计算机图形学

哈尔滨工业大学（威海）
计算机科学与技术学院
伯彭波

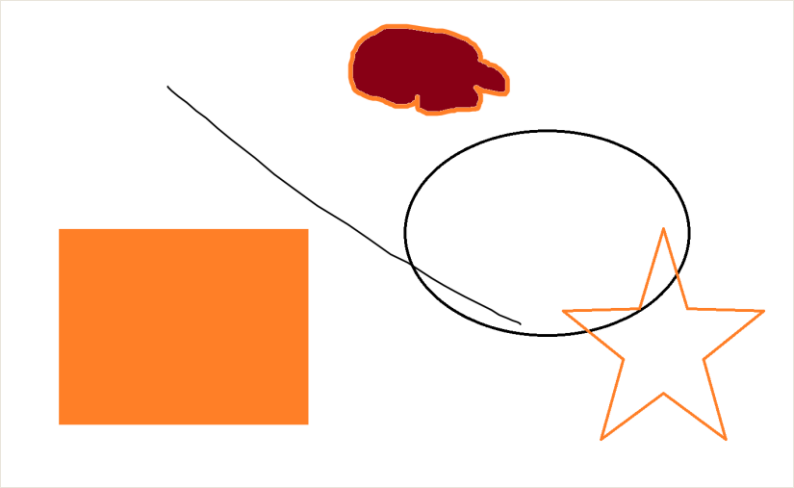
在窗口中绘图

内容：

- 介绍在窗口中绘图的方法:在哪画？怎么画？

目的：

- 能够绘制二维图形
- 为三维图形编程和图像相关处理奠定基础



主要内容

- **图形设备接口（GDI）与设备环境（DC）**
- **CDC类**
- **应用举例**
- **用GDI绘图**

图形设备接口（GDI）

- **GDI(Graphics Device Interface，GDI): Windows提供的图形设备的抽象接口。**
- **GDI 作为Windows的重要组成部分，负责管理用户绘图操作时功能的转换。**
- **用户通过调用GDI 函数与设备打交道，GDI通过设备提供的驱动程序将绘图语句转换为绘图指令，实现设备无关性（显示器或打印机）。**

GDI的图形输出

- 应用程序可以使用**GDI** 创建如下图形输出
  - 矢量图形：画线和填充图形，包括点、直线、曲线、多边形、扇形和矩形等。
  - 光栅图形：通过光栅图形函数对以位图形式存储的数据进行操作，包括各种位图和图标的输出。
  - 文本：以图形方式输出文本，可以设置文本的各种效果，如加粗、斜体、设置颜色等。

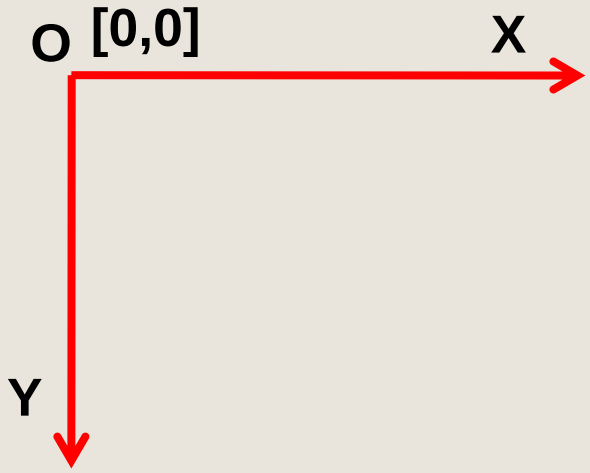
设备环境 (DC)

设备环境 (Device Context，DC)：虚拟逻辑设备，也称设备描述表或设备上下文。

- **DC 由GDI 创建，用来代表设备连接的数据结构。**
- **DC 的主要功能**
  - **允许应用程序使用一个输出设备。**
  - **提供应用程序、设备驱动和输出设备之间的连接。**
  - **保存当前信息，例如当前的画笔、画刷、字体和位图等图形对象及其属性，以及颜色和背景等影响图形输出的绘图模式。**
  - **保存窗口剪切区域(Clipping Region)，限制程序输出到输出设备中窗口覆盖的区域。**

设备坐标系统

- 设备坐标系统
  - x轴自左至右，y轴从上到下，坐标原点在屏幕左上角

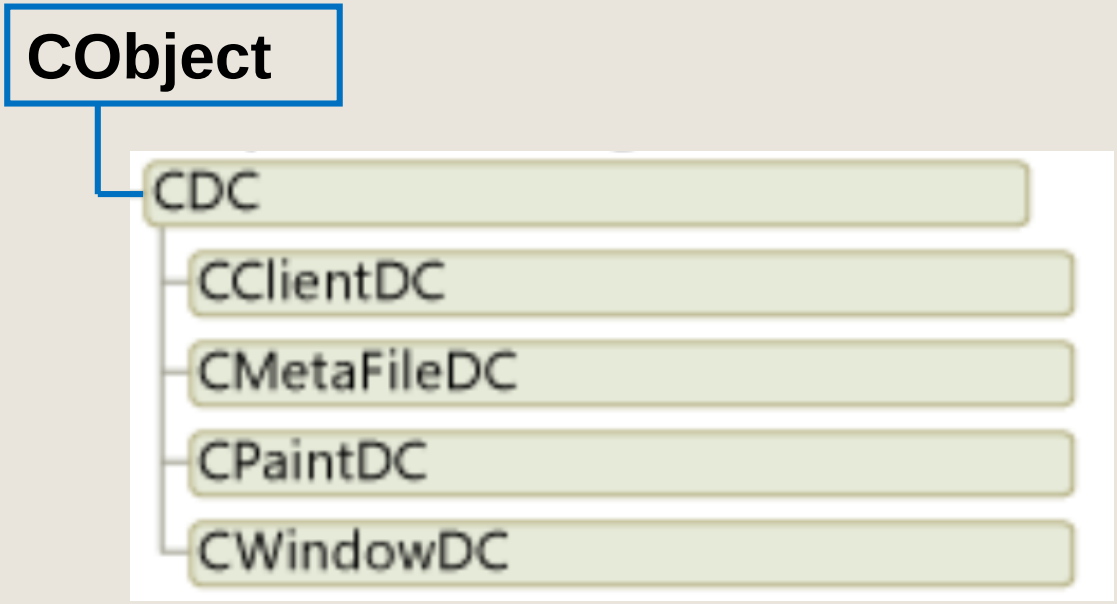


主要内容

- 图形设备接口（GDI）与设备描述表（DC）
- CDC类
- 应用举例
- 用GDI绘图

设备环境类CDC

- **MFC的CDC类将 DC 和 GDI函数进行封装。**
- **CDC类派生自CObject类，其子类包括**
  - **CClientDC**
  - **CMetaFileDC**
  - **CPaintDC**
  - **CWindowDC**



CPaintDC

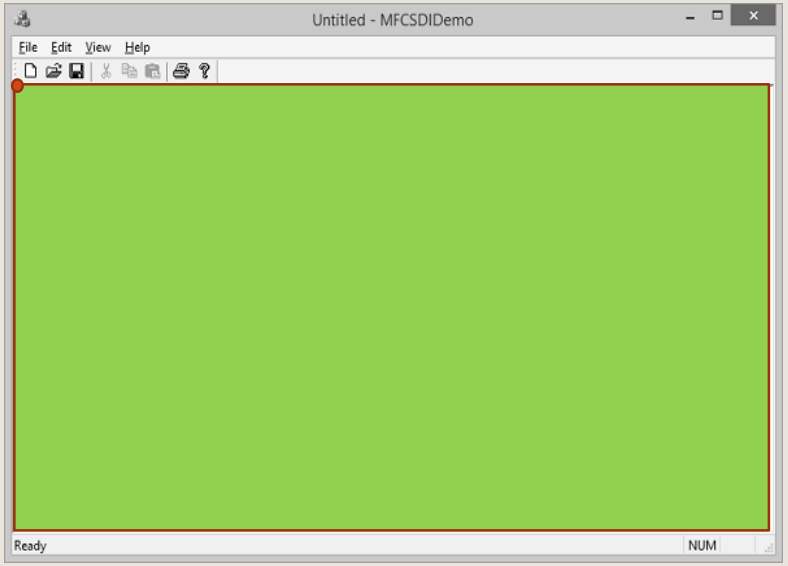
- 当窗口的某个区域需要重绘(比如窗口的大小改变)时激发窗口重绘消息**WM_PAINT**，相应消息处理函数**CWnd::OnPaint()** 将被调用。
- **CPaintDC**一般只用于**OnPaint**函数中，在处理完窗口重绘后， **CPaintDC**对象的析构函数把**WM_PAINT**消息从消息队列中清除，避免不断地重绘操作。
- 坐标原点**(0,0)**是客户区的左上角。

CPaintDC

- **Windows系统发送WM_PAINT消息的时机**
  - 第一次创建一个窗口时
  - 改变窗口的大小时
  - 把窗口从另一个窗口背后移出时
  - 窗口显示数据变化时，应用程序引发重绘操作
    - **通过CWnd::Invalidate、CWnd::InvalidateRect或CWnd::InvalidateRgn函数把指定区域加到窗口的Update Region中。**
    - **窗口的Update Region不为空时，系统会自动产生WM_PAINT消息。**

