rect.Height() / 4);

//设置文本颜色和背景色
pDC->SetBkColor(RGB(0, 0, 0));
pDC->SetTextColor(RGB(255, 255, 255));
pDC->SetTextAlign(TA_CENTER | TA_TOP);

//绘制x轴刻度
for (int i = 0; i < cx; i++) {
    if (i == cx - 1) {// x轴末端绘制箭头
        pDC->MoveTo(10 + (rect.Width() / 2 - 10) * (i + 1) / cx, 0);
        pDC->LineTo((rect.Width() / 2 - 10) * (i + 1) / cx, 5);
        pDC->MoveTo(10 + (rect.Width() / 2 - 10) * (i + 1) / cx, 0);
        pDC->LineTo((rect.Width() / 2 - 10) * (i + 1) / cx, -5);
    }
    else {
        pDC->MoveTo(10 + (rect.Width() / 2 - 10) * (i + 1) / cx, 0);
        CString str1;
        str1.Format(_T("%d"), i + 1);
        pDC->TextOut(10 + (rect.Width() / 2 - 10) * (i + 1) / cx, 10, str1);
        pDC->LineTo(10 + (rect.Width() / 2 - 10) * (i + 1) / cx, -10);
    }
}

//绘制y坐标刻度
for (int i = 0; i < cy; i++) {
    if (i == cy - 1) {// y轴末端绘制箭头
        pDC->MoveTo(10, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4);
```

```
        pDC->LineTo(15, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4 + 10);
        pDC->MoveTo(10, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4);
        pDC->LineTo(5, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4 + 10);
    }
    else {
        pDC->MoveTo(10, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4);
        pDC->LineTo(20, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4);
        CString str1;
        str1.Format(_T("%d"), -(cy / 2 - i - 1) * dy);
        pDC->TextOut(25, -rect.Height() / 2 * (i + 1) / cy + rect.Height() / 4 + 5,
            str1);
    }
}
```

2.3 绘制正弦曲线

为实现正弦曲线与窗口同步缩放，需要求出横纵方向数据点的间距，作为缩放比例。定义红色画笔 m_redPen，将每个数据点的横纵坐标乘上缩放比例后，绘制于图像上。

```
//计算横向数据点之间的间距
double x_step = (ptRight.x / 2 - ptLeft.x - 10) / ((double)DATA_NUM);
double arrayMax = -1, arrayMin = 1000; //获取数据最大值最小值
for (int i = 0; i < DATA_NUM; i++) {
    if (m_array[i] > arrayMax) arrayMax = m_array[i];
    if (m_array[i] < arrayMin) arrayMin = m_array[i];
}

//计算纵向数据点之间的间距
double y_step = (ptLeft.y - ptRight.y / 2) / (arrayMax - arrayMin);

CPen m_redPen;
m_redPen.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
pDC->SelectObject(m_redPen);
int x, y;

//绘制正弦图像
for (int i = 0; i < DATA_NUM - 1; i++) {
    x = (int)(i * x_step) + 10;
    y = (int)(m_array[i] * y_step);
    pDC->MoveTo(x, y);
    x = (int)((i + 1) * x_step) + 10;
    y = (int)(m_array[i + 1] * y_step);
```

```
    pDC->LineTo(x, y);
}
pDC->SelectObject(pOldPen);
```

2.4 鼠标左键点击

鼠标左键点击对应的是 OnLButtonDown 函数。在该函数中，首先定义了蓝色画笔 Pen。然后，判断点击位置 point 是否在左上绘图范围内，若在此范围内，则在点击位置绘制十字并显示坐标值。

```
void CHomeworkView::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
  CView::OnLButtonDown(nFlags, point);
  CClientDC dc(this);
  CPen Pen;
  Pen.CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
  dc.SelectObject(Pen);

  CRect rect;
  GetClientRect(&rect);
  if (point.x < rect.Width() / 2 && point.y < rect.Height() / 2) {
    // 绘制十字
    dc.MoveTo(point.x - 10, point.y);
    dc.LineTo(point.x + 10, point.y);
    dc.MoveTo(point.x, point.y - 10);
    dc.LineTo(point.x, point.y + 10);

    // 显示坐标
    CString str1;
    double x, y;
    x = double(point.x) / rect.Width() * dx * cx * 2;
    y = 10 - double(point.y) / rect.Height() * dy * cy * 2;
    str1.Format(_T("x=%f,y=%f"), x, y);
    dc.TextOut(point.x, point.y, str1);
  }
}
```

2.5 鼠标右键点击

鼠标右键点击对应的是 OnRButtonDown 函数。在该函数中，利用 Invalidate 函数使整个窗口客户区无效，从而清除绘制的十字与坐标，但不会清除程序绘制的函数图像。

