# Compiling AGG under Microsoft eMbedded VC 4.0

## A simple step-by-step tutorial

If you have a Microsoft eMbedded Visual C++ 4.0 (eVC4.0) installed you can develop applications based on **AGG** for PocketPCs that use Windows CE.

I didn't have any problems compiling **AGG** under eVC4.0, it looks pretty much the same as building of Win32 API applications.

Below there is a spep-by-step instruction how to create a simple WinCE application with **AGG**.

1. Create a new `WCE Application`, for example in `agg2/examples/win32_ce/agg_test`.
   Choose **"A typical Hello World Application"**.
2. Add path to the **AGG** include directory:
   **Project/Settings**, set **"Setting For:"** to **"All configurations"**.
   Select tab **"C/C++"**, category **"Preprocessor"**.
   Type path to the **AGG** include directory in the **"Additional Include Directories"**.
   If you put the project to `agg2/examples/win32_ce/agg_test`, type `../../../include`
3. Add necessary **AGG** source files to the project from `agg2/src`. You can actually add all of them (except the ones in the sub-directories).
4. Select all the **AGG** source files in the project (`agg_*.cpp`, except `agg_test.*`).
5. Go to **Project/Settings**, set **"Setting For:"** to **"All configurations"**.
   Select tab **"C/C++"**, category **"Precompiled Headers"**.
   Choose option **"Not using precompiled headers"**.
6. Replace the content of `agg_test.cpp` to the following (see the source in agg_test.cpp).
   Basically you need to replace the `WM_PAINT` event handler, add the **AGG** includes, and add the `agg_draw()` function.

```
// agg_test.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "agg_test.h"
#include <commctrl.h>

#include "agg_rendering_buffer.h"
#include "agg_curves.h"
#include "agg_conv_stroke.h"
#include "agg_rasterizer_scanline_aa.h"
#include "agg_scanline_p.h"
```

```
#include "agg_renderer_scanline.h"
#include "agg_pixfmt_rgb555.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE           hInst;          // The current instance
HWND                hwndCB;         // The command bar handle

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

int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow)
{
    MSG msg;
    HACCEL hAccelTable;

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_AGG_TEST);

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    It is important to call this function so that the application
//    will get 'well formed' small icons associated with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance, LPTSTR szWindowClass)
{
    WNDCLASS    wc;

    wc.style            = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc      = (WNDPROC) WndProc;
    wc.cbClsExtra       = 0;
```

```
    wc.cbWndExtra       = 0;
    wc.hInstance        = hInstance;
    wc.hIcon            = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_AGG_TEST));
    wc.hCursor          = 0;
    wc.hbrBackground    = (HBRUSH) GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName     = 0;
    wc.lpszClassName    = szWindowClass;

    return RegisterClass(&wc);
}

//
//  FUNCTION: InitInstance(HANDLE, int)
//
//  PURPOSE: Saves instance handle and creates main window
//
//  COMMENTS:
//
//    In this function, we save the instance handle in a global variable and
//    create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND    hWnd;
    TCHAR   szTitle[MAX_LOADSTRING];            // The title bar text
    TCHAR   szWindowClass[MAX_LOADSTRING];      // The window class name

    hInst = hInstance;      // Store instance handle in our global variable
    // Initialize global strings
    LoadString(hInstance, IDC_AGG_TEST, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance, szWindowClass);

    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    hWnd = CreateWindow(szWindowClass, szTitle, WS_VISIBLE,
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, NULL, NULL, hInstance, NULL);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    if (hwndCB)
        CommandBar_Show(hwndCB, TRUE);

    return TRUE;
}




void agg_draw(unsigned char* buf, unsigned w, unsigned h, int stride)
{
    typedef agg::pixfmt_rgb555 pixfmt;

    //============================================================
    // AGG lowest level code.
    agg::rendering_buffer rbuf;
    rbuf.attach((unsigned char*)buf, w, h, stride);
```

```
    // Pixel format and basic primitives renderer
    pixfmt pixf(rbuf);
    agg::renderer_base<pixfmt> renb(pixf);

    renb.clear(agg::rgba8(255, 255, 255, 255));

    // Scanline renderer for solid filling.
    agg::renderer_scanline_aa_solid<agg::renderer_base<pixfmt> > ren(renb);

    // Rasterizer & scanline
    agg::rasterizer_scanline_aa<> ras;
    agg::scanline_p8 sl;

    agg::curve4 curve;
    agg::conv_stroke<agg::curve4> poly(curve);
    unsigned i;

    srand(12365);

    for(i = 0; i < 100; i++)
    {
        poly.width(double(rand() % 3500 + 500) / 500.0);

        curve.init(rand() % w, rand() % h,
                   rand() % w, rand() % h,
                   rand() % w, rand() % h,
                   rand() % w, rand() % h);

        ren.color(agg::rgba8(rand() & 0xFF,
                             rand() & 0xFF,
                             rand() & 0xFF,
                             rand() & 0xFF));

        ras.add_path(poly, 0);
        agg::render_scanlines(ras, sl, ren);
    }

    //============================================================
}





//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    int wmId, wmEvent;