CClientDC

- **CClientDC用于窗口客户区（窗口中除边框、标题栏、菜单栏、状态栏外的中间部分）的输出，其构造函数中包含了GetDC,析构函数中包含了ReleaseDC，不需要显式释放DC资源。**
- **一般用于响应非重绘消息（如键盘和鼠标消息）的绘图操作。**
- **坐标原点(0,0)是客户区的左上角。**



CWindowDC

- **CWindowDC在整个窗口上画图，而CClientDC和CPaintDC只能在客户区绘制图形。**
- **除非要自己绘制窗口边框和按钮，否则一般不用它。**

设备环境类的基本功能

- **设备环境类**

设备环境类	功能描述
CDC	所有设备环境类的基类，对GDI的所有绘图函数进行了封装；可用来直接访问整个显示器或非显示设备（如打印机）的上下文。
CPaintDC	用于响应窗口重绘消息(WM_PAINT)，用于窗口客户区画图(仅限于OnPaint()处理函数)
CClientDC	代表窗口客户区的设备环境，响应非窗口消息对客户区绘图时用到（除OnPaint外其它任何处理程序）。
CWindowDC	代表整个窗口的设备环境，除非要自己绘制窗口边框和按钮，否则一般不用它。
CMetaFileDC	代表Windows图元文件的设备环境，存储GDI命令序列，这些命令可重新执行，产生实际的输出。

利用设备环境类绘图

CWnd 类的成员函数OnPaint() 响应 WM_PAINT消息

```
void CView::OnPaint()
{
   CPaintDC dc(this);
   OnPrepareDC(&dc);
   OnDraw(&dc); //调用了OnDraw
}
```

- **OnPaint函数调用了CView::OnDraw函数。因而一般在OnDraw函数内添加绘图代码，完成绘图任务。**

利用设备环境类绘图

OnDraw函数: 视类的成员函数

- **OnDraw函数由OnPaint()函数调用， 参数为一个指向CDC 对象的指针**

```
void OnDraw(CDC* pDC)
{
// 使用pDC进行绘图
}
```

主要内容

- **GDI与设备环境（DC）**
- **CDC类**
- 应用举例
- **用GDI绘图**

MFC绘图程序举例

- **编写一个单文档MFC应用程序，完成以下要求：**

（1）在客户区中画最大的椭圆

（2）当点击鼠标左键时，以鼠标左键点击的位置为中心，画一个半径为20个像素的圆

MFC绘图程序举例

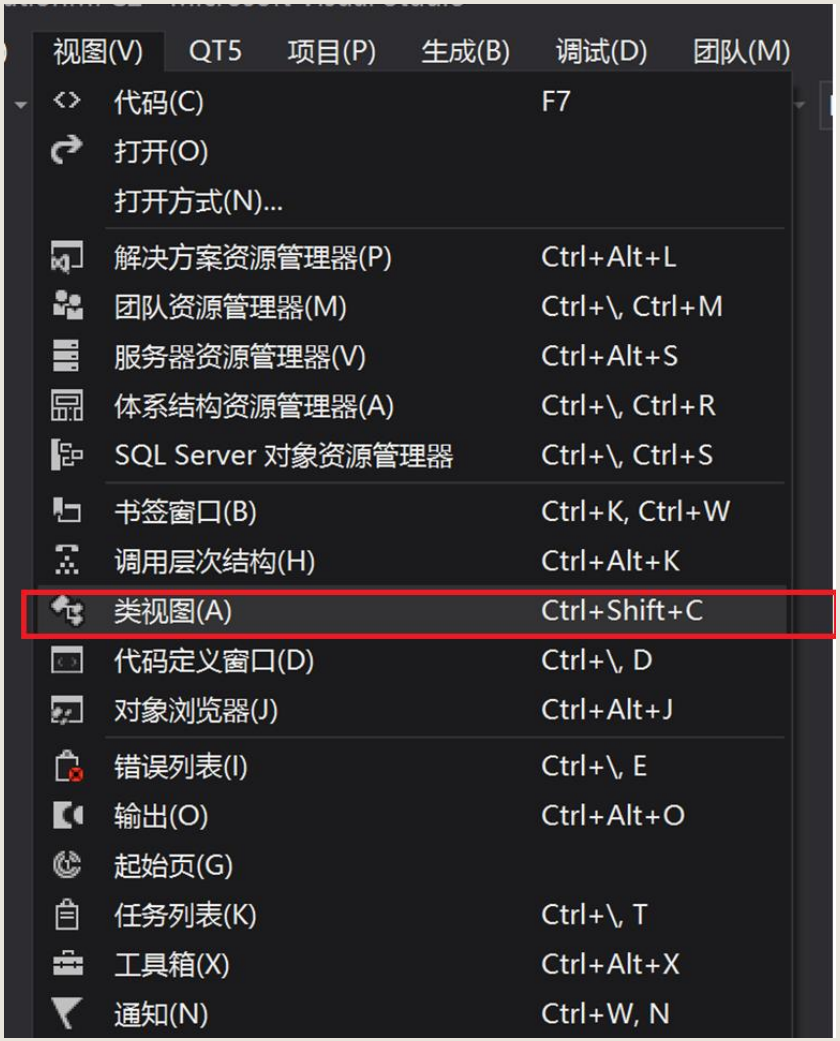
- **绘制椭圆方法一：修改CView类的虚函数OnDraw**
  - **函数CWnd::GetClientRect得到客户区域的大小**

```
void CMFCApplicationMFC2View::OnDraw(CDC* pDC/*pDC*/)
{
    CMFCApplicationMFC2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    CRect rect;
    GetClientRect(&rect);
    pDC->Ellipse(rect);
}
```

添加绘图代码

MFC绘图程序举例

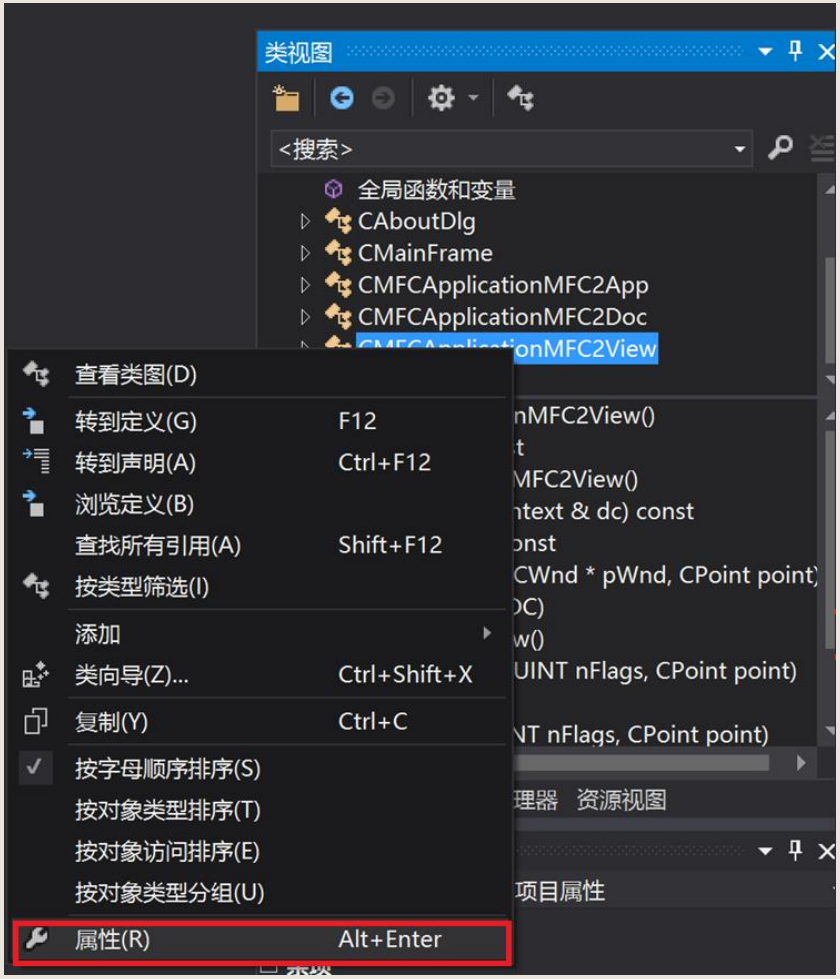
- 绘制椭圆方法二：在OnPaint函数中添加绘图代码
  - 添加WM_PAINT消息处理函数OnPaint
    1. 打开类视图
    2. 右键选中视类，点击属性，打开属性窗口
    3. 点击消息按钮



