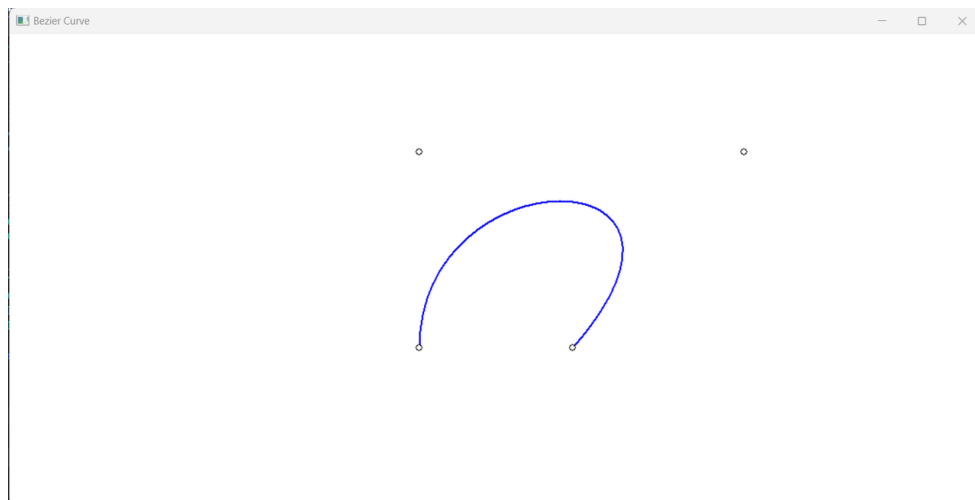


# 程序设计专题报告

**组员：** 3220105728肖思勃 3220105231石宝库 3220103357王周涵















## 摘要

- (1) **选题：** Project#2 Draw Bezier Curves. Use GDI, or implement the rendering by yourself.
- (2) **项目过程：** 初期利用Visual Studio搭建应用程序编译环境，建立文档及鼠标接口；  
中期设计PlotBezier函数，连接文档及鼠标接口；  
后期调参(RGB,控制点的初始位置etc.)，写作实验报告。
- (3) **实现功能：** 从文档init.txt中读取8个整型数据，显示4个控制点与Bezier曲线，  
且控制点可交互，Bezier曲线会动态显示交互后的图像。
- (4) **改进空间：** 可以尝试在程序中显示多条曲线。
- (5) **结果：** 实现了绘制Bezier曲线的程序，且完成了Bonus Task。



## 项目描述

- (1) **项目架构：** 基于Visual Studio应用程序创建模板搭建

 x64	2023/4/21 23:46	文件夹	
 Bezel curve	2023/4/26 17:39	CPP 文件	8 KB
 Bezel curve	2023/4/21 23:45	H 文件	1 KB
 Bezel curve	2022/11/6 17:21	ICO 文件	46 KB
 Bezel curve.rc	2023/4/21 23:45	Resource Script	7 KB
 Bezel curve.sln	2023/4/21 23:45	Visual Studio Sol...	2 KB
 Bezel curve.vcxproj	2023/4/21 23:45	VCXPROJ 文件	7 KB
 Bezel curve.vcxproj.filters	2023/4/21 23:45	VC++ Project Fil...	2 KB
 Bezel curve.vcxproj.user	2023/4/21 23:45	Per-User Project...	1 KB
 framework	2023/4/26 16:05	H 文件	1 KB
 init	2023/4/26 17:53	文本文档	1 KB
 Resource	2023/4/21 23:45	H 文件	1 KB
 small	2022/11/6 17:21	ICO 文件	46 KB
 targetver	2023/4/21 23:45	H 文件	1 KB

## (2) 编译环境：Visual Studio

## (3) 实现过程：

### 1, 头文件

```
SDKDDKVer.h windows.h stdlib.h malloc.h
```

```
memory.h tchar.h resource.h
```

### 2, 变量及函数的前向声明：

```
HINSTANCE hInst;
```

```
WCHAR szTitle[MAX_LOADSTRING];
```

```
WCHAR szWindowClass[MAX_LOADSTRING];
```

程序首先定义了一些全局变量，包括当前实例hInst、标题栏文本szTitle、主窗口类名szWindowClass。

```
ATOM MyRegisterClass(HINSTANCE hInstance);
```

```
BOOL InitInstance(HINSTANCE, int);
```

