Computer Graphics Programming

- Graphics APIs
- Using MFC (Microsoft Foundation Class) in Visual C++
- OpenGL Programming in Visual C++
- GLUT in Windows and Unix platform
- OpenGL Overview and Application

Graphics APIs

- Provide a software interface to graphics hardware (an intermediary between an application program and the graphics hardware)
- Increase software portability
- Reduce the development time
- Example: OpenGL, DirectX

Graphics APIs Hierarchy

- Application programs
- Window system library
  - Xlib (UNIX)
  - GDI (MS Windows)
- OpenGL Extensions
  - GLX (UNIX)
  - WGL (MS Windows)
- GLU (low-level)
  - GL utility lib
- GLUT
- GL utility tools
Overview of the structure of a window program

- **Overview**
  - Initial activities
    - Process messages from windows (the message loop)

- **Pseudocode**
  - Initialize variables, memory space
  - Create & show window, update window for client area updating
  - Loop
    - Fetch any message sent from Windows to this program
      - If message is QUIT
        - Terminate program, return control to Windows
      - If any other messages
        - Do corresponding actions depending on the message ID and parameters
        - Return control to Windows
    - End Loop

MFC Program

- **MFC**
  - MFC was created to make programming in Windows easier. It automates many routine programming tasks.

- **AppWizard**
  - Creating MFC Doc/View application using AppWizard.
  - Files
    - CMainFrame /*create and define window, toolbars, ..*/
    - CLibraryApp /*initial instance, show and update window...*/
    - CLibraryDoc /*new document, serialize, ...*/
    - CLibraryView /*display, draw, ...*/

- **Resource**
  - Menu, Mouse, Dialog, Key activity

Tutorial: MFC Program
OpenGL in VC++

- **OpenGL Win32**: The way to describe and represent the object in the real world.

  - **Including files**:
    ```
    #include <gl/gl.h>
    #include <gl/glu.h>
    ```
    Make sure that the corresponding library is linked: `opengl32.lib`, `glu32.lib`

  - **Device context**
    - Initializing OpenGL under Win32 is to grab a device context (DC). (Since we are drawing to a window)
    ```
    HDC hDC = GetDC(hWnd)
    ```
    /* In order to make our drawing independent to different devices, Windows uses "Graphics Device Interface (GDI) to handle this. GDI provides an "abstract surface" where we can draw on. which is called device context (DC).  DC is managed by GDI. Commands to draw on a DC are same regardless of hardware (video card, printers…), GDI translates drawing commands to hardware commands to output to the physical device. */
    /* DC is accessed with a "handle to a DC", and it is got from Windows using GetDC() */
    - When we've done with OpenGL and any other drawing,
      ```
      ReleaseDC(hWnd, hDC)
      ```

- **Pixel format for the DC**
  Before we jump to the OpenGL, we need to set the pixel format, this informs the system how we are going to use the DC.
  ```
  PIXELFORMATDESCRIPTOR pfd;
  /*define the low-level aspects, like double buffering, z-buffer, color format, alpha buffer, etc*/
  if (SetPixelFormat(…);)
  ```

- **Double buffer**
  ```
  SwapBuffers(hDC);
  ```

- **Pixel format for the DC**
  Before we jump to the OpenGL, we need to set the pixel format, this informs the system how we are going to use the DC.
  ```
  PIXELFORMATDESCRIPTOR pfd;
  /*define the low-level aspects, like double buffering, z-buffer, color format, alpha buffer, etc*/
  if (SetPixelFormat(…);)
  ```

- **Double buffer**
  ```
  SwapBuffers(hDC);
  ```
OpenGL Win32: The way to describe and represent the object in the real world.

• Render Context

There are a couple more function calls before we can start playing with OpenGL

The next step is to create a render context (RC).

Just like anything else in Win32, we are dealing with a handle and so we have the HGLRC, this is our bridge to the OpenGL system.

Once we have a DC and set the pixel format, we can begin by initializing OpenGL and enabling our RC.

- Create our OpenGL context
  
  ```
  HGLRC hRC;
  hRC = wglCreateContext(hDC);
  ```

- After we're done working with the RC, we have to delete it to free system resources:
  
  ```
  wglDeleteContext(hRC);
  ```

- In multi-tasking environment (several instances of OpenGL RC running in the same time), how OpenGL knows where the commands are going? Do this by making our OpenGL RC current:
  
  ```
  wglMakeCurrent(hDC, hRC);
  ```

- If you want to preserve the previous current DC and RC, you can:
  
  ```
  HDC hOldDC = wglGetCurrentDC();
  HGLRC hOldRC = wglGetCurrentContext();
  wglMakeCurrent(hDC, hRC);
  // do our OpenGL stuff in between here …
  wglMakeCurrent(hOldDC, hOldRC);
  ```

GLUT (Graphics Language Utility Tools)

Advantage:

- Simplify opening window, detecting input (key and mouse), and so on.

  ```
  glutInit(argc, argv);
  glutCreateWindow(windowName);
  glutDisplayFunc(displayFunction);
  glutMainLoop();
  glutSwapBuffers();
  ```

OpenGL

2D/3D Graphics API
- Most widely used/supported by industry
- Runs on Unix, Irix, Linux, Windows 85/98/2000/NT, OS/2, etc.
- Works with X-Windows, Win32, Mac OS, etc.
- Hardware independent
- Commands for rendering, easy to use (portable, scalable, reliable)
- Commands for windowing task and input event handling are provided by GLUT (GL Utility Tools)

