# 学习大纲

|  |
| --- |
|  |

## Polygon函数的用法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Polygon** 函数绘制一个由两个或多个直线连接的顶点组成的多边形。 多边形使用当前笔进行轮廓，并使用当前画笔和多边形填充模式进行填充。 语法 C++复制  BOOL Polygon(  [in] HDC hdc,  [in] const POINT \*apt,  [in] int cpt  ); 参数 [in] hdc  设备上下文的句柄。  [in] apt  指向 [POINT](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point) 结构的数组的指针，该数组以逻辑坐标形式指定多边形的顶点。  [in] cpt  数组中的顶点数。 此值必须大于或等于 2。 返回值 如果该函数成功，则返回值为非零值。  如果函数失败，则返回值为零。 注解 多边形通过绘制从最后一个顶点到第一个顶点的线条自动关闭。  **Polygon** 函数既不使用也不更新当前位置。  忽略任何额外的点。 若要绘制具有更多点的线条，请将数据划分为多个组，其中每个组的点数都小于最大点数，并为每个点组调用 函数。 请记得连接线段。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [填充的形状函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/filled-shape-functions)  [填充形状概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/filled-shapes)  [GetPolyFillMode](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getpolyfillmode)  [点](https://learn.microsoft.com/zh-cn/windows/win32/api/windef/ns-windef-point)  [PolyPolygon](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polypolygon)  [折线](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polyline)  [PolylineTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-polylineto)  [SetPolyFillMode](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setpolyfillmode) |

## SetPolyFillMode函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SetPolyFillMode** 函数为填充多边形的函数设置多边形填充模式。 语法 C++复制  int SetPolyFillMode(  [in] HDC hdc,  [in] int mode  ); 参数 [in] hdc  设备上下文的句柄。  [in] mode  新的填充模式。 此参数的取值可为下列值之一：  展开表   |  |  | | --- | --- | | **值** | **含义** | | ALTERNATE | 选择备用模式 (填充每个扫描线) 上奇数和偶数多边形面之间的区域。 | | WINDING | 选择绕组模式 (使用非零绕组值) 填充任何区域。 |  返回值 返回值指定以前的填充模式。 如果发生错误，则返回值为 0。 注解 通常，这些模式仅在必须填充复杂重叠多边形 (的情况不同，例如，形成五角star的五角多边形，中心) 五角形。 在这种情况下，ALTERNATE 模式会填充多边形 (即star) 的点，但 ALTERNATE 模式填充所有区域， (即点和五角形) 。  当填充模式为 ALTERNATE 时，GDI 将填充每个扫描线上奇数和偶数多边形边之间的区域。 也就是说，GDI 填充第一侧和第二侧、第三侧和第四侧之间的区域，依此填充。  当填充模式为 WINDING 时，GDI 将填充具有非零绕组值的任何区域。 此值定义为用于绘制多边形的笔围绕该区域的次数。 多边形的每个边缘的方向非常重要。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [GetPolyFillMode](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-getpolyfillmode)  [区域函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/region-functions)  [区域概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/regions) |

## GetPolyFillMode函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GetPolyFillMode** 函数检索当前多边形填充模式。 语法 C++复制  int GetPolyFillMode(  [in] HDC hdc  ); 参数 [in] hdc  设备上下文的句柄。 返回值 如果函数成功，则返回值指定多边形填充模式，可以是以下值之一。  展开表   |  |  | | --- | --- | | **值** | **含义** | | ALTERNATE | 选择备用模式 (填充每个扫描线) 奇数和偶数多边形面之间的区域。 | | WINDING | 选择绕组模式 (使用非零绕组值) 填充任何区域。 |     如果发生错误，则返回值为 0。 要求  |  |  | | --- | --- | | **标头** | wingdi.h (包括 Windows.h) | | **Library** | Gdi32.lib | | **DLL** | Gdi32.dll |  另请参阅 [区域函数](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/region-functions)  [区域概述](https://learn.microsoft.com/zh-cn/windows/desktop/gdi/regions)  [SetPolyFillMode](https://learn.microsoft.com/zh-cn/windows/desktop/api/wingdi/nf-wingdi-setpolyfillmode) |

