Contents

- Basic Windows Programming
- Windows GDI
- Preparing for DirectX Programming
Prerequisites

- Windows 2000/XP or later
- Microsoft Visual Studio
  - Visual C++ 6 is not recommended
    - Too old - grammatical flaws and bugs
    - Microsoft’s technical support expired in Sep. 2005
    - Recent DirectX SDKs don’t support VC++ 6 any more
- Microsoft DirectX SDK
  - 9.0c or later
- A DirectX 9 compatible graphic card
  - ATI Radeon 9500+
  - Nvidia GeForce FX, 6/7/8 Series
  - Intel GMA900 integrated graphics or later
  - ATI Express-200 integrated graphics or later
Basic Windows Programming

Computer Graphics, 2008 Spring
How to Program a Win32 Application

- Win32 API
  - The most primitive method
  - C-based definitions
- MFC (Microsoft Foundation Class)
  - Object oriented framework
  - C++ based encapsulation of Win32 API
  - Intuitive UI coding
  - Complicated internal structures
- Other third-party frameworks
  - Qt, GTK, GLUT...

Win32 API is enough for this course. Using some framework is on your choice.
Creating a Win32 Project
Creating a Win32 Project

A project for creating a Win32 application, console application, DLL, or static library

Name: Test
Location: D:\#Worx#
Solution Name: Test

[Screen capture of the New Project dialog in a Visual Studio environment, showing the selection of Project types and Templates for creating a Win32 project.]
Creating a Win32 Project

If you want an empty project and to write whole code, check ‘Empty project’.
Creating a Win32 Project

If you are *really* not interested in Win32 API at all, this is all you should know.
Win32 Application Structure : Brief Description

- **WinMain**
  - Entry point: the application starts from here
  - Contains a message loop

- **WndProc**
  - Callback function
  - The actual message processing routine
Windows : Message-based System

- All of the Windows event is processed via message
WinMain Entry Function

```c
int WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow )
```

- **Parameters**
  - HINSTANCE hInstance : instance handle of the window
  - HINSTANCE hPrevInstace : not used in Win2000/XP
  - LPTSTR lpCmdLine : command line arguments
  - int nCmdShow : showing style of the window
    (maximized, minimized, etc.)
  - wWinMain : unicode version
  - _tWinMain : TCHAR version

_t, TCHAR and LPTSTR are macros for encoding. See ‘TCHAR.H mappings’ on MSDN.
WinMain Entry Function

- Registering a window class
  - Define characteristics of the window
  - Distinguished by “Class Name”

```c
WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS_HREDRAW | CS_VREDRAW;
wcecx.lpfWndProc = WndProc;
wcecx.cbClsExtra = 0;
wcecx.cbWndExtra = 0;
wcecx.hInstance = hInstance;
wcecx.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_TEST));
wcecx.hCursor = LoadCursor(NULL, IDC_ARROW);
wcecx.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wcecx.lpszMenuName = MAKEINTRESOURCE(IDC_TEST);
wcecx.lpszClassName = "MYAPPCLASS";
wcecx.hIconSm = LoadIcon(wcecx.hInstance, MAKEINTRESOURCE(IDI_SMALL));

return RegisterClassEx(&wcex);
```
WinMain Entry Function

- Creating a window
  - Create an instance of the registered window class
  - Show the created window
  - Redraw the window

```c
HWND hWnd = CreateWindow(“MYAPPCLASS”, “My Application”, WS_OVERLAPPEDWINDOW,
  CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd) return FALSE;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);
```
WinMain Entry Function

- Message loop
  - Get messages
    - GetMessage : waiting
    - PeekMessage : polling
  - Translates incoming messages
  - Dispatches translated messages to the WndProc function

MSG msg = {0};
do // message loop
{
  if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) // if there’s a delivered message
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
} while(WM_QUIT != msg.message); // until ‘quit the application’ message is delivered
WndProc Message Callback Function