MFC绘图程序举例

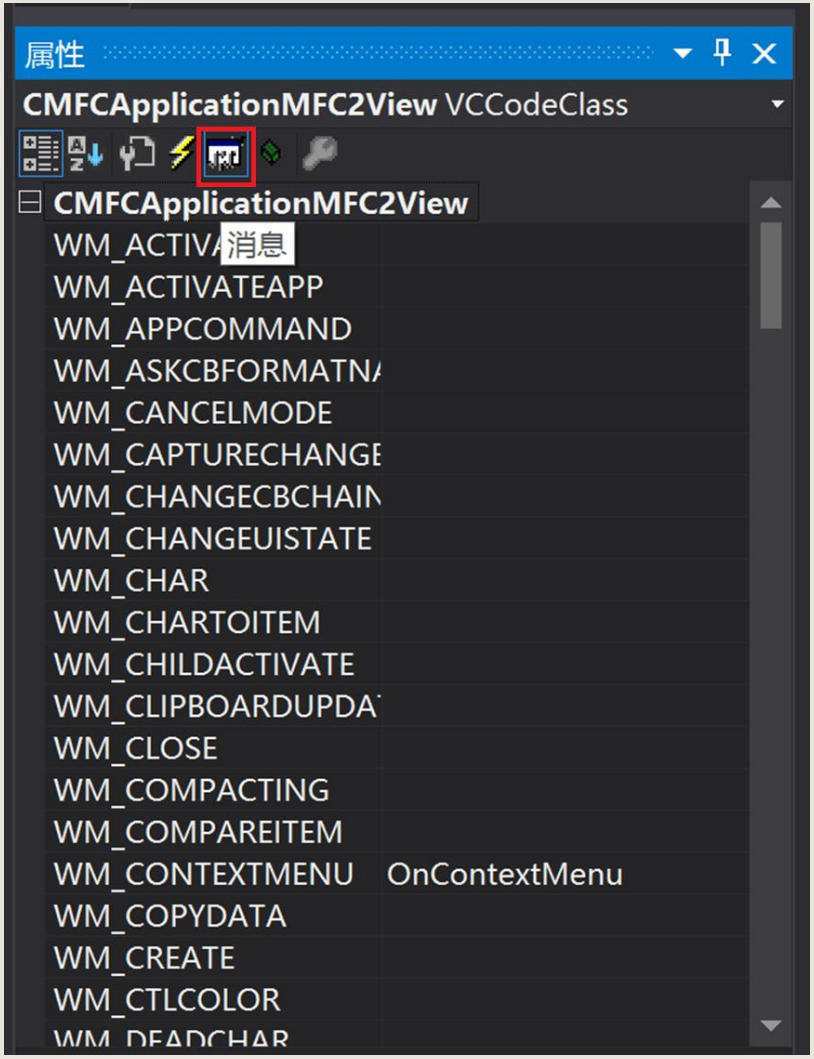
- **绘制椭圆方法二：在OnPaint函数中添加绘图代码**
  - **添加WM_PAINT消息处理函数OnPaint**

1. 打开类视图
2. 右键选中视类，点击属性，打开属性窗口
3. 点击消息按钮



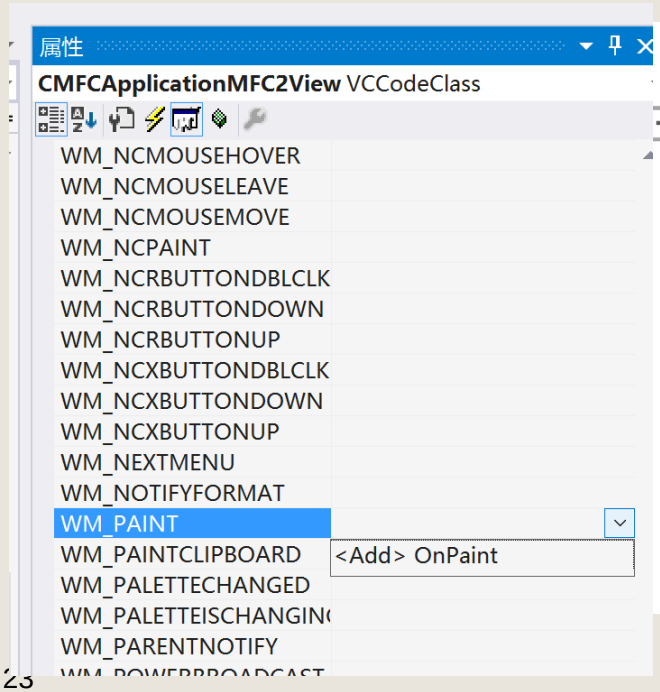
MFC绘图程序举例

- **绘制椭圆方法二：在OnPaint函数中添加绘图代码**
  - **添加WM_PAINT消息处理函数OnPaint()**
    1. 打开类视图
    2. 右键选中视类，点击属性，打开属性窗口
    3. 点击消息按钮



MFC绘图程序举例

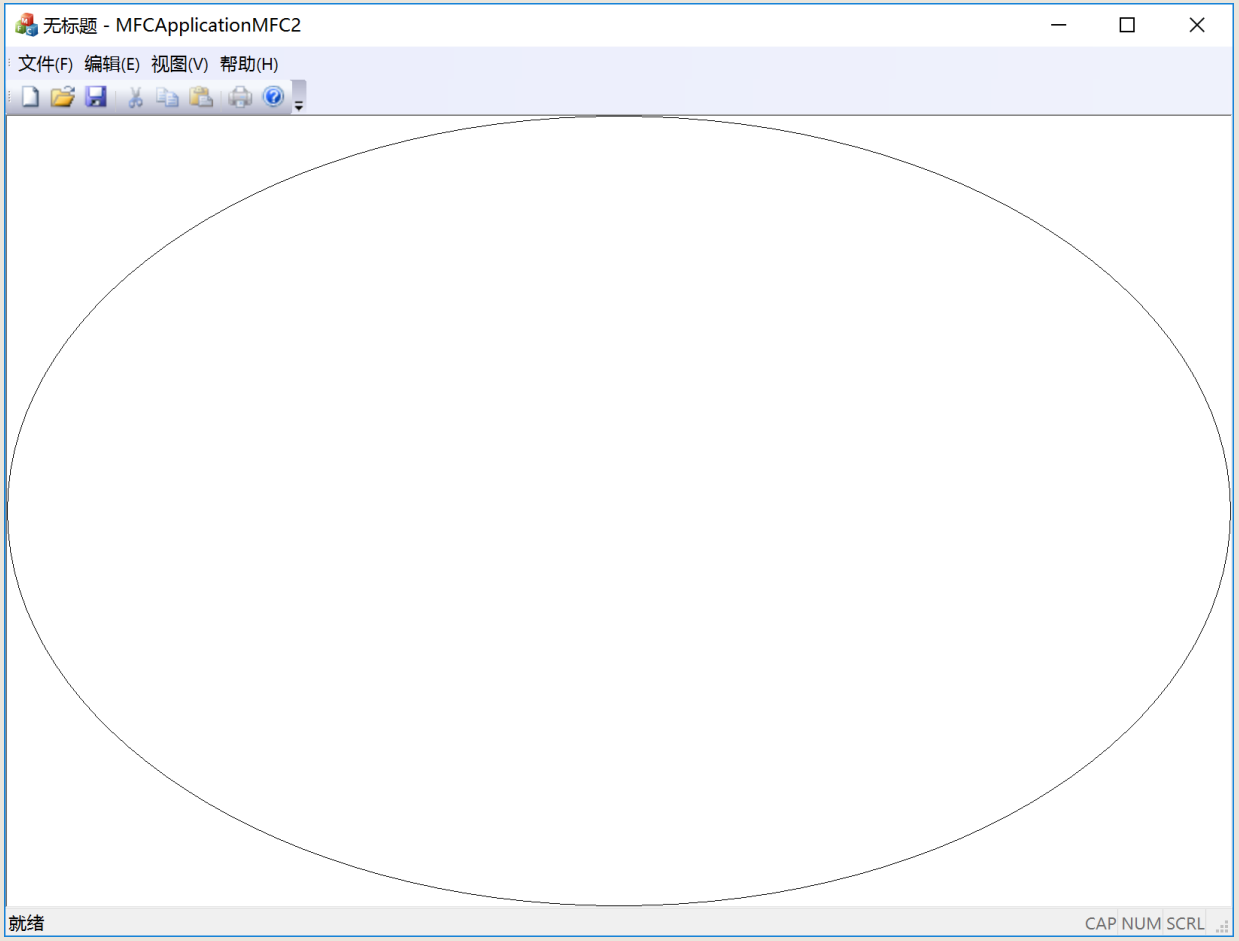
- **绘制椭圆方法二：在OnPaint函数中添加绘图代码**
  - **添加WM_PAINT消息处理函数OnPaint**
  - **添加绘图代码**




```
void CMFCApplicationMFC2View::OnPaint()
{
    CPaintDC dc(this); // device context for
    CRect rect;
    GetClientRect(&rect);
    dc.Ellipse(rect);
}
```

