InterGrafx
Moving graphics.

InterGrafx V4 引擎簡介

Ver 1.02, 2011/4/19

目錄

1. V4 引擎介紹 4

2. V4 引擎的軟體架構層 5

3. V4 程式設計概述 7

4. 數值計算問題 21

4.1 浮點數與定點數 21

4.2 精確度 22

檔修訂紀錄

2006/3/19	0.95 版文件	初版 (charles)
2011/4/19	1.03 版	

1. V4 引擎介紹

InterGrafx V4 引擎是 InterGrafx 第四代行動繪圖核心技術。它基於代號 IGV3 的 InterGrafx Proprietary 3D Engine 第三代為軟體渲染核心，並且增加針對具有 OpenGL ES 1.x / 2.0 硬體的支援。您甚至可以將 V4 視為是一個針對先進行動裝置而設計的「全新產品」，而不僅僅是 V3 的擴充改版。

也正是因為這樣，所以 V4 引擎是根據行動 3D 繪圖規格 JSR184 為基礎的全新設計，和 V3 引擎並沒有完全向下相容。過去基於 InterGrafx V3 引擎所開發的產品，需要修改才可移植到 V4 引擎上。使用者必須根據產品的需求重新撰寫程式。但是這樣的投資通常是值得的，因為：

- V4 引擎提供了過去 V3 引擎所沒有的許多新功能。要使用這些新的功能，原本就必須要使用新的應用程式介面。
- V4 引擎主要針對較高階的行動裝置。這些裝置所使用的應用軟體，通常比針對 V3 引擎所適用的中低階行動裝置所設計的軟體功能需要更高。所以直接把功能簡易的 V3 遊戲移植到 V4 引擎所支援的高階行動裝置，其實意義不大。

儘管如此，對於有使用 V3 引擎經驗的程式師，仍會發現 V4 引擎有許多與 V3 引擎共通的觀念，以及額外擴充的功能。

2. V4 引擎的軟體架構層

V4 引擎設計的架構分為兩個主要的軟體架構層(layer)

除了作業系統的底層之外，V4 引擎主要包括繪圖渲染底層與 M3G 3D 應用程式介面層兩個主要的軟體架構層。

2.1 M3G 3D 應用程式介面層

M3G 應用程式介面層提供業界標準的高階的 3D 繪圖 API。主要包括下列功能：

- 在有限的硬體環境下提供常用的 3D 繪圖 API

行動裝置對於 3D API 的需求與一般 PC、有 3D 圖形處理能力的電視遊樂器等不同。行動裝置使用極為有限的記憶體，CPU 速度與 PC 比起來慢很多，很多時候更缺乏 3D 運算大量使用的 FPU。這些都導致行動裝置的 API 在制定上並不是以提供一套完整的 3D API，而是提供一套適用於有限裝置的 3D API。M3G API 的設計就是基於這樣的理念。

- 提供業界標準的資源檔案格式

目前市面上許多 3D API 都使用自己的特殊資源檔案格式，因此除了程式師需要熟悉這些 API，而美術與製作人員更常會苦於沒有熟悉的 3D 軟體可以使用。M3G 使用公開的資源檔案格式，主流 3D 編輯軟體如 3DS Max 都是直接就支援本格式。

- 支援多種 Profile

行動裝置的運算能力從 1Ghz 雙核處理器到 200Mhz 沒有 FPU 的處理器。以這樣廣泛的運算能力，並不容易制定一套 API 可以滿足各種極端的情況。M3G 使用區分不同的硬體 Profile 來處理這種問題。IGV4 實作的 M3G

版本支援 Common 與 Common lite 兩種 Profile。

2.2 繪圖渲染底層

InterGrafx V4 引擎內建針對目標平台客製化的渲染引擎，它可以支援：

- IGV3 軟體 3D 渲染底層

這是 InterGrafx 開發能在相對低階的手機平台如 300Mhz CPU 上發揮最大效能的 3D 純軟體渲染底層。在比較局限的平台上 (CPU 較慢而需要渲染大屏幕) ，建議使用 IGV3 可以得到最佳的效能要求。IGV3 有如 IGV4 的精簡版 API。許多 MediaTek 6236/6276 的遊戲，就是直些使用 IGV3 以達到最佳的渲染效能。

- IGES OpenGL ES 1.x 軟體渲染底層

這是 InterGrafx 開發之標準 OpenGL ES 1.x 軟體渲染底層。它相較於 IGV3 提供更多的渲染功能，但是適合於 CPU 較快的平台。在速度較慢的平台上，它的渲染效能會影響如遊戲等需要及時高速的運作。