```
void CHomeworkView::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
   Invalidate(); // 清除十字与坐标显示
   CView::OnRButtonDown(nFlags, point);
}
```

3 结果展示

运行程序后，显示的窗口如图3.1所示。

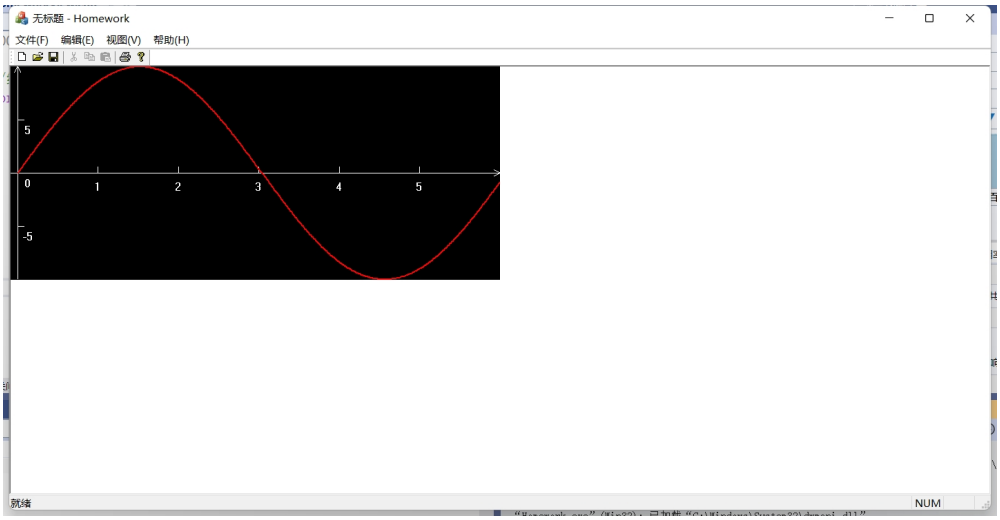

图 3.1: 运行程序

拖动窗口进行缩放，函数图像可以同步进行比例缩放，如图3.2所示。

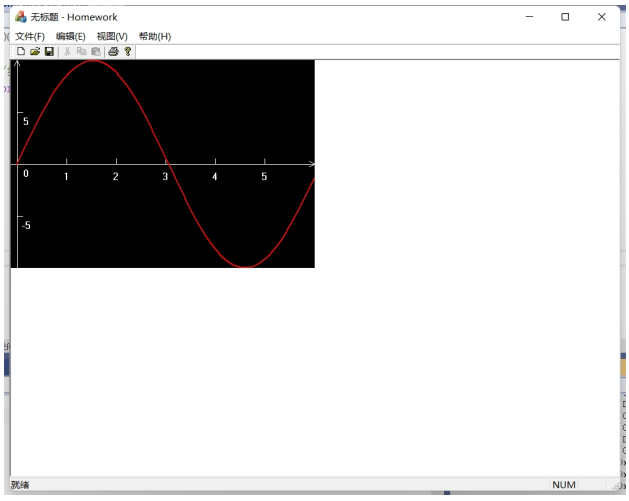

图 3.2: 拖动窗口

鼠标左键点击绘图区域显示该点在设定坐标系下的坐标，并显示蓝色十字。若左键点击的其他白色区域将不会显示十字与坐标值。点击右键后十字与坐标值将清除。

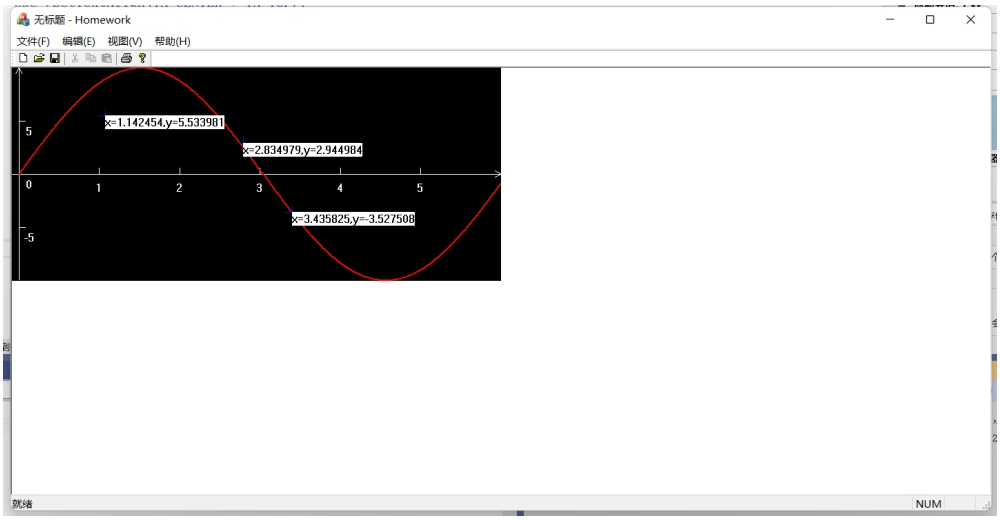

图 3.3: 鼠标左键点击，显示十字与坐标

4 实验总结

在本次作业中，我实现了坐标轴和正弦曲线在指定区域的绘制，控制绘图大小随窗口大小而变化，并实现鼠标左键点击显示十字光标与坐标，以及鼠标右键点击清除十字光标与坐标。通过本次作业，我学习并掌握了 MFC 指定区域进行函数图像绘制的方法，对各坐标系的转换有了更深刻的认识，学习到了利用绘图比例控制图像与窗口同步缩放的原理。