需要添加的代码

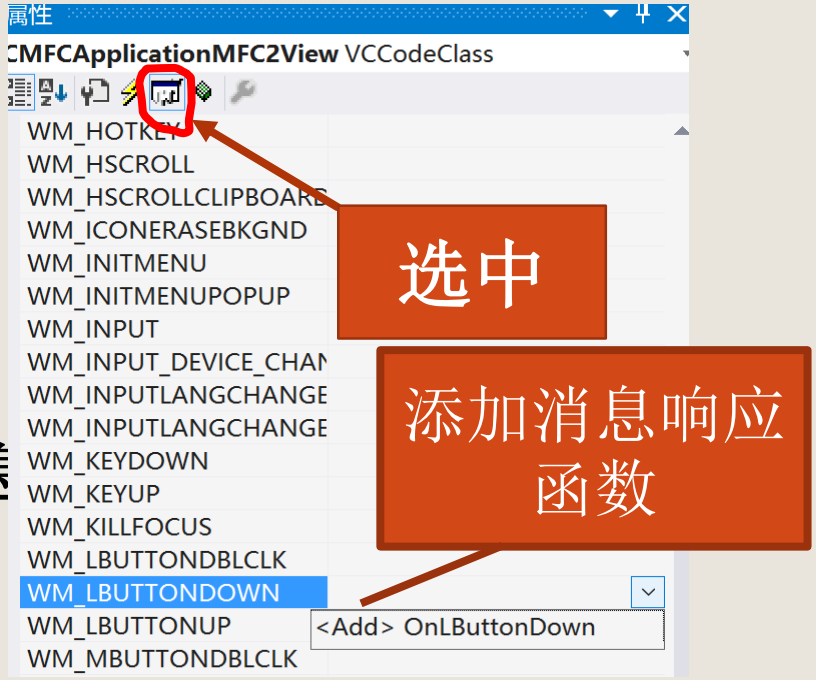
MFC绘图程序举例



MFC绘图程序举例

如何在通过鼠标点击绘图？

- 在类视图中选中**MFCApplication1MFC2View类**
- 在属性窗口中选择“**WM_LBUTTONDOWN**”消息添加消息处理函数

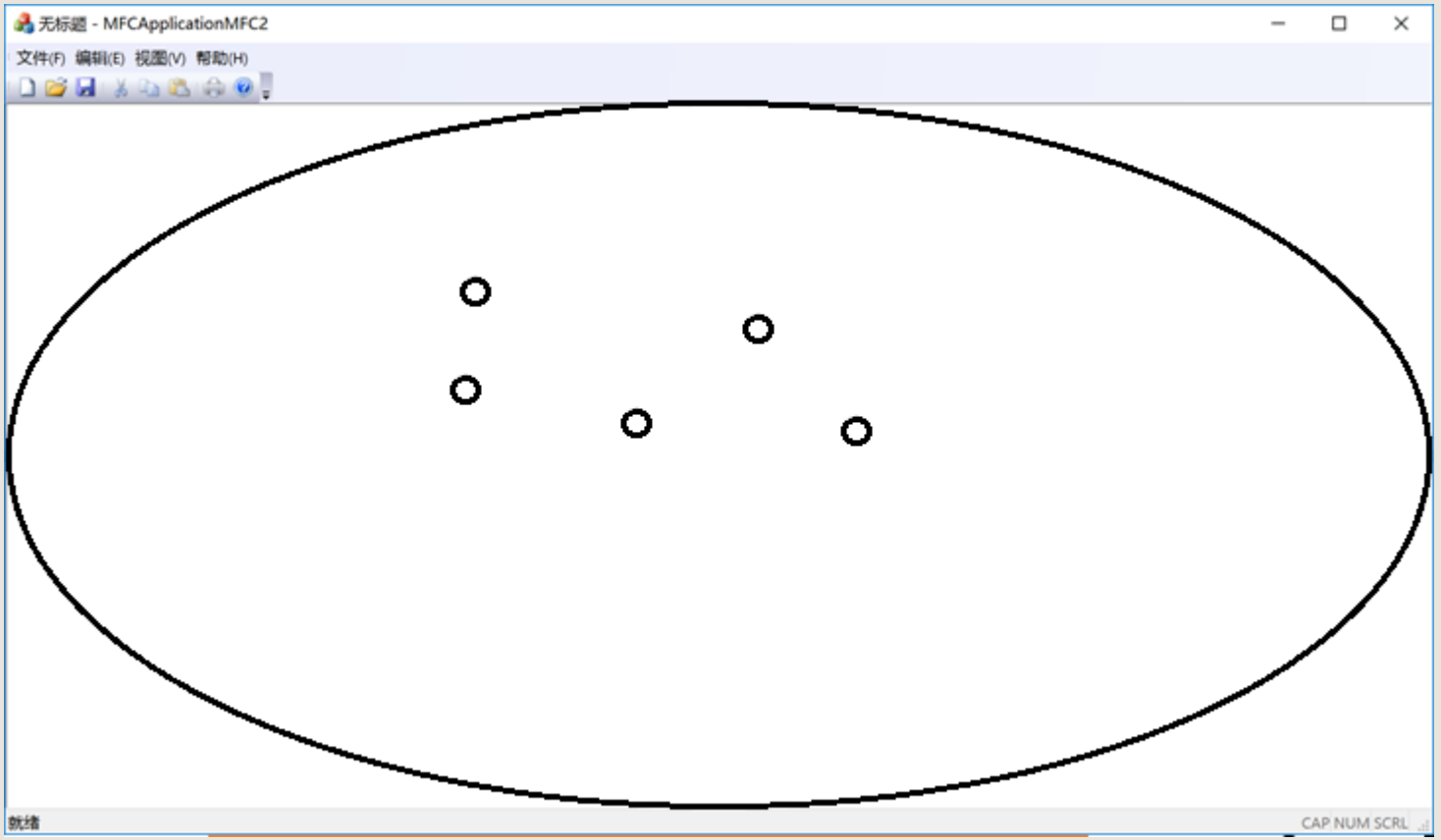



```
void CMFCApplicationMFC2View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    dc.Ellipse(point.x-20, point.y-20, point.x+20, point.y+20);
    CView::OnLButtonDown(nFlags, point);
}
```

需要添加的代码

MFC绘图程序举例

- 问题： 刷新屏幕后，小圆圈消失, 为什么？



MFC绘图程序举例

- 如何解决？

保存圆圈，在屏幕刷新时进行绘制

在视类的声明文件.h里

```
#include <vector>

std::vector<CPoint>  m_plist;
```

MFC绘图程序举例

- 保存圆圈

在视类的实现文件.cpp里

```
Void OnLButtonDown(UINT nFlags, CPoint point) {
        m_plist.push_back(point);
        CRect rect;
        GetClientRect(&rect);
        InvalidateRect(rect, NULL);
}
```

发出WM_PAINT消息

MFC绘图程序举例

- 保存圆圈

在视类的实现文件.cpp里

```
 void OnDraw(CDC * pDC)
{
    for(auto point:m_plist)
    {
        pDC->Ellipse(point.x - 20, point.y - 20,
                     point.x + 20, point.y + 20);
    }
}
```

主要内容

- **GDI与设备环境（DC）**
- **CDC类**
- **应用举例**
- **用GDI绘图**

常用绘图函数

- **Arc：绘制一段圆弧**
- **Chord：绘制弦形**
- **Ellipse：绘制椭圆或圆**
- **MoveTo：移动到指定位置**
- **LineTo：从当前位置到指定位置画一条直线**
- **Polyline：画连接指定点的折线段**
- **PolyBezier：画贝塞尔曲线**
- **Pie：画饼状**
- **Polygon：根据顶点绘制一个多边形**
- **Rectangle：根据指定的左上角和右下角坐标绘制一个矩形**
- **RoundRect：画圆角矩形**
- **SetPixel：画一个点**

绘图颜色

- **DWORD类型的COLORREF数据用于存放颜色值“0x00bbggrr”**
  - **低位字节存放红色强度值**
  - **第2 个字节存放绿色强度值**
  - **第3个字节存放蓝色强度值**
  - **高位字节存放0**

绘图颜色

- 可用RGB宏设置颜色值，将红、绿、蓝分量值转换为COLORREF类型的颜色数据

```
COLORREF RGB(
  BYTE byRed, // red component of color
  BYTE byGreen, // green component of color
  BYTE byBlue // blue component of color );
```

例如：

```
CBrush  brush(RGB(255, 0,0));
```

画点与画线函数操作

1．画点

CDC的成员函数SetPixel用来在指定位置上绘制一个特定的像素点，其原型为：

COLORREF SetPixel(POINT point, COLORREF crColor);

其中参数point指定所绘制的点，crColor指定画点所用的颜色。

倘若要在屏幕的（100,100）处画一个红色点，则代码为：

```
pDC-> SetPixel(CPoint(100,100),RGB(255,0,0));
```

画点与画线函数操作

2．画折线

PolyLine()函数用于画一条折线，它的原型如下所示。

BOOL Polyline(LPPOINT lpPoints, int nCount);

其中lpPoints是指向折线顶点数组的指针，而cCount则指定折线顶点数组中的顶点数目。

画弧线

3．画弧线

使用CDC的成员函数Arc和ArcTo，可以用缺省的笔画一段不填充的椭圆弧。Arc函数的原型如下：

```
BOOL Arc( int x1, int y1, int x2, int y2, int x3, int y3, int x4, int  y4 );
BOOL Arc( LPCRECT lpRect, POINT ptStart, POINT ptEnd );
```

这两个函数画弧成功返回非0，否则返回0，函数中各参数的含义如下：

参数x1与y1为包围弧的矩形的左上角x、y坐标；x2与y2为包围弧的矩形的右下角x、y坐标；x3与y3为弧的起点x、y坐标；x 4 与y 4 为弧的终点x、y坐标。

参数lpRect表示围绕弧的矩形，它可以是LPRECT 或CRect对象，ptStart表示弧的起点的CPoint 或POINT对象，该点不必精确地位于弧上；PtEnd表示弧的终点的CPoint 或POINT对象，该点不必精确地位于弧上。

绘制封闭图形

Windows提供了Rectangle、Ellipse、RoundRect、Chord、Pie等五个函数用来绘制并填充图形。

1．画矩形

BOOL Rectangle(int x1, int y1, int x2, int y2);

BOOL Rectangle(LPCRECT lpRect);

此函数成功调用后返回非0值，否则返回0。

参数(x1,y1)为指定矩形的左上角x与y坐标；（x2,y2）为指定矩形右下角的x与y坐标。

参数LpRect为一个矩形结构的指针，用它来表示矩形的四个角。

绘制封闭图形

2．画椭圆或圆

使用CDC的成员函数Ellipse，可以使用当前笔绘制一个用当前画刷填充的椭圆或圆。其函数原型如下

BOOL Ellipse(int x1, int y1, int x2, int y2);

BOOL Ellipse(LPCRECT lpRect);