然后定义了函数MyRegisterClass和InitInstance，它们分别用于注册窗口类和创建窗口。

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

接着定义了窗口消息处理的回调函数WndProc，它处理窗口消息，包括鼠标事件、键盘事件、窗口大小改变事件等。

```
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);
```

最后定义了对话框消息处理函数About，它处理关于对话框的消息。

### 3, 执行应用程序初始化:

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ LPWSTR  
lpCmdLine, _In_ int nCmdShow);
```

```
UNREFERENCED_PARAMETER(hPrevInstance);
```

```
UNREFERENCED_PARAMETER(lpCmdLine);
```

首先调用了 UNREFERENCED\_PARAMETER 宏，用于防止编译器产生未使用参数的警告。

```
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

```
LoadStringW(hInstance, IDC_BEZELCURVE, szWindowClass, MAX_LOADSTRING);
```

```
MyRegisterClass(hInstance);
```

加载了应用程序的标题和窗口类名，并调用 MyRegisterClass 函数注册窗口类。

```
if (!InitInstance (hInstance, nCmdShow))
```

```
{ return FALSE; }
```

调用 InitInstance 函数初始化应用程序实例，并检查初始化是否成功。如果初始化失败，函数返回 FALSE。

```
HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_BEZELCURVE));
```

```
MSG msg;
```

程序加载加速键表，并进入主消息循环。

### 4, 注册窗口: ATOM MyRegisterClass(HINSTANCE hInstance)

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
```

```
WNDCLASSEXW wcex;
```

定义一个WNDCLASSEXW结构体变量，用于存储窗口类的信息。

```
wcex.cbSize = sizeof(WNDCLASSEX); // wcex.cbSize: WNDCLASSEXW结构体的大小，以字节为单位。
```

```
wcex.style = CS_HREDRAW | CS_VREDRAW; // wcex.style: 窗口类的风格，包括CS_HREDRAW和CS_VREDRAW，表示当窗口大小改变时，重绘水平和垂直方向的内容。
```

```
wcex.lpfnWndProc = WndProc; // wcex.lpfnWndproc: 指向窗口过程的指针，用于处理窗口消息。
```

```
wcex.cbClsExtra = 0; // wcex.cbClsExtra: 窗口类的额外类空间大小，以字节为单位。
```

```
wcex.cbWndExtra = 0; // wcex.cbWndExtra: 窗口的额外窗口空间大小，以字节为单位。
```

```
wcex.hInstance = hInstance; // wcex.hInstance: 应用程序实例句柄。
```

```
wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_BEZELCURVE)); // wcex.hIcon: 窗口的图标句柄。
```

```
wcex.hCursor = LoadCursor(nullptr, IDC_ARROW); // wcex.hCursor: 窗口的光标句柄。
```

```
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); // wcex.hbrBackground: 窗口的背景画刷句柄。
```

```
wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_BEZELCURVE); // wcex.lpszMenuName: 窗口的菜单资源名称。
```

```
wcex.lpszClassName = szWindowClass; // wcex.lpszClassName: 窗口类的名称。
```

```
wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL)); // wcex.hIconSm: 窗口的小图标句柄。
```

```
return RegisterClassExW(&wcex); //最后，函数返回一个ATOM类型的值，表示窗口类的原子标识符。
```

## 5, 显示主窗口: `BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)`

```
hInst = hInstance;
```

函数首先将实例句柄存储在全局变量 hInst 中，以便后续使用。

```
HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
```

```
CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
```

接着，函数调用 CreateWindowW 函数创建一个窗口。CreateWindowW 函数的参数依次为窗口类名、窗口标题、窗口样式、窗口位置和大小、父窗口句柄、菜单句柄、实例句柄和附加参数。

```
if (!hWnd) return FALSE;
```

```
ShowWindow(hWnd, nCmdShow);
```

```
UpdateWindow(hWnd);
```

```
return TRUE;
```

如果创建窗口成功，则调用 ShowWindow 函数将窗口显示出来，并调用 UpdateWindow 函数更新窗口。

最后，函数返回 TRUE 表示初始化成功。

## 6, 处理交互消息: `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;
```

```
}
```