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

- Parameters
  - Identical to MSG structure
  - hWnd : handle of the window which dispatched the message
  - message : message type
  - wParam, lParam : additional event information

- e.g. mouse move event
  - message : WM_LBUTTONDOWN
  - wParam : state of function keys and mouse buttons
  - lParam : x and y coordinate
WndProc Message Callback Function

- **DefWindowProc**
  - Default message handler function
- **PostQuitMessage**
  - Issue WM_QUIT message

```c
switch (message)
{
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
}
```
Windows GDI
GDI

- **Primitive Windows modules**
  - **Kernel**
    - Memory management and process scheduling
  - **User**
    - UI and window management
  - **GDI (Graphical Device Interface)**
    - Output and graphical processing interface
    - Device-independent abstraction layer
GDI

- **DC(Device Context)**
  - Abstraction of ‘output’ devices
    - Screen
    - Printer
    - Memory (functioned as output buffer)
  - Contains all the information needed to output

- **GDI Object**
  - Abstraction of an information for output
    - Pen, Brush, Font, Bitmap...
    - Contains information of color, size, height...
Drawing on the Window

- Drawing once in WinMain or WM_CREATE handler
  - Problem
    - Does the Windows hold what is drawn in canvas?
Drawing on the Window

- **WM_PAINT message**
  - Issued when the window is need to be redrawn
  - UpdateWindow
    - Just issues WM_PAINT message to the window
- **VS template code**
  - BeginPaint prepares the window for painting
    - gets DC and information of the Window
  - EndPaint marks the end of painting

```cpp
\begin{lstlisting}
case WM_PAINT:
  {
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    // TODO: Add any drawing code here...
    EndPaint(hWnd, &ps);
    break;
  }
\end{lstlisting}
```
Drawing on the Window

- **SetPixel**
  - Draw a pixel
  - GDI object is not necessary
  - Very slow

```c
COLORREF SetPixel(  
    HDC hdc,         // handle to DC  
    int X,           // x-coordinate of pixel  
    int Y,           // y-coordinate of pixel  
    COLORREF crColor // pixel color  
);  
```

- **COLORREF**
  - X8B8G8R8 DWORD  e.g. 0x000000FF
  - RGB(r, g, b) macro  e.g. RGB(0,0,255)
Using GDI objects

- Create GDI objects
  - Creation functions
    - CreatePen, CreateSolidBrush, ...
    - Memory consuming objects → need to be deleted later
  - GetStockObject function
    - pre-defined objects
    - Deletion is not necessary (actually, not allowed!)

- Attach the new object to DC
  - SelectObject function
    - Returns the previous object handle

- Draw with attached objects
- Restore the previous object
- Delete created objects

A black solid pen and a null brush are default GDI objects attached to the window DC.
Using GDI objects

HPEN myPen, myPen2, oldPen;

myPen=CreatePen(PS_DASH, 1, RGB(255,0,0)); // create a red, 3-px-width, dashed pen
myPen2=(HPEN)GetStockObject(BLACK_PEN); // get the black solid pen

SelectObject(hdc, CreateSolidBrush(RGB(0, 255, 0))); // use a green solid brush
oldPen=(HPEN)SelectObject(hdc, myPen); // use myPen

Rectangle(hdc, 200, 200, 300, 300);
MoveToEx(hdc, 50, 50, NULL);
LineTo(hdc, 120, 80);

SelectObject(hdc, myPen2); // use myPen2
LineTo(hdc, 180, 30);
SelectObject(hdc, oldPen); // use previous pen

DeleteObject(myPen); // delete created objects
DeleteObject(SelectObject(hdc, GetStockObject(NULL_BRUSH)));
Buffered Drawing on GDI

- Problems when drawing on the window directly
  - GDI drawing function is slow at all
    - flickering, tearing and shearing
- Buffering
  - Draw or do something with memory buffer
  - Write the buffer to the screen
- In Win32, buffering can be implemented with DIB and Memory DC
Using Bitmap and Memory DC