这两个函数画椭圆成功后返回非0值，否则返回0。

所画椭圆高度为y2-y1,宽度为x2-x1。椭圆由其外接矩形确定，外接矩形的中心与椭圆中心重合，矩形的长和宽和椭圆的长短轴相等。函数中分别表示椭圆外接矩形的左上角和右下角坐标。

绘制封闭图形

3．画圆角矩形

BOOL RoundRect(int x1, int y1, int x2, int y2, int x3, int y3);

该函数用于绘制一个圆角矩形，并用当前的画刷来填充该圆角矩形的内部区域。

参数(x1,y1)为指定矩形的左上角位置x与y坐标；（x2,y2）为指定矩形右下角位置x与y坐标，(x3,y3)用于定义矩形四个角上的边角内切椭圆的宽度和高度，值越大，圆角矩形的角就越明显。如果x3=x2-x1，并且y3=y2-y1，则所绘制的圆角矩形变为一个椭圆。

绘制封闭图形

4. 画饼图扇形

饼图是一条弧和从弧的两个端点到中心的连线组成的图形。CDC的成员函数Pie可用于画饼图，函数原型如下：

BOOL Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);

BOOL Pie(LPCRECT lpRect, POINT ptStart, POINT ptEnd);

该函数的参数与Arc函数的参数的含义相仿，只不过Pie函数画的是封闭图形，Arc画的是非封闭图形。各参数参见7.3.1节介绍。

绘制封闭图形

5. 画弓形

弓形图是一条椭圆弧和连接该弧线两个端点的弦，并用当前的画刷来填充其内部区域的封闭图形。Chord原型如下：

BOOL Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);

该函数参数与Pie函数参数的含义相仿。

例子

```
void CMFCApplicationMFC2View::OnDrawPixel()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    CPoint p(100, 100);
    dc.SetPixel(p, RGB(200, 0, 0));
}
```

例子

```
void CMFCApplicationMFC2View::OnDrawLine()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    dc.MoveTo(1000,500);
    dc.LineTo(1500,600);
}
```

例子

```
void CMFCApplicationMFC2View::OnDrawPolyline()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    CPoint plist[6];
    plist[0] = CPoint(500, 400);
    plist[1] = CPoint(600, 300);
    plist[2] = CPoint(800, 500);
    plist[3] = CPoint(700, 650);
    plist[4] = CPoint(900, 500);
    plist[5] = CPoint(1000, 400);

    dc.Polyline(plist, 6);
}
```

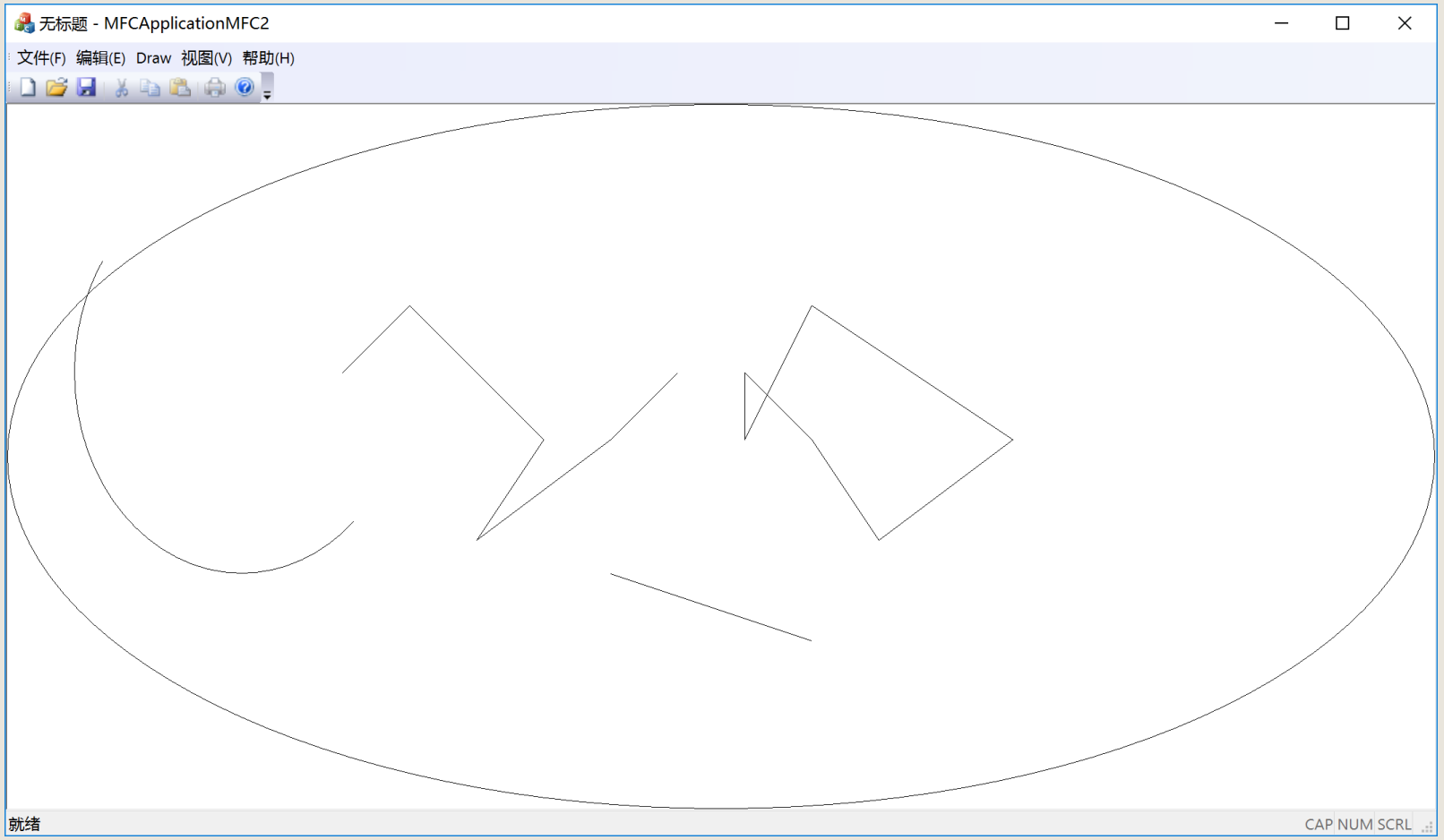
例子

```
void CMFCApplicationMFC2View::OnDrawRectangle()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    CPoint plist[6];
    plist[0] = CPoint(500, 400);
    plist[1] = CPoint(600, 300);
    plist[2] = CPoint(800, 500);
    plist[3] = CPoint(700, 650);
    plist[4] = CPoint(900, 500);
    plist[5] = CPoint(1000, 400);
    dc.Polygon(plist, 6);
}
```

例子

```
void CMFCApplicationMFC2View::OnDrawCircle()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    CRect  rect(100, 100, 600, 700);
    CPoint p1(100, 200);
    CPoint p2(500, 600);
    dc.Arc(rect, p1, p2);
}
```

例子



设备描述表属性

图形 = 形+图

- 用CDC输出函数在屏幕上画图时，某些属性（例如颜色）并没有在函数调用过程中规定。
- 这些属性保存在设备描述表（ CDC ）中
- 这些属性可通过设备描述表获得

主要设备描述表属性

Attribute	Default	Set with	Get with
文本颜色	Black	CDC::SetTextColor	CDC::GetTextColor
背景颜色	White	CDC::SetBkColor	CDC::GetBkColor
背景模式	OPAQUE	CDC::SetBkMode	CDC::GetBkMode
映射模式	MM_TEXT	CDC::SetMapMode	CDC::GetMapMode
绘图模式	R2_COPYPEN	CDC::SetROP2	CDC::GetROP2
当前位置	(0,0)	CDC::MoveTo	CDC::GetCurrentPos
当前画笔	BLACK_PEN	CDC::SelectObject	CDC::SelectObject
当前画刷	WHITE_BRUSH	CDC::SelectObject	CDC::SelectObject
当前字体	SYSTEM_FONT	CDC::SelectObject	CDC::SelectObject