- 硬體加速 OpenGL ES 1.x 底層 (EGL+ES API)

IGV4 使用硬體 OpenGL ES 1.x 時能提供最好的品質與效能。iPhone / iPad / Android 版本的 IGV4 可以直接使用平台的 ES 1.x 做為渲染底層。

- 硬體加速 OpenGL ES 2.0 底層 (EGL+ES API)

IGV4 使用硬體 OpenGL ES 2.x 時能提供最好的品質與效能。不過 IGV4 第二代 (IGV4.2) 版才支援 OpenGL ES 2.x 的 Shader 功能。IGV4.2 預計於 MediaTek 6255 平台開始提供。

- 第三方實作之 OpenGL ES 1.x 軟體渲染底層

IGV4 也可以轉接其他第三方開發之軟體 OpenGL ES 1.x 渲染底層。不

過 InterGrafx 並不保證第三方 OpenGL ES 1.x 實作具備所有必要的功能，也無法保證第三方 OpenGL ES 1.x 實作的穩定性。

3. V4 程式設計概述

以下是一個 V4 引擎的簡易範例程式。它是基於 Win32 版本的 V4 引擎所開發的，以下假設讀者有使用 SDK 開發 Win32 程式的經驗。其中標示為紅字的部分為 V4 引擎相關的程式碼。

```
// v4Sample.cpp : Defines the entry point for the application.
//
#ifdef WIN32
#include <math.h>
#endif

#include "stdafx.h"
#include "resource.h"
#include "windows.h"
```

以下是 V4 引擎所使用的一些含入檔。在這個範例裡用到了這幾個檔案

```
#include "IGM3GGraphics3D.h"
#include "IGM3GEngine.h"
#include "IGM3GLoader.h"
#include "IGSmartObjArray.h"
```

以下為了說明方便，把一些 Win32 常用的變數宣告成全域變數，因為

它們會被許多程式使用到。

```
#define MAX_LOADSTRING 100

// Global Variables:
HWND g_hWnd;
HINSTANCE hInst;                                    // current instance
// The title bar text
TCHAR szTitle[MAX_LOADSTRING];
TCHAR szWindowClass[MAX_LOADSTRING];    // The title bar text
```

以下為了說明方便，把 V4 引擎所使用的一些變數宣告成全域變數，因為它們會被許多程式參考到。

```
HBITMAP g_hBitmap;          // handle of bitmap for engine render
int g_iWidth = 256;         // bitmap width
int g_iHeight = 256;        // bitmap height
IGMemory* g_pMem = NULL;
IGM3GGraphics3D* g_pG3D = NULL;
IGM3GWorld* g_pWorld = NULL;
int g_Timer = 0;
```

以下是一些函數的提前宣告，因為我們會在定義它之前就先用到它們：

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

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

以下函數建立一個 V4 引擎的畫布。我們使用標準 Win32 的視窗 Bitmap 做為畫布：

```
/*---------------------------------------------------------------------*//*!
 * \brief      Create a DIB section bitmap.
 * \param      hDC DC handle.
 * \return     Returns the handle to the created bitmap.
 *//*---------------------------------------------------------------------*/
#define SAMPLE_PIXMAP_BITS 32

static HBITMAP es184CreatePixmap (HDC hDC, int width, int height, void **bits)
{
      const size_t       bmiSize = sizeof(BITMAPINFO) + 256U*sizeof(RGBQUAD);
      BITMAPINFO*        bmi;
      HBITMAP hBitmap = NULL;

      bmi = (BITMAPINFO*)malloc(bmiSize);
      memset(bmi, 0, bmiSize);

      bmi->bmiHeader.biSize                = sizeof(BITMAPINFOHEADER);
      bmi->bmiHeader.biWidth               = width;
      bmi->bmiHeader.biHeight              = -height;
      bmi->bmiHeader.biPlanes              = (short)1;
      bmi->bmiHeader.biBitCount            = (unsigned
short)SAMPLE_PIXMAP_BITS;
      bmi->bmiHeader.biCompression         = BI_RGB;
      bmi->bmiHeader.biSizeImage           = 0;
```

```
    bmi->bmiHeader.biXPelsPerMeter    = 0;
    bmi->bmiHeader.biYPelsPerMeter    = 0;
    bmi->bmiHeader.biClrUsed            = 0;//3;
    bmi->bmiHeader.biClrImportant     = 0;

    hBitmap = CreateDIBSection(hDC, bmi, DIB_RGB_COLORS, bits, NULL, 0);
    free(bmi);
    return hBitmap;
}
```