    PAINTSTRUCT ps;
```

```
//  TCHAR szHello[MAX_LOADSTRING];

    switch (message)
    {
        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_HELP_ABOUT:
                   DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                   break;
                case IDM_FILE_EXIT:
                   DestroyWindow(hWnd);
                   break;
                default:
                   return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_CREATE:
            hwndCB = CommandBar_Create(hInst, hWnd, 1);
            CommandBar_InsertMenubar(hwndCB, hInst, IDM_MENU, 0);
            CommandBar_AddAdornments(hwndCB, 0, 0);
            break;

        case WM_PAINT:
            {
                hdc = BeginPaint(hWnd, &ps);
                RECT rt;
                GetClientRect(hWnd, &rt);

                int width = rt.right - rt.left;
                int height = rt.bottom - rt.top;

                //============================================================
                //Creating compatible DC and a bitmap to render the image
                BITMAPINFO bmp_info;
                bmp_info.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
                bmp_info.bmiHeader.biWidth = width;
                bmp_info.bmiHeader.biHeight = height;
                bmp_info.bmiHeader.biPlanes = 1;
                bmp_info.bmiHeader.biBitCount = 16;
                bmp_info.bmiHeader.biCompression = BI_RGB;
                bmp_info.bmiHeader.biSizeImage = 0;
                bmp_info.bmiHeader.biXPelsPerMeter = 0;
                bmp_info.bmiHeader.biYPelsPerMeter = 0;
                bmp_info.bmiHeader.biClrUsed = 0;
                bmp_info.bmiHeader.biClrImportant = 0;

                HDC mem_dc = ::CreateCompatibleDC(hdc);

                void* buf = 0;

                HBITMAP bmp = ::CreateDIBSection(
                    mem_dc,
                    &bmp_info,
                    DIB_RGB_COLORS,
                    &buf,
                    0,
                    0
```

```
                );

                // Selecting the object before doing anything allows you
                // to use AGG together with native Windows GDI.
                HBITMAP temp = (HBITMAP)::SelectObject(mem_dc, bmp);

                // Calculate the aligned stride value for the 16-bit BMP.
                int stride = ((width * 2 + 3) >> 2) << 2;

                // Negate the stride value to have the Y-axis flipped
                agg_draw((unsigned char*)buf, width, height, -stride);

                //-----------------------------------------------------------
                // Display the image. If the image is B-G-R-A (32-bits per pixel)
                // one can use AlphaBlend instead of BitBlt. In case of AlphaBlend
                // one also should clear the image with zero alpha, i.e. rgba8(0,0,0,0)

                ::BitBlt(
                  hdc,
                  rt.left,
                  rt.top,
                  width,
                  height,
                  mem_dc,
                  0,
                  0,
                  SRCCOPY
                );

                // Free resources
                ::SelectObject(mem_dc, temp);
                ::DeleteObject(bmp);
                ::DeleteObject(mem_dc);

                EndPaint(hWnd, &ps);
            }
            break;

        case WM_DESTROY:
            CommandBar_Destroy(hwndCB);
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Mesage handler for the About box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    RECT rt, rt1;
    int DlgWidth, DlgHeight;    // dialog width and height in pixel units
    int NewPosX, NewPosY;

    switch (message)
    {
        case WM_INITDIALOG:
            // trying to center the About dialog
            if (GetWindowRect(hDlg, &rt1)) {
                GetClientRect(GetParent(hDlg), &rt);
```

```
                DlgWidth    = rt1.right - rt1.left;
                DlgHeight   = rt1.bottom - rt1.top ;
                NewPosX     = (rt.right - rt.left - DlgWidth)/2;
                NewPosY     = (rt.bottom - rt.top - DlgHeight)/2;

                // if the About box is larger than the physical screen
                if (NewPosX < 0) NewPosX = 0;
                if (NewPosY < 0) NewPosY = 0;
                SetWindowPos(hDlg, 0, NewPosX, NewPosY,
                    0, 0, SWP_NOZORDER | SWP_NOSIZE);
            }
            return TRUE;

        case WM_COMMAND:
            if ((LOWORD(wParam) == IDOK) || (LOWORD(wParam) == IDCANCEL))
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;
    }
    return FALSE;
}
```

Download the whole project: (agg_test.zip). It's supposed to be placed in `agg2/examples/win32_ce/agg_test/`. Create `win32_ce` and `agg_test` directories. If you choose another directory, modify the **Additional Include Directories** path, and you will have to remove all the **AGG** source files from the project and add new ones from the actual **AGG** directory (don't forget to turn **Precompiled Headers** off).

**NOTE**
I'm not a pro in WinCE and embedded systems, but I presumed that there's an RGB555 pixel format is used. If it's not so, use another pixel format that fits the native display format.

Change function `agg_draw()` and try other **AGG** stuff. Also note, that the whole scene is being drawn from scratch every time the `WM_PAINT` event comes. It's a good idea to cache the rendered image and just `BitBlt()` it if there were no changes.

Here's a screenshot (click to see the enlarged version)