设备描述表属性

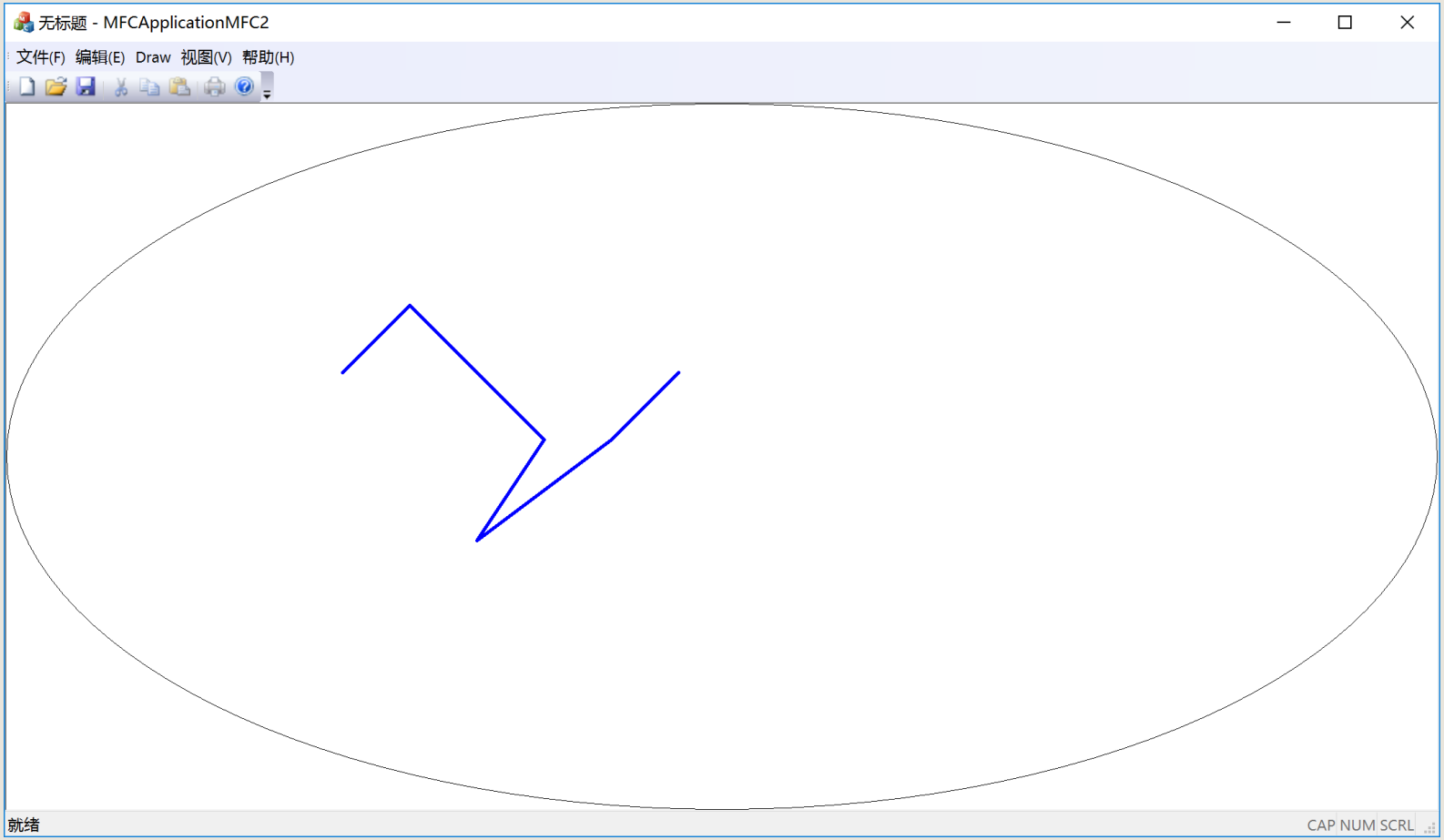
例子：

```
void CMFCApplicationMFC2View::OnDrawPolyline()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    CPen pen;
    pen.CreatePen(PS_SOLID, 5, RGB(0,0,255));
    dc.SelectObject(pen);

    CPoint plist[6];
    plist[0] = CPoint(500, 400);
    plist[1] = CPoint(600, 300);
    plist[2] = CPoint(800, 500);
    plist[3] = CPoint(700, 650);
    plist[4] = CPoint(900, 500);
    plist[5] = CPoint(1000, 400);

    dc.Polyline(plist, 6);
}
```

定义画笔

选择画笔

Cpen 例子



设备描述表属性

定义设备描述表的属性：

- **常用函数： SelectObject()**

例子： 下面6个GDI对象，可由SelectObject选入设备描述表

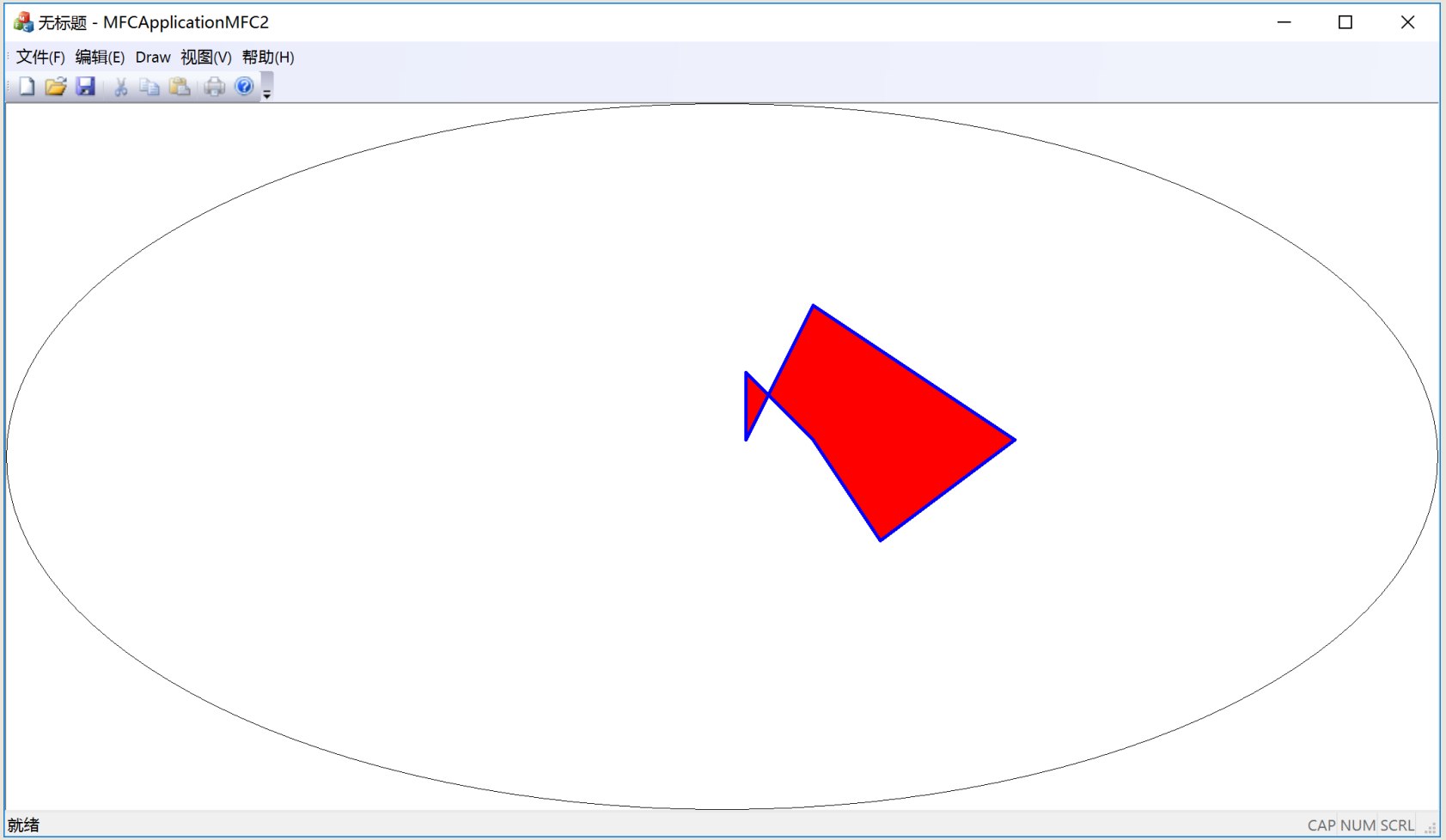
画笔Pen 画刷 Brush 字体Font
位图Bitmap 调色板Palette 区域Region

```
dc.SelectObject(pPen);
dc.SelectObject(pBrush);
dc.Ellipse(0,0,100,100);
```

