```cpp
// WellPerturbation.cpp (c) Tuesday, October 29, 2024
// by James Pate Williams, Jr., BA BS, MSWE, PhD
// https://dabm.stanford.edu/wp-content/uploads/2021/12/Lecture_23.pdf

#include "framework.h"
#include "WellPerturbation.h"
#include <float.h>
#include <math.h>
#include <stdio.h>

#define MAX_LOADSTRING 100
#define nPts 2048

typedef struct tagPoint2d
{
    double x, y;
} POINT2D, * PPOINT2D;

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name
double f;                                       // field constant
int function;                                   // 0 = 1s Order, 1 = 2nd Order
int u, v;                                       // sine function parameters
POINT2D pts[nPts + 1];                          // two-dimensional double points

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    GetData(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    Draw(HWND, UINT, WPARAM, LPARAM);

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

    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_WELLPERTURBATION, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
```

```
53      {
54          return FALSE;
55      }
56
57      HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE
          (IDC_WELLPERTURBATION));
58
59      MSG msg;
60
61      // Main message loop:
62      while (GetMessage(&msg, nullptr, 0, 0))
63      {
64          if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
65          {
66              TranslateMessage(&msg);
67              DispatchMessage(&msg);
68          }
69      }
70
71      return (int) msg.wParam;
72  }
73  //
74  //  FUNCTION: MyRegisterClass()
75  //
76  //  PURPOSE: Registers the window class.
77  //
78  ATOM MyRegisterClass(HINSTANCE hInstance)
79  {
80      WNDCLASSEXW wcex;
81
82      wcex.cbSize = sizeof(WNDCLASSEX);
83
84      wcex.style          = CS_HREDRAW | CS_VREDRAW;
85      wcex.lpfnWndProc    = WndProc;
86      wcex.cbClsExtra     = 0;
87      wcex.cbWndExtra     = 0;
88      wcex.hInstance      = hInstance;
89      wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE
          (IDI_WELLPERTURBATION));
90      wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
91      wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
92      wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_WELLPERTURBATION);
93      wcex.lpszClassName  = szWindowClass;
94      wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
95
96      return RegisterClassExW(&wcex);
97  }
98  //
99  //   FUNCTION: InitInstance(HINSTANCE, int)
100 //
101 //   PURPOSE: Saves instance handle and creates main window
102 //
```

```
103 //   COMMENTS:
104 //
105 //        In this function, we save the instance handle in a global variable and
106 //        create and display the main program window.
107 //
108 BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
109 {
110    hInst = hInstance; // Store instance handle in our global variable
111
112    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
113       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
114
115    if (!hWnd)
116    {
117       return FALSE;
118    }
119
120    ShowWindow(hWnd, nCmdShow);
121    UpdateWindow(hWnd);
122
123    return TRUE;
124 }
125 //
126 //  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
127 //
128 //  PURPOSE: Processes messages for the main window.
129 //
130 //  WM_COMMAND  - process the application menu
131 //  WM_PAINT    - Paint the main window
132 //  WM_DESTROY  - post a quit message and return
133 //
134 //
135 LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
136 {
137     switch (message)
138     {
139     case WM_COMMAND:
140         {
141             int wmId = LOWORD(wParam);
142             // Parse the menu selections:
143             switch (wmId)
144             {
145             case IDM_ABOUT:
146                 DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
147                 break;
148             case ID_FILE_1STORDER:
149                 function = 1;
150                 MessageBoxA(hWnd, "No 1st order energy correction",
151                     "Information", MB_OK | MB_ICONINFORMATION);
152                 break;
153             case ID_FILE_2NDORDER:
154                 function = 2;
```

```
155                 DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, GetData);
156                 DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG2), hWnd, Draw);
157                 break;
158             case IDM_EXIT:
159                 DestroyWindow(hWnd);
160                 break;
161             default:
162                 return DefWindowProc(hWnd, message, wParam, lParam);
163             }
164         }
165         break;
166     case WM_PAINT:
167         {
168             PAINTSTRUCT ps;
169             HDC hdc = BeginPaint(hWnd, &ps);
170             // TODO: Add any drawing code that uses hdc here...
171             EndPaint(hWnd, &ps);
172         }
173         break;
174     case WM_DESTROY:
175         PostQuitMessage(0);
176         break;
177     default:
178         return DefWindowProc(hWnd, message, wParam, lParam);
179     }
180     return 0;
181 }
182 // Message handler for about box.