首先通过switch语句对消息类型进行判断。如果是WM\_INITDIALOG消息，则返回TRUE，表示对话框已经初始化完成。如果是WM\_COMMAND消息，则判断wParam的值是否为IDOK或IDCANCEL，如果是则调用EndDialog函数关闭对话框，并返回TRUE。如果消息类型不是以上两种，则返回FALSE，表示该消息未被处理。

```
static TCHAR szAppName[] = TEXT("BezierCurve");
```

```
HWND hwnd;
```

```
MSG msg;
```

```
WNDCLASS wndclass;
```

```

wndclass.style = CS_HREDRAW | CS_VREDRAW;

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

```

首先定义了一个静态的TCHAR类型的字符串szAppName，用于指定程序的名称。

接着，定义了一个WNDCLASS结构体，用于描述窗口类的属性，包括样式、窗口过程、额外的类空间和窗口空间、实例句柄、图标、光标、背景刷、菜单和类名。其中，窗口过程是指处理窗口消息的函数，这里使用了一个名为WndProc的函数。

```

if (!RegisterClass(&wndclass))

{

    MessageBox(NULL, TEXT("This program requires Windows NT!"),

    szAppName, MB_ICONERROR);

    return 0;

}

```

然后，调用RegisterClass函数来注册窗口类，如果注册失败，则弹出一个错误提示框并退出程序。

```

hwnd = CreateWindow(szAppName, TEXT("Bezier Curve"),

    WS_OVERLAPPEDWINDOW,

    CW_USEDEFAULT, CW_USEDEFAULT,

    CW_USEDEFAULT, CW_USEDEFAULT,

    NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, iCmdShow);

UpdateWindow(hwnd);

```

接着，使用CreateWindow函数创建一个窗口，并指定窗口的样式、位置和大小、父窗口、菜单和实例句柄等参数。然后，使用ShowWindow和UpdateWindow函数显示和更新窗口。

```
while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}
```

最后，使用GetMessage函数从消息队列中获取消息，并使用TranslateMessage和DispatchMessage函数将消息翻译和分派给窗口过程处理。当消息队列为空时，退出消息循环并返回消息的wParam参数。

7，控制点初始化： **LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)**

```
static int cxClient, cyClient;

static POINT apt[4];

static int iCtrlPt = -1;

FILE* fp;

fopen_s(&fp, "init.txt", "r");

static int x1, y1, x2, y2, x3, y3, x4, y4;

fscanf_s(fp, "%d %d %d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3, &x4, &y4);

fclose(fp);
```

其中

```
cxClient = LOWORD(lParam);

cyClient = HIWORD(lParam);

apt[0].x = x1+cxClient / 3;

apt[0].y = y1+2*cyClient / 3;
```

```

apt[1].x = x2+cxClient / 3;

apt[1].y = y2+cyClient / 4;

apt[2].x = x3+2*cxClient / 3;

apt[2].y = y3+cyClient / 4;

apt[3].x = x4+2*cxClient / 3;

apt[3].y = y4+2*cyClient / 3;

```

打开名为“init.txt”的文件，读取其中的八个整数，分别为四个点的坐标相对于元参数的偏置。这些坐标用于初始化控制点的位置。

#### 8, 绘制Bezier曲线: `LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)`

```

switch (message)
{
case WM_SIZE:

cxClient = LOWORD(lParam);
cyClient = HIWORD(lParam);

apt[0].x = x1+cxClient / 3;

apt[0].y = y1+2*cyClient / 3;

apt[1].x = x2+cxClient / 3;

apt[1].y = y2+cyClient / 4;

apt[2].x = x3+2*cxClient / 3;

apt[2].y = y3+cyClient / 4;

apt[3].x = x4+2*cxClient / 3;

apt[3].y = y4+2*cyClient / 3;

return 0;

```

WM\_SIZE消息处理：当窗口大小改变时，重新计算控制点的位置，使得贝塞尔曲线能够适应新的窗口大小。



```

case WM_PAINT:
{
    HDC hdc;

    PAINTSTRUCT ps;

    HPEN hPen;

    hdc = BeginPaint(hwnd, &ps);

    hPen = CreatePen(PS_SOLID, 2, RGB(0, 0, 255));

    SelectObject(hdc, hPen);

    PolyBezier(hdc, apt, 4);

    DeleteObject(hPen);

    for (int i = 0; i < 4; i++)
    {
        hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));

        SelectObject(hdc, hPen);

        Ellipse(hdc, apt[i].x - 4, apt[i].y - 4, apt[i].x + 4, apt[i].y + 4);

        DeleteObject(hPen);
    }

    EndPaint(hwnd, &ps);

    return 0;
}