CBrush 例子

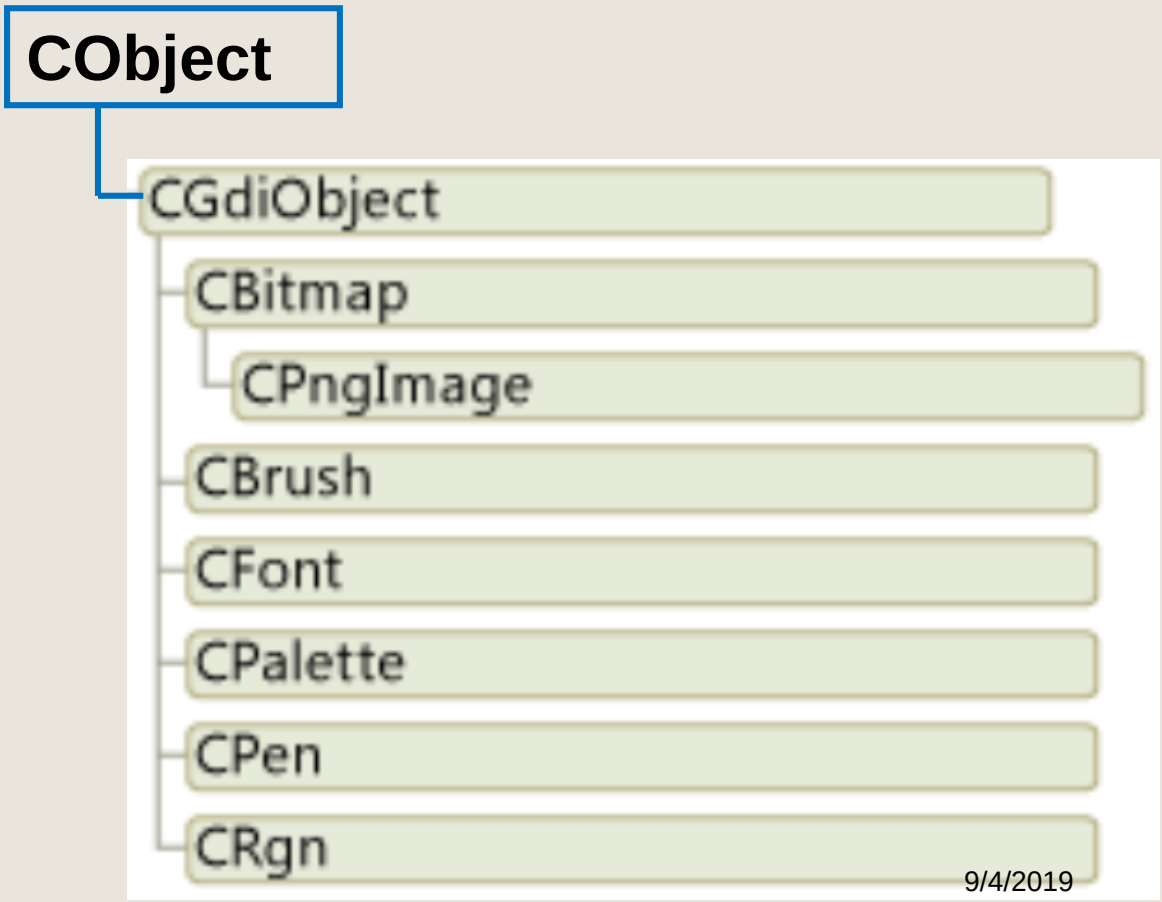
```
void CMFCApplicationMFC2View::OnDrawRectangle()
{
    // TODO:  在此添加命令处理程序代码
    CClientDC dc(this);
    CPen pen;
    pen.CreatePen(PS_SOLID, 5, RGB(0, 0, 255));
    dc.SelectObject(pen);
    CBrush brush;
    brush.CreateSolidBrush(RGB(255,0,0));
    dc.SelectObject(brush);

    CPoint plist[6];
    plist[0] = CPoint(1100, 500);
    plist[1] = CPoint(1200, 300);
    plist[2] = CPoint(1500, 500);
    plist[3] = CPoint(1300, 650);
    plist[4] = CPoint(1200, 500);
    plist[5] = CPoint(1100, 400);
    dc.Polygon(plist, 6);
}
```

CBrush 例子



绘图工具类CGdiObject

- 绘图工具类主要包括画笔CPen、画刷Cbrush、字体CFont、位图Bitmap和调色板Cpalette等。
- 类继承关系如图



绘图工具类CGdiObject

- **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。**
- **CBrush类：GDI 画刷，用来填充一个封闭图形对象(如矩形、圆形)的内部区域，默认的画刷颜色是白色。**
- **CFont类：GDI字体，用来绘制文本，可设置文字的大小、是否加粗、是否斜体、是否加下划线等。**
- **CBitmap类：GDI 位图，用于填充区域。**
- **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。**

例子：交互绘制一个椭圆

- 鼠标左键按下开始绘制
- 鼠标移动中绘制椭圆
- 鼠标左键抬起，绘制椭圆固定住

例子：交互绘制一个椭圆

- 阴影画刷

```
BOOL Cbrush::CreateHatchBrush(
        int nIndex, // 指定阴影样式
        COLORREF crColor // 指定阴影颜色);
```

- 创建一个黄色的实心画刷

```
CBrush brush;
brush.CreateHatchBrush(HS_DIAGCROSS, RGB(255,255,0));
```

阴影样式	说　明
HS_BDIAGONAL	从左下角到右上角的 45 度阴影线
HS_FDIAGONAL	从左上角到右下角的 45 度阴影线
HS_DIAGCROSS	十字交叉的 45 度阴影线
HS_ CROSS	水平和垂直交叉的阴影线
HS_HORIZONTAL	水平阴影线
HS_VERTICAL	垂直阴影线

例子：交互绘制一个椭圆

预定义的画笔、画刷或字体

- 使用CDC类的SelectStockObject函数，可以使用

  CGdiObject* SelectStockObject(int nIndex);

  pDC-> SelectStockObject(GRAY_BRUSH);

- nIndex的部分可取值如下（完整参数列表可参考MSDN）
  - BLACK_BRUSH：黑色画刷
  - DKGRAY_BRUSH：深灰色画刷
  - GRAY_BRUSH：灰色画刷
  - HOLLOW_BRUSH：中空画刷，不做填充
  - LTGRAY_BRUSH：浅灰色画刷

例子：交互绘制一个椭圆

为视类增加以下成员变量

```
CPoint  m_StartPoint;
CPoint  m_EndPoint;
BOOL   m_Drawing;//初始化为FALSE
```

例子：交互绘制一个椭圆

在鼠标左键按下响应函数

OnLButtonDown(UINT nFlags, **Cpoint point)**

中添加代码

```
if(!m_Drawing)
{
    m_StartPoint = point;
    m_Drawing =  true;
}
```

例子：交互绘制一个椭圆

在鼠标左键抬起响应函数

OnLButtonUp(UINT nFlags, **Cpoint point)**

中添加代码

```
if(m_Drawing)
    m_Drawing =  false;

CRect rect;
GetClientRect(&rect);
InvalidateRect(rect);
```

例子：交互绘制一个椭圆

在鼠标移动 响应函数OnMouseMove(UINT nFlags, CPoint point)**中添加代码**

```
CRect rect;
GetClientRect(&rect);

if (nFlags&MK_LBUTTON)
{
  m_EndPoint = point;
  InvalidateRect(rect);
}
```

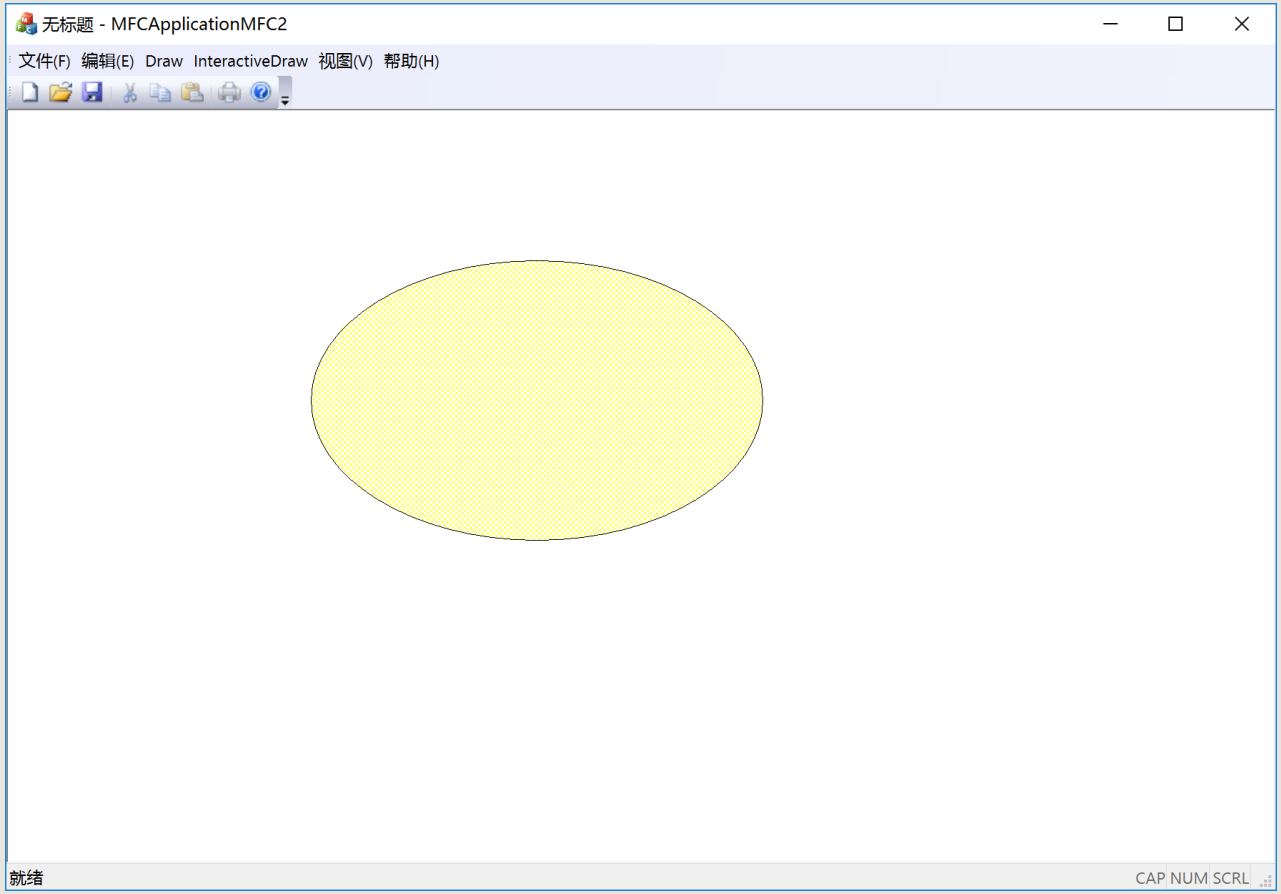
例子：交互绘制一个椭圆

绘制： 在 OnDraw(CDC * pDC)中添加代码

```
CRect rect2;
rect2.SetRect(m_StartPoint, m_EndPoint);
pDC->Ellipse(rect);
```