183 INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
184 {
185     UNREFERENCED_PARAMETER(lParam);
186     switch (message)
187     {
188     case WM_INITDIALOG:
189         return (INT_PTR)TRUE;
190 
191     case WM_COMMAND:
192         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
193         {
194             EndDialog(hDlg, LOWORD(wParam));
195             return (INT_PTR)TRUE;
196         }
197         break;
198     }
199     return (INT_PTR)FALSE;
200 }
201 
202 double SecondOrderIntegrand(double xi) {
203     double pi = 4.0 * atan(1.0);
204     double su = sin(u * pi * xi);
205     double sv = sin(v * pi * xi);
206     return 2.0 * f * (xi - 0.5) * su * sv;
```

```
207 }
208
209 double SimpsonsRule(double a, double b, int n,
210     double (*f)(double))
211 {
212     double h = (b - a) / n;
213     double h2 = 2.0 * h;
214     double s = 0.0;
215     double t = 0.0;
216     double x = a + h;
217
218     for (int i = 1; i < n; i += 2)
219     {
220         s += f(x);
221         x += h2;
222     }
223
224     x = a + h2;
225
226     for (int i = 2; i < n; i += 2)
227     {
228         t += f(x);
229         x += h2;
230     }
231
232     return h * (f(a) + 4 * s + 2 * t + f(b)) / 3.0;
233 }
234
235 INT_PTR CALLBACK GetData(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
236 {
237     UNREFERENCED_PARAMETER(lParam);
238     char line[128] = { };
239     double dx = 0, sr = 0, xi = 0;
240     switch (message)
241     {
242     case WM_INITDIALOG:
243         return (INT_PTR)TRUE;
244
245     case WM_COMMAND:
246         if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
247         {
248             f = GetDlgItemInt(hDlg, IDC_EDIT1, FALSE, FALSE);
249             u = GetDlgItemInt(hDlg, IDC_EDIT2, FALSE, FALSE);
250             v = GetDlgItemInt(hDlg, IDC_EDIT3, FALSE, FALSE);
251             dx = 1.0 / nPts;
252
253             for (int i = 0; i < nPts; i++)
254             {
255                 pts[i].x = xi;
256                 pts[i].y = SecondOrderIntegrand(xi);
257                 xi += dx;
258             }
```

```
259
260             sr = SimpsonsRule(0, 1, 1024,
261                 SecondOrderIntegrand);
262             sprintf_s(line, "%lf", sr);
263             MessageBoxA(hDlg, line, "2nd Order Matrix Element",
264                 MB_OK | MB_ICONINFORMATION);
265             EndDialog(hDlg, LOWORD(wParam));
266             return (INT_PTR)TRUE;
267         }
268         if (LOWORD(wParam) == IDCANCEL)
269         {
270             EndDialog(hDlg, LOWORD(wParam));
271             return (INT_PTR)TRUE;
272         }
273         break;
274     }
275     return (INT_PTR)FALSE;
276 }
277
278 void FindMinMax(
279     double& xMin, double& xMax,
280     double& yMin, double& yMax)
281 {
282     // uses global 2D double points structure
283
284     xMin = yMin = DBL_MAX;
285     xMax = yMax = DBL_MIN;
286
287     for (int i = 0; i < nPts; i++)
288     {
289         POINT2D pt = pts[i];
290         double x = pt.x;
291         double y = pt.y;
292
293         if (x < xMin)
294             xMin = x;
295         if (x > xMax)
296             xMax = x;
297         if (y < yMin)
298             yMin = y;
299         if (y > yMax)
300             yMax = y;
301     }
302 }
303
304 void DrawFormattedText(HDC hdc, char text[], RECT rect)
305 {
306     // Draw the text with formatting options
307     DrawTextA(hdc, text, -1, &rect, DT_SINGLELINE | DT_NOCLIP);
308 }
309
310 INT_PTR CALLBACK Draw(
```

```
311      HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
312  {
313      // draws an associated Legendre function
314      UNREFERENCED_PARAMETER(lParam);
315      switch (message)
316      {
317      case WM_INITDIALOG:
318          if (function == 1)
319              SetWindowTextA(hDlg, "1st Order Perturbation Matrix Element
                   Integrand");
320          if (function == 2)
321              SetWindowTextA(hDlg, "2nd Order Perturbation Matrix Element
                   Integrand");
322          return (INT_PTR)TRUE;
323      case WM_COMMAND:
324          if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
325          {
326              EndDialog(hDlg, LOWORD(wParam));
327              return (INT_PTR)TRUE;