以下是 Win32 的入口程式。我們在這裡除了起始標準 Win32 的應用程式之外，也起始我們的 V4 引擎。

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    MSG msg;
    HACCEL hAccelTable;

    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_V4SAMPLE, szWindowClass,
MAX_LOADSTRING);
    MyRegisterClass(hInstance);

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

```
{
    return FALSE;
}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_V4SAMPLE);

// allocate memory for v4 engine
g_pMem = (IGMemory *) malloc(1024*1024);
IGMemory_Init(g_pMem, 1024*1024);

// initial memory pool for M3G
igm3gEngineInit(g_pMem);

// new Graphic3D for render
g_pG3D = New(IGM3GGraphics3D, g_pMem);

// create render target
void *bits;
IGPixmap pixmapSur;

HDC hDC = GetDC(g_hWnd);
g_hBitmap = es184CreatePixmap(hDC, g_iWidth, g_iHeight, &bits);
pixmapSur.format = IG_PIXMAPFORMAT_8888;
pixmapSur.width = g_iWidth;
pixmapSur.height = -g_iHeight;
pixmapSur.pixels = bits;
pixmapSur.pixelBytes = SAMPLE_PIXMAP_BITS/8;
pixmapSur.lineOffsetBytes = g_iWidth*pixmapSur.pixelBytes;
// Ensure bmp_stride is DWORD aligned
```

```
while((pixmapSur.lineOffsetBytes % 4) != 0)
    pixmapSur.lineOffsetBytes++;

ReleaseDC(g_hWnd, hDC);

// bind render target
IGM3GGraphics3D_BindTarget(g_pMem, g_pG3D, &pixmapSur);

// new Loader
IGM3GLoader *pLoader = NULL;
pLoader = New(IGM3GLoader, g_pMem);

//載入一個事先製作好的 V4 content
char filename[] = "otokka_jump2.m3g";
IGSmartObjArray* pObjArray;
IGErr err = IGERR_NONE;
IGM3GErr err2 = IGM3GERR_NONE;

// create object array
pObjArray = New(IGSmartObjArray, g_pMem);

// load file
err2 = IGM3GLoader_LoadFromFile(g_pMem, pLoader, filename, pObjArray);

// get the world object
err = IGSmartObjArray_Get(g_pMem, pObjArray, 0, (void**)&g_pWorld);

// release loader
Del(pLoader, g_pMem, IGM3GLoader);
```

```
    // release object array
    Del(pObjArray, g_pMem, IGSmartObjArray);

    SetTimer(g_hWnd,                    // handle to main window
        0x123456,                   // timer identifier
        100,                        // 0.1-second interval
        (TIMERPROC) NULL);          // no timer callback

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

    return msg.wParam;
}
```

這個函數用來註冊一個 Win32 視窗類別。這是標準 Win32 應用程式要做的事情。

```
//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
```

```
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = (WNDPROC)WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_V4SAMPLE);
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = (LPCSTR)IDC_V4SAMPLE;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

    return RegisterClassEx(&wcex);
}
```

這個函數用來建立一個 Win32 應用程式。

```
//
//    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;

    hInst = hInstance; // Store instance handle in our global variable

    hWnd = CreateWindow(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, g_iWidth, g_iHeight, NULL,
NULL, hInstance, NULL);

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

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

    g_hWnd = hWnd;

    return TRUE;
}
```

這個函數用來處理 Win32 系統傳來的視窗訊息。這是標準 Win32 應用程式要做的事情。

```
//
//  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)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    TCHAR szHello[MAX_LOADSTRING];
    LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
```

```
    switch (message)
    {
        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_ABOUT:
                    DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
(DLGPROC)About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);
            // TODO: Add any drawing code here...
              if (g_pMem && g_pG3D && g_pWorld)
              {
                  HDC memDC;
                  HBITMAP oldObj;
                  memDC = CreateCompatibleDC(hdc);
                  oldObj = (HBITMAP) SelectObject(memDC, g_hBitmap);
```

```
                POINT p1, p2;
                SetMapMode(memDC, GetMapMode(hdc));
                p1.x = g_iWidth;
                p1.y = g_iHeight;
                DPtoLP(hdc, &p1, 1);
                p2.x = 0;
                p2.y = 0;
                DPtoLP(memDC, &p2, 1);

                BitBlt(hdc, 0, 0, p1.x, p1.y, memDC, p2.x, p2.y, SRCCOPY);
                SelectObject(memDC, oldObj);
                DeleteDC(memDC);
            }
            EndPaint(hWnd, &ps);
            break;
        case WM_TIMER:
            switch (wParam)
            {
                case 0x123456:
                    // process the 0.1-second timer
                if (g_pMem && g_pG3D && g_pWorld)
                {
                    int inteval;
                    IGM3GWorld_Animate(g_pMem, g_pWorld, g_Timer,
&inteval);
                    if (g_Timer < 6800) g_Timer += 50;
                    IGM3GGraphics3D_RenderWorld(g_pMem, g_pG3D,
g_pWorld);
```

```
                    InvalidateRect(hWnd, NULL, FALSE);
                    UpdateWindow(hWnd);
                }
                return 0;
            }
            break;
        case WM_DESTROY:
            {
                KillTimer(hWnd, 0x123456);
                IGMemory* pMem = g_pMem;
              // release the whole world
                igSmartRelease(g_pWorld);
                // release graphics 3d
                Del(g_pG3D, g_pMem, IGM3GGraphics3D);
                // release memory pool
                igm3gEngineFinal(g_pMem);
                // free memory
                IGMemory_Final(g_pMem);
                free(g_pMem);
            }
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

這個函數用來處理當使用者按下「關於」功能表。這也是標準 Win32 應用程式要做的事情。