# 演练

## 1.新建一个win32项目，取名：Lesson45-poly-fillmode-demo

|  |
| --- |
|  |

## 2.定位到WndProc里面的WM\_PAINT消息处理代码中，先添加一些点用来绘制五角星

|  |
| --- |
|  |
|  |

### 我们发现，无论我们使用什么填充模式，五角星都没有变化，这是因为默认的背景色是白色的，不会有效果

## 3.我们获取一个库存画刷选择灰色的画刷，然后添加填充代码，先使用ALTERNATE模式

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

## 4.然后我们改为WINDING

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 默认的填充模式是ALTERNATE

### ALTERNATE填充原理

|  |
| --- |
|  |

#### 从多边形的中心开始往外扩展，如果一个点发出一条射线经过偶数条边，这个区域不填充，否则就填充

### WINDING模式的填充原理

|  |
| --- |
|  |

## 5.WINDING这种填充模式比较复杂，我们通过代码来理解它，需要定义一个新数组

|  |
| --- |
|  |
|  |

### 效果

|  |
| --- |
|  |

## 6.此时图形太小，不好看，我们需要把他放大一些，需要定义一个新数组来保存放大后的点，还需要一些变量保存客户区数据

|  |
| --- |
|  |

## 7.需要添加对WM\_SIZE消息的响应代码来获取窗口客户区的大小

|  |
| --- |
|  |

## 8.然后我们就可以来把原来的坐标放大

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 9.如果把填充模式改为ALTERNATE

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

## 10.我们可以修改一下代码，把两种模式的绘图放在一起对比

|  |
| --- |
|  |

### 效果：左边是ALTERNAMTE模式，右边是WINDING模式

|  |
| --- |
|  |

#### WINDING的填充规则是和边的绘制方向有关，从一个区域的一个点发出一条射线，如果和偶数条边相交，如果这两天边的绘制方向是一致的，就填充这个区域，如果方向不一致就不填充

# 这一节的学习到此为止，完整代码如下

## Lesson45-poly-fillmode-demo.cpp

|  |
| --- |
| // Lesson45-poly-fillmode-demo.cpp : 定义应用程序的入口点。  //  #include "stdafx.h"  #include "Lesson45-poly-fillmode-demo.h"  #define MAX\_LOADSTRING 100  // 全局变量:  HINSTANCE hInst; // 当前实例  TCHAR szTitle[MAX\_LOADSTRING]; // 标题栏文本  TCHAR szWindowClass[MAX\_LOADSTRING]; // 主窗口类名  // 此代码模块中包含的函数的前向声明:  ATOM MyRegisterClass(HINSTANCE hInstance);  BOOL InitInstance(HINSTANCE, int);  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);  INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);  int APIENTRY \_tWinMain(HINSTANCE hInstance,  HINSTANCE hPrevInstance,  LPTSTR lpCmdLine,  int nCmdShow)  {  UNREFERENCED\_PARAMETER(hPrevInstance);  UNREFERENCED\_PARAMETER(lpCmdLine);  // TODO: 在此放置代码。  MSG msg;  HACCEL hAccelTable;  // 初始化全局字符串  LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);  LoadString(hInstance, IDC\_LESSON45POLYFILLMODEDEMO, szWindowClass, MAX\_LOADSTRING);  MyRegisterClass(hInstance);  // 执行应用程序初始化:  if (!InitInstance (hInstance, nCmdShow))  {  return FALSE;  }  hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LESSON45POLYFILLMODEDEMO));  // 主消息循环:  while (GetMessage(&msg, NULL, 0, 0))  {  if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  }  return (int) msg.wParam;  }  //  // 函数: MyRegisterClass()  //  // 目的: 注册窗口类。  //  // 注释:  //  // 仅当希望  // 此代码与添加到 Windows 95 中的“RegisterClassEx”  // 函数之前的 Win32 系统兼容时，才需要此函数及其用法。调用此函数十分重要，  // 这样应用程序就可以获得关联的  // “格式正确的”小图标。  //  ATOM MyRegisterClass(HINSTANCE hInstance)  {  WNDCLASSEX wcex;  wcex.cbSize = sizeof(WNDCLASSEX);  wcex.style = CS\_HREDRAW | CS\_VREDRAW;  wcex.lpfnWndProc = WndProc;  wcex.cbClsExtra = 0;  wcex.cbWndExtra = 0;  wcex.hInstance = hInstance;  wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_LESSON45POLYFILLMODEDEMO));  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);  wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_LESSON45POLYFILLMODEDEMO);  wcex.lpszClassName = szWindowClass;  wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));  return RegisterClassEx(&wcex);  }  //  // 函数: InitInstance(HINSTANCE, int)  //  // 目的: 保存实例句柄并创建主窗口  //  // 注释:  //  // 在此函数中，我们在全局变量中保存实例句柄并  // 创建和显示主程序窗口。  //  BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)  {  HWND hWnd;  hInst = hInstance; // 将实例句柄存储在全局变量中  hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,  CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);  if (!hWnd)  {  return FALSE;  }  ShowWindow(hWnd, nCmdShow);  UpdateWindow(hWnd);  return TRUE;  }  //  // 函数: WndProc(HWND, UINT, WPARAM, LPARAM)  //  // 目的: 处理主窗口的消息。  //  // WM\_COMMAND - 处理应用程序菜单  // WM\_PAINT - 绘制主窗口  // WM\_DESTROY - 发送退出消息并返回  //  //  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  {  int wmId, wmEvent;  PAINTSTRUCT ps;  HDC hdc;  POINT pts[] = {{0,200},{600,200},{100,600},{300,0},{500,600}};//绘制五角星的顶点  POINT apts[]={{10,70},{50,70},{50,10},{90,10},{90,50},{30,50},{30,90},{70,90},{70,30},{10,30}};  static int cxClient,cyClient;  POINT ampts[10]; //保存放大后的点  switch (message)  {  case WM\_SIZE:  cxClient = LOWORD(lParam);  cyClient = HIWORD(lParam);  break;  case WM\_COMMAND:  wmId = LOWORD(wParam);  wmEvent = HIWORD(wParam);  // 分析菜单选择:  switch (wmId)  {  case IDM\_ABOUT:  DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);  break;  case IDM\_EXIT:  DestroyWindow(hWnd);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  break;  case WM\_PAINT:  HBRUSH hbrush,hOrgBrush;  hdc = BeginPaint(hWnd, &ps);  // TODO: 在此添加任意绘图代码...  hbrush = (HBRUSH)GetStockObject(GRAY\_BRUSH);  hOrgBrush = (HBRUSH)SelectObject(hdc,hbrush);  //ALTERNATE填充模式  SetPolyFillMode(hdc,ALTERNATE);  //Polygon(hdc,pts,5);  //把新多边形的坐标放大  for(int i=0;i<10;i++)  {  ampts[i].x = apts[i].x \* cxClient/200;  ampts[i].y = apts[i].y \* cxClient/200;  }  //使用放大后的坐标绘图-左边  Polygon(hdc,ampts,10);  //更新坐标  for(int i=0;i<10;i++)  {  ampts[i].x += cxClient/2;    }  //设置WINDING填充模式  SetPolyFillMode(hdc,WINDING);  //使用更新后的坐标绘图-右边  Polygon(hdc,ampts,10);  SelectObject(hdc,hOrgBrush);  DeleteObject(hbrush);  EndPaint(hWnd, &ps);  break;  case WM\_DESTROY:  PostQuitMessage(0);  break;  default:  return DefWindowProc(hWnd, message, wParam, lParam);  }  return 0;  }  // “关于”框的消息处理程序。  INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)  {  UNREFERENCED\_PARAMETER(lParam);  switch (message)  {  case WM\_INITDIALOG:  return (INT\_PTR)TRUE;  case WM\_COMMAND:  if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)  {  EndDialog(hDlg, LOWORD(wParam));  return (INT\_PTR)TRUE;  }  break;  }  return (INT\_PTR)FALSE;  } |