328          }
329          break;
330      case WM_PAINT:
331      {
332          double h = 0, pi = 0, plm = 0, theta = 0;
333          double xMax = 0, xMin = 0, yMax = 0, yMin = 0;
334          FindMinMax(xMin, xMax, yMin, yMax);
335          float xSpan = (float)(xMax - xMin);
336          float ySpan = (float)(yMax - yMin);
337          RECT rect = { };
338          GetClientRect(hDlg, &rect);
339          float width = (float)(rect.right - rect.left + 1);
340          float height = (float)(rect.bottom - rect.top - 32 + 1);
341          float sx0 = 2.0f * width / 16.0f;
342          float sx1 = 14.0f * width / 16.0f;
343          float sy0 = 2.0f * height / 16.0f;
344          float sy1 = 14.0f * height / 16.0f;
345          float deltaX = xSpan / 8.0f;
346          float deltaY = ySpan / 8.0f;
347          float xSlope = (sx1 - sx0) / xSpan;
348          float xInter = (float)(sx0 - xSlope * xMin);
349          float ySlope = (sy0 - sy1) / ySpan;
350          float yInter = (float)(sy0 - ySlope * yMax);
351          float px = 0, py = 0, sx = 0, sy = 0;
352          PAINTSTRUCT ps;
353          POINT wPt = { };
354          HDC hdc = BeginPaint(hDlg, &ps);
355          int i = 0;
356          float x = (float)xMin;
357          float y = (float)yMax;
358          px = x;
359          py = y;
360          sx = xSlope * px + xInter;
```

```
361         sy = ySlope * py + yInter;
362         MoveToEx(hdc, (int)sx, (int)sy0, &wPt);
363         char buffer[128] = { };
364
365         while (i <= 8)
366         {
367             sx = xSlope * x + xInter;
368             MoveToEx(hdc, (int)sx, (int)sy0, &wPt);
369             LineTo(hdc, (int)sx, (int)sy1);
370             sprintf_s(buffer, "%5.2f", x);
371             SIZE size = { };
372             GetTextExtentPoint32A(
373                 hdc,
374                 buffer,
375                 strlen(buffer),
376                 &size);
377             RECT textRect = { };
378             textRect.left = (long)(sx - size.cx / 2.0f);
379             textRect.right = (long)(sx + size.cx / 2.0f);
380             textRect.top = (long)sy1;
381             textRect.bottom = (long)(sy1 + size.cy / 2.0f);
382             DrawFormattedText(hdc, buffer, textRect);
383             x += deltaX;
384             i++;
385         }
386
387         i = 0;
388         y = (float)yMin;
389
390         while (i <= 8)
391         {
392             sy = ySlope * y + yInter;
393             MoveToEx(hdc, (int)sx0, (int)sy, &wPt);
394             LineTo(hdc, (int)sx, (int)sy);
395
396             if (i != 0)
397             {
398                 sprintf_s(buffer, "%5.2f", y);
399                 SIZE size = { };
400                 GetTextExtentPoint32A(
401                     hdc,
402                     buffer,
403                     strlen(buffer),
404                     &size);
405                 RECT textRect = { };
406                 textRect.left = (long)(sx0 - size.cx - size.cx / 5.0f);
407                 textRect.right = (long)(sx0 - size.cx / 2.0f);
408                 textRect.top = (long)(sy - size.cy / 2.0f);
409                 textRect.bottom = (long)(sy + size.cy / 2.0f);
410                 DrawFormattedText(hdc, buffer, textRect);
411             }
412
```

```
413             y += deltaY;
414             i++;
415         }
416
417         px = (float)pts[0].x;
418         py = (float)pts[0].y;
419         sx = xSlope * px + xInter;
420         sy = ySlope * py + yInter;
421         MoveToEx(hdc, (int)sx, (int)sy, &wPt);
422
423         HGDIOBJ rPenNew = NULL;
424         HGDIOBJ hPenOld = NULL;
425         rPenNew = CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
426         hPenOld = SelectObject(hdc, rPenNew);
427
428         for (int j = 0; j < nPts; j++)
429         {
430             px = (float)pts[j].x;
431             py = (float)pts[j].y;
432             sx = xSlope * px + xInter;
433             sy = ySlope * py + yInter;
434             LineTo(hdc, (int)sx, (int)sy);
435         }
436
437         SelectObject(hdc, hPenOld);
438         DeleteObject(rPenNew);
439         EndPaint(hDlg, &ps);
440         return (INT_PTR)FALSE;
441         break;
442     }
443     }
444     return (INT_PTR)FALSE;
445 }
```