Features:
- Low-level graphics library specification
- It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps.
- It provides a set of commands that allow the specification of geometric objects in two or three dimensions
- Using the provided primitives together with commands, we can control how the objects are rendered into frame buffer.
- It is operating system and windowing system independent

Syntax:
command ---- glXXXX (e.g., glBegin)

Primitives:
- All geometric objects are ultimately described as an ordered set of vertices:
  glBegin(GL_LINES)
  glVertex*()
  …
  glEnd()
- Primitives for use in constructing geometric entities:
  GL_POINTS, _LINES, _LINE_STRIP, _LINE_LOOP,
  GL_POLYGON, _QUADS, _TRIANGLES, …..
OpenGL

**Primitives:**
- Windows functions: `SetPixel(hDC, x, y, colorRef)`
- OpenGL: `glBegin(GL_POINTS)`
  - MoveToEx(hDC, x, y, NULL)
  - LineTo(hDC, x, y)
  - Polygon(hDC, lppts, num_points)

**Attributes (State Variables):**
- Property of primitives: color, line style, area fill patterns, etc.
  - e.g., `glColor3f(R, G, B)`

**Transformations – done with matrix math:**
- Setting window/viewports, object geometric transformation (e.g., rotation, translation, etc.): `glLoadMatrix()`, `glRotatef()`, `glTranslatef()`, `glScalef()`, `glViewport()`, ...
- `glPushMatrix()`, `glPopMatrix()`, `glMatrixMode()`, ...

**Execution:**
- `glFlush()`
  - It is necessary to call this command to ensure all previously issued commands are executed.

**Changing the State:**
- `glEnable()`, `glDisable()`

**Drawing in Three Dimensions:**
- Hidden surface removal is achieved through the use of a depth buffer and depth testing:
  - `glEnable(GL_DEPTH_TEST)`
- Culling the face (indicate which polygons should be discarded before they are converted to screen coordinates):
  - `glEnable(GL_CULL_FACE)`
  - `glCullFace(GL_BACK)`
OpenGL

- **Viewing**
  - Viewing transformation is analogous to positioning and aiming a camera.
  - Modeling transformation is analogous to positioning and orienting the model to be viewed.
  - Viewing transformations must precede modeling transformations in OpenGL code.

```c
// Projection
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
glOrtho(…)

// Modelview
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
```

- **Viewport transformation** is analogous to choosing the size of the developed photograph.

```c
// Viewport
glViewport(…)
```

OpenGL

- **Input/Interaction**
  - Auxiliary library (aux, GLX, GLUT), e.g., glutMouseFunc( )

- **Control/Housekeeping**
  - Initialize graphics system, put window up, etc.
    - e.g., auxInitWindow(), glutCreateWindow(), etc.
    - (Windows uses CreateWindow())

- **Storing/Retrieving bitmap images**
  - glReadPixels(), glDrawPixels(), glCopyPixels()
    - (Windows use BitBlt(), StretchBlt(), etc.)

- **Photorealism**
  - Hidden surfaces, lighting, shading, etc.
    - e.g., glLightf(), glColorMaterial(), glShadeModel(), etc.

OpenGL

- **Display Lists**
  - A display list is a group of OpenGL commands that have been stored (cached) for later execution. When a display list is invoked, the commands in it are executed in the order in which they were issued.
  - One common use is creating a display list for an object that is to be drawn more than once. If the object is drawn by calculating data that calculations need only be performed rather than each time the object is drawn:

```c
glNewList(1)
………
glEndList()
```

- **Animation**
  - Redraw + Swap.
OpenGL

Texture Mapping:
//Initialization
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
gBindTexture(GL_TEXTURE_2D, texName[1]);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, BMBACK.bmWidth, BMBACK.bmHeight, GL_BGR_EXT, GL_UNSIGNED_BYTE, BMBACK.bmBits);
gTexImage2D(….., bmBits);

//Texture mapping
glBindTexture(GL_TEXTURE_2D, texName[1]);
gTexCoord2f(0,0);
gVertex3f(-BM.bmWidth,0,0);
gTexCoord2f(0,1);
gVertex3f(-BM.bmWidth,BM.bmHeight,0);
gTexCoord2f(1,1);
gVertex3f(0,BM.bmHeight,0);
gTexCoord2f(1,0);
gVertex3f(0,0,0);
gEnd();

Typical OpenGL Program Organization:
main:
Find GL visual (display) and create window
initialize GL states (viewing, color, lighting)
initialize display list
loop
check for events (and process them)
if window event (window moved, exposed, etc.)
modify viewport, if needed
redraw
else if mouse or keyboard
do something, e.g., change states and redraw
redraw:
clear screen (to background color)
change state(s), if needed
render more states
render some more graphics
swap buffers

OpenGL Order of Operation:
(1) Construct shapes from geometric primitives
(2) Arrange the objects in 3D space, select the view.
(3) Calculate the color of all objects
(4) Convert the mathematical description of objects and color information to pixels on the screen (called rasterization)

Reference:
http://www.geocities.com/SiliconValley/Code/1219/opengl32.html
http://devcentral.iftech.com/articles/MFC/simple_MFC/default.php
http://www.eecs.tulane.edu/www/Terry/OpenGL/Introduction.html
http://fly.cc.fer.hr/~unreal/theredbook/
or
http://ask.ii.uib.no/ebt-bin/nph-dweb/dynaweb/SGI_Developer/OpenGL_PG/