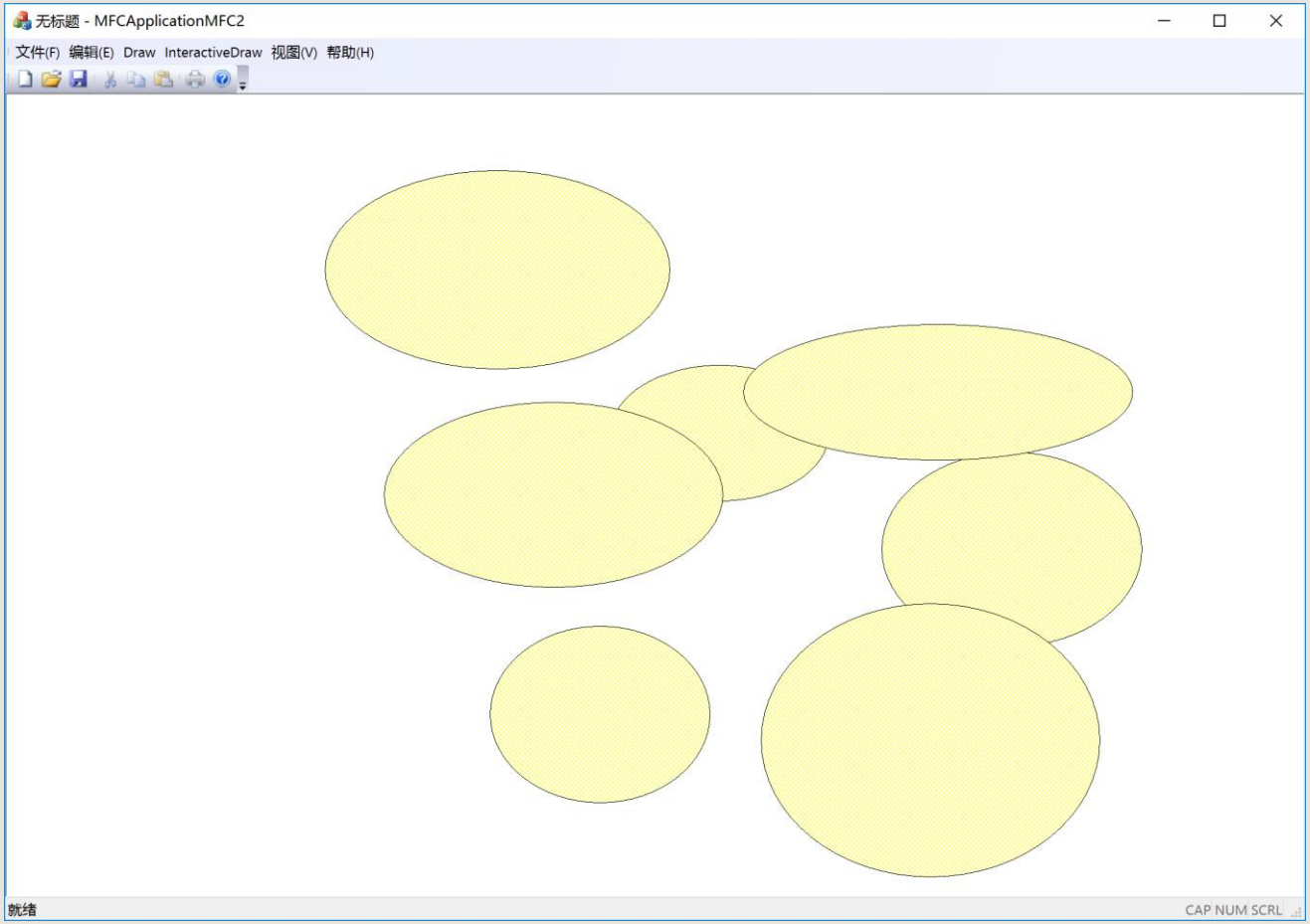
例子：交互绘制一个椭圆

- 运行结果如图。同时只能绘制一个椭圆。



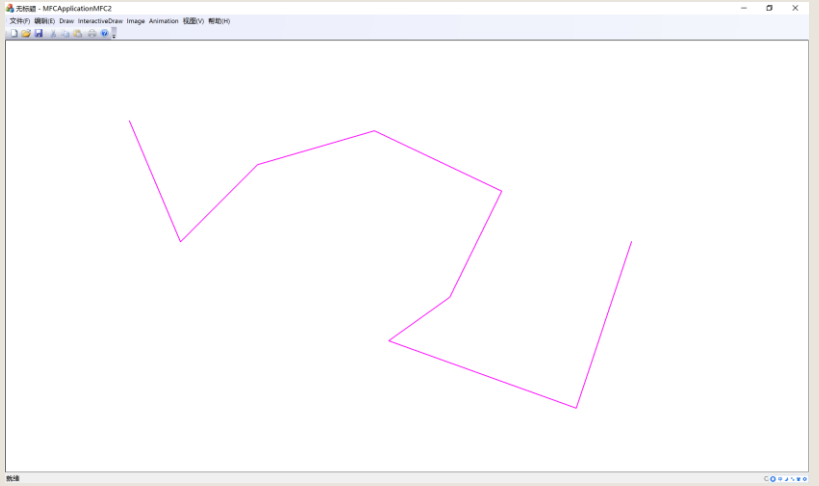
例子：交互绘制一个椭圆

- 思考：如何显示多个椭圆？



作业

- 创建一个单文档程序
- 能够交互绘制折线
- 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

附录

以下为附录

设置绘图参数

绘图参数主要包括文本的前景色、文本的背景色、背景模式、绘图模式、位图伸展模式等，恰当地设置绘图参数，可以达到各种绘图效果。

文本前景色靠**CDC::SetTextColor()**来控制

文本的背景色靠**CDC::SetBkColor()**来控制

绘图模式

- **GDI将像素点输出到逻辑显示平面上时，通过一系列的布尔运算将输出像素点和目标位置的像素点的颜色进行合成。**
- **使用的逻辑关系又设备描述表当前的绘图模式确定**
- **调用CDC的成员函数SetROP2改变绘图模式，函数原型为：**

```
int SetROP2(int nDrawMode);
```

绘图模式

GDI 绘图模式

绘图模式	执行的运算
R2_NOP	dest = dest
R2_NOT	dest = NOT dest
R2_BLACK	dest = BLACK
R2_WHITE	dest = WHITE
R2_COPYPEN	dest = src
R2_NOTCOPYPEN	dest = NOT src
R2_MERGEPENNOT	dest = (NOT dest) OR src
R2_MASKPENNOT	dest = (NOT dest) AND src
R2_MERGENOTPEN	dest = (NOT src) OR dest
R2_MASKNOTPEN	dest = (NOT src) AND dest
R2_MERGEPEN	dest = dest OR src
R2_NOTMERGEPEN	dest = NOT (dest OR src)
R2_MASKPEN	dest = dest AND src
R2_NOTMASKPEN	dest = NOT (dest AND src)
R2_XORPEN	dest = src XOR dest
R2_NOTXORPEN	dest = NOT (src XOR dest)

绘图模式

绘图模式	说明	布尔操作（P为画笔像素D为目标像素）
R2_BLACK	像素最终颜色为黑色	0
R2_WHITE	像素最终颜色为白色	1
R2_NOP	像素颜色没有变化,还是原先目标像素颜色	D
R2_NOT	像素最终颜色为原来颜色的反色	~D
R2_COPYPEN(缺省)	像素最终颜色为当前画笔的颜色	P
R2_NOTCOPYPEN	像素最终颜色为当前画笔颜色的反色	~P
R2_MERGEPEN	像素最终颜色为当前画笔颜色P和原来颜色D的逻辑或	P|D
R2_MERGEPENNOT	像素最终颜色为当前画笔颜色P和原来颜色反色~D的逻辑或	P|~D
R2_MASKPEN	像素最终颜色为当前画笔颜色P和原来颜色D的逻辑与	P&D

绘图模式

绘图模式	说明	布尔操作（P为画笔像素D为目标像素）
R2_MASKPENNOT	像素最终颜色为当前画笔颜色P和原来颜色反色~D的逻辑与	P&~D
R2_MERGENOTPEN	像素最终颜色为当前画笔颜色P的反色和原来颜色的逻辑或	~P|D
R2_MASKNOTPEN	像素最终颜色为当前画笔颜色的反色和原来颜色的逻辑与	~P&D
R2_NOTMERGEPEN	像素最终颜色为R2_MERGEPEN结果的反色	~(P|D)
R2_NOTMASKPEN	像素最终颜色为R2_MASKPEN结果的反色	~(P&D)
R2_XORPEN	像素最终颜色为当前画笔颜色和原来颜色的异或结果	P^D
R2_NOTXORPEN	像素最终颜色为R2_XORPEN结果的反色	~(P^D)

绘图模式

注意:

(1) 在画线时,如果绘图模式为R2_NOT,则画出的线的颜色为原先屏幕颜色的反色。这样画的线均可见，而且第二次画同一条线时，将自动擦除该线并恢复为当前显示颜色。

(2) R2_NOP绘图模式等效于同时选择NULL画笔和NULL画刷。

(3) 在R2_XORPEN的操作模式下，用相同的参数两次调用某一条绘图命令时（包括文字输出函数），其结果仍然保留原样，利用这种方式，我们可以实现图形动画效果。

绘图模式

注意:

(4) 在画虚线时，用于填充线间的空白颜色取决于当前背景模式和背景颜色。

(5) 在R2_NOTXORPEN的操作模式下，其功能有二，第一，将同一直线绘制两次，将删除该直线；第二，无论背景是何颜色，第一次绘制的直线总是可见的。