- **Bitmap**
  - One of the GDI object types
  - DDB(Device Dependent Bitmap)
  - DIB(Device Independent Bitmap)
    - BMP file

- **Memory DC**
  - Not attached to any actual device
  - Consumes memory
    - Delete using DeleteDC function after use
  - Can select BITMAP as a GDI object
    - Bitmap must be compatible with DC
    - The bound bitmap works as ‘surface’ of the DC
    - Impossible for actual device DCs
  - Can be copied to normal DC fast
    - BitBlt, StretchBlt, ...
Using Bitmap and Memory DC

HDC mdc = CreateCompatibleDC(hdc);  // Create a memory DC that is compatible with screen
BITMAPINFO bmi;  // bitmap header
// this is identical to BMP header. if you don’t know about it, just change biWidth and biHeight
bmi.bmiHeader.biSize = sizeof(BITMAPINFO);
bmi.bmiHeader.biWidth = 200;  // bitmap width
bmi.bmiHeader.biHeight = 200;  // bitmap height
bmi.bmiHeader.biBitCount = 32;  // bit count of a pixel
bmi.bmiHeader.biCompression = BI_RGB;
bmi.bmiHeader.biSizeImage = 0;
bmi.bmiHeader.biClrUsed = 0;
DWORD* buf;  // create a DIB with the above header information. Actual buffer pixel data will be allocated to buf
HBITMAP myBitmap = CreateDIBSection(hdc, &bmi, DIB_RGB_COLORS, (void**)(buf), NULL, NULL);
BITMAP myBitmapInfo;
GetObject(myBitmap, sizeof(BITMAP), &myBitmapInfo);
buf[200*100+50] = 0x00FFFF00;  // Now you can access the buffer immediately.
buf[200*50+150] = 0x0000FF00;  // The pixel format is X8R8G8B8.
SelectObject(mdc, myBitmap);
BitBlt(hdc, 10, 10, 200, 200, mdc, 0, 0, SRCCOPY); // copy from mdc to hdc
DeleteObject(myBitmap);  // delete bitmap. buf will be freed.
DeleteDC(mdc);  // delete the memory DC
EndPaint(hWnd, &ps);

Manipulating GDI Bitmaps is so complicated for its device-independent design concepts. If it’s so difficult for you to understand, just use this sample code.
Other GDI features

- See MSDN
Summary

- Make full use of MSDN documents
- If you can’t understand at all...
  - Just remind
    - How to create a window
    - How to draw a pixel
  - It will so slow, but you can complete the assignment #1 with only these features
- For reasonable execution time, using bitmap is recommended

Speed is not a grading factor, but debugging a “SetPixel program” will drain your endurance!
Preparing for DirectX Programming

Computer Graphics, 2008 Spring
Preparing for DirectX Programming

- **DirectX SDK**
  - Development kit for DirectX application
  - Frequent updates
    - Recent update: Nov. 2007
    - The latest version is recommended

- **Supported language**
  - Visual C/C++
  - .NET languages: Managed DirectX

Recent versions of DirectX SDK contain DX10 API for Windows Vista. We use DX9.
Preparing for DirectX Programming

- Visual Studio settings for DirectX
  - Register DirectX include and library directories
    - <Menu> Tools → Options
  - Move the DirectX directories to the top
    - Visual Studio contains old-version DX include and libraries
References

- **Windows Programming**
  - Programming Windows Fifth Edition
    - Charles Petzold, Microsoft Press, 1998
  - MSDN Win32 Platform SDK Documents
    - Online available

- **DirectX Programming**
  - MSDN Direct3D 9 Documents
    - Included in DirectX SDK
    - Online available
    - This course will follow tutorials on this document

If you install the offline MSDN package, “Context-sensitive help” will help you to code more efficiently.
Any Question?