```
// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam)
{
	switch (message)
	{
	   case WM_INITDIALOG:
			return TRUE;

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

其中比較重要的函數譬如 es184CreatePixmap，用來建立一個 OpenGL es 的繪圖緩衝區，做為之後 V4 引擎真正用來繪圖的畫布。我們使用 Win32 的標準 WM_TIMER 視窗訊息來更新動畫，WM_PAINT 視窗訊息將 V4 引擎所繪製的內容拷貝到螢幕上。也使用 WM_DESTROY 視窗訊息來釋放引擎所使用的資源。

4. 數值計算問題

行動裝置由於省電的需求，因此 CPU 處理數值運算時，不像個人電腦那樣可以使用專屬的數值運算輔助器(FPU)。這些 CPU 不僅執行速度比較慢，更糟糕的是它們通常沒有直接以單一個 CPU 指令計算浮點數的能力，而是必須要執行幾百個指令後，才能做完一個浮點數的除法。這兩種差異會變成計算整數的 10 除以 5 只要一個指令就可以做完，但是要執行 10.0 除以 5.0 這兩個浮點數的除法，需要好幾百個指令才能做完。於是浮點數的除法變成「慢上加慢」，導致整個數位遊戲的執行速度變的非常慢。

這使的 3D 繪圖過程中會大量運用到的數值計算，需要特別小心處理，與注意精確度的問題。此外為了增加運算的效能，需要使用特殊格式的數值來進行運算。因此在這些數值格式轉換的過程中，又增加了精確度流失的困擾。

其實就算是浮點數，許多程式師都沒有注意它其實也有相當程度的誤差。以 32 位浮點數來說，它的最低精確度是就是小數的最末位(稱為 LSB)的數值，或是能表示最小數字的一半。由於能表示最小數字是 2^E(忽略指數的偏移誤差)，並且我們有 23 位的小數，所以 LSB 就是 2^{E-24}。例如非常接近 1 個數值，最小的誤差區間(granularity)就是 2^{-24}，或大約是 0.000001192。

4.1 浮點數與定點數

IGV4 引擎為了效能，提供兩種計算浮點數的模式。一種是直接用平臺所提供的浮點數格式去計算。以 iPhone 平台為例，因為使用 ARM code 編譯而非 Thumb 方式，因此可以直接使用浮點數的版本。

有些平臺通常都是靠 CPU 去執行許多個指令才能模擬出計算浮點數的過程，因此速度是比較慢的。另一種是使用稱為定點數(fixed point)的特殊浮點數標記法。用這樣的方法可以大幅加快計算浮點數，但是卻很容易導致精

確度的流失。所以使用者必須在「速度」與「正確性」之間，作出最符合應用程式需求的決定。

誤差則有可能被 3D 繪圖硬體更加擴大。為了達到更好的效能，大多數的 CPU 都是在類似 24 位或 16 位等比較低精確度的模式下計算。24 位的 S16E7 格式使用了 16 位的小數，比單精確度的浮點數還少了 7 位數。這就意味著誤差區間被放大的 2^7，也就是 128 倍之多！16 位浮點數的 S10E5 格式則又比 24 位的浮點數大了 64 倍的誤差區間。這樣講，就很容易理解定點數比浮點數差了多少倍的精確度！

4.2 精確度

V4 引擎使用的定點數是 32 位元的定點數，16 位元的小數加上 16 位的指數。16 位元的定點數所能表示的數值範圍遠比一般的浮點數要來的小。

如果是使用 IGV4 引擎所提供的 OpenGL ES，在它的內部使用的也是定點數 (API 介面是浮點數)。