```

WM\_PAINT消息处理：绘制贝塞尔曲线和控制点。首先创建一个蓝色的画笔，绘制贝塞尔曲线；然后创建黑色的画笔，绘制椭圆形空心的控制点。

```

case WM_LBUTTONDOWN:
{
    int x = LOWORD(lParam);

    int y = HIWORD(lParam);

    for (int i = 0; i < 4; i++)
    {

```

```

if (x >= apt[i].x - 4 && x <= apt[i].x + 4 && y >= apt[i].y - 4 && y <= apt[i].y + 4)
{
    iCtrlPt = i;
    break;
}
}

return 0;
}

```

WM\_LBUTTONDOWN消息处理：当鼠标左键按下时，判断鼠标是否在控制点的范围内，如果是，则记录下该控制点的索引。

```

case WM_LBUTTONUP:
{
    iCtrlPt = -1;
    return 0;
}

```

WM\_LBUTTONUP消息处理：当鼠标左键松开时，清除控制点的索引。

```

case WM_MOUSEMOVE:
{
    if (iCtrlPt != -1)
    {
        apt[iCtrlPt].x = LOWORD(IParam);
        apt[iCtrlPt].y = HIWORD(IParam);
        InvalidateRect(hwnd, NULL, TRUE);
    }
    return 0;
}

```

WM\_MOUSEMOVE消息处理：当鼠标移动时，如果有控制点被选中，则更新该控制点的位置，并重新绘制贝塞尔曲线和控制点。

```
case WM_DESTROY:
```

```
PostQuitMessage(0);
```

```
return 0;
```

```
}
```

WM\_DESTROY消息处理：当窗口被销毁时，发送退出消息，结束程序运行。

```
return DefWindowProc(hwnd, message, wParam, lParam);
```

#### (4) 优化：

1. 该程序可能重复创建了窗口（同时使用了WinMain函数和wWinMain函数）。因为该程序通过Visual Studio的创建应用程序框架搭建，主程序（.sln）前185行是自动生成的，此后是本小组的code，但是修改后程序无法运行（可能是没有正确修改），程序可以正常运行，因此不做修改，但是在此还是指出这个问题。
2. 该程序的核心组件:Window.h中的PolyBezier函数：对该函数的调用，直接减少了笔刷轨迹的拟合计算工作量，进而减少了代码量。
3. 该程序可以在窗口中创建多条曲线，作为重复性的代码，没有必要性，因此该程序不做绘制。

#### (5) 关键结果&收获：

##### 1. 建立及连接鼠标交互接口

```
UNIT message
```

```
switch(message)
```

```
case WM_LBUTTONDOWN
```

```
case WM_LBUTTONUP
```

```
case WM_MOUSEMOVE
```

##### 2. 控制点的设计

使用Ellipse函数绘制椭圆形区域

```
for (int i = 0; i < 4; i++)
{
    hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
    SelectObject(hdc, hPen);
    Ellipse(hdc, apt[i].x - 4, apt[i].y - 4, apt[i].x + 4, apt[i].y + 4);
    DeleteObject(hPen);
}
```

交互区是椭圆长轴\*短轴的矩形区域

```
for (int i = 0; i < 4; i++)
{
    if (x >= apt[i].x - 4 && x <= apt[i].x + 4 && y >= apt[i].y - 4 && y <= apt[i].y + 4)
    {
        iCtrlPt = i;
        break;
    }
}
```

### 3. 变量前向声明

例如,

```
HINSTANCE hInst;
```

```
WCHAR szTitle[MAX_LOADSTRING];
```

```
WCHAR szWindowClass[MAX_LOADSTRING];
```

程序定义了一些全局变量, 包括当前实例hInst、标题栏文本szTitle、主窗口类名szWindowClass。

.....

程序中存在较多特殊数据类型, 其中一部分是编译器自动生成的, 这部分理解和使用非常困难, 通过编译器debug窗口的提示以及通过对相关案例的模仿, 实现了程序的运行。

### 4. 回调函数的使用

例如,

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

窗口消息处理函数WndProc, 它处理窗口消息, 包括鼠标事件、键盘事件、窗口大小改变事件等。

```
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);
```

对话框消息处理函数About, 它处理关于对话框的消息。

### 5. Debug

在调试的过程中, 程序出现了拖动控制点即崩溃闪退的情况,

最终发现忘记添加fclose (fp) ,

可能的原因是在窗口消息循环会不断的积累空指针, 造成程序崩溃。