# OpenGL & GLUT

321190
2013년 봄학기
3/12/2013
박경신

## OpenGL & GLUT & GLEW

- OpenGL
  - http://www.opengl.org/
  - http://www.sgi.com/software/opengl
  - Windows95 이후 OpenGL 이 표준으로 들어가 있음.
  - ftp://ftp.microsfot.com/softlib/MSLFILES/opengl95.exe
- freeglut for Win32
  - http://freeglut.sourceforge.net/
  - freeglut-MSVC-2.8.0-1.mp.zip 내려받기
- glew for Win32
  - http://glew.sourceforge.net/
  - glew-1.9.0-win32.zip 내려받기



## Installing OpenGL & GLUT & GLEW

- Libraries를 Visual C++의 **C:₩Program Files₩Microsoft SDKs₩Windows₩v7.0A₩lib₩**에 설치
  - **freeglut.lib**
  - **glew32.lib glew32mx.lib**
- Include files을 Visual C++의 **C:₩Program Files₩Microsoft SDKs₩Windows₩v7.0A₩include₩GL₩** 에 설치
  - **freeglut.h freeglut_ext.h freeglut_std.h glut.h**
  - **glew.h glxew.h wglew.h**
- Dynamically-linked libraries를 **C:₩Windows₩system32** (Windows 7 32-bit운영체제 기준)에 설치
  - **freeglut.dll, glew32.dll, glew32mx.dll**
  - **C:₩Windows₩SysWOW64** (Windows 7 64-bit운영체제 기준)에도 freeglut.dll, glew32.dll, glew32mx.dll 설치



## Compiling OpenGL Programs

- Visual Studio 2010 VC++ 실행
- 프로젝트 새로 만들기
  - 메뉴에서 File->New->Projects
  - Win32 Console Application 선택
  - 프로젝트 이름 지정
- Linker에 library files을 지정
  - 메뉴에서 Project->Settings->link->Object/library modules (Project->Properties->Linker->Input->Additional dependencies)
  - **glew32.lib freeglut.lib**를 넣는다
- 프로젝트에 파일 새로 만들기
  - 메뉴에서 Project->Add to Project-> Files
  - 파일 이름 지정
- 빌드 (build) (F7)와 실행 (execute) (F5)

## Windows System

- 윈도우 시스템
  - Microsoft Windows
  - X Window systems
- 윈도우 시스템과 OpenGL 시스템은 모두 래스터 그래픽스 시스템임
- OpenGL 프로그래밍을 하기 위해서는
  - 사용 윈도우 시스템에서 제공하는 래스터 시스템을 기반으로 윈도우 프로그래밍 수행
  - 윈도우 프로그래밍 문맥에서 추상적인 래스터 시스템인 OpenGL 시스템을 윈도우 시스템에 연결
  - OpenGL에서 제공하는 함수들을 사용하여 3차원 그래픽스 프로그래밍을 수행
  - 원하는 OpenGL 작업이 실제로 하드웨어를 제어하고 있는 사용 윈도우 시스템이 효율적으로 이해할 수 있는 형태로 전환



## OpenGL

- 실리콘 그래픽스사 (SGI)가 개발한 3차원 그래픽스 라이브러리 API (1992)
- 2차원 그래픽스는 (z축의 값을 0으로 처리한) 3차원의 특수 경우로 봄
- OpenGL 그래픽스 함수는 프로그래밍 언어에 독립적인 기능으로 지정되어 있음
  - C/C++, Java, Fortran, Python 등 다수 언어와 사용가능
- OpenGL은 하드웨어에 중립적임 (Window system이나 OS에 독립적인 API)
  - No I/O library
  - No specific model loading mechanism
  - No hardware specific functions (but available as extensions)



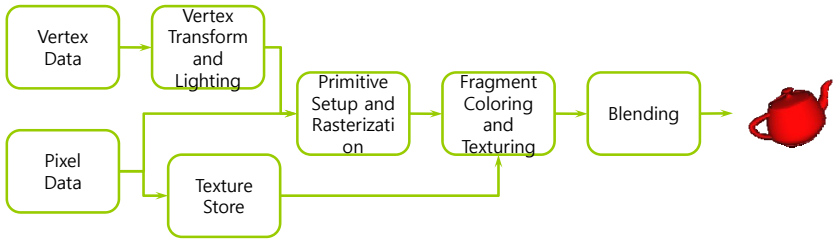
## OpenGL Evolution

- 초창기에 Architectural Review Board (ARB)에 의해 주도
  - SGI, Microsoft, Nvidia, HP, 3DLabs, IBM 등
  - 현재는 Kronos Group
- OpenGL 3.1
  - 전적으로  Shader 기반으로 감. 즉, 모든 응용프로그램은 반드시 vertex 와 fragment shader를 사용해야 함
  - 대부분 2.5 함수는 deprecated (누락될 것이라 사용을 권유하지 않음)
- OpenGL ES
  - 임베디드 시스템용.
  - Ver1.0은 OpenGL 2.1을 간소화. Ver2.0은 OpenGL 3.1을 간소화
- WebGL
  - OpenGL ES 2.0을 Javascript로 구현
- OpenGL 4.1 & 4.2
  - Geometry shader와 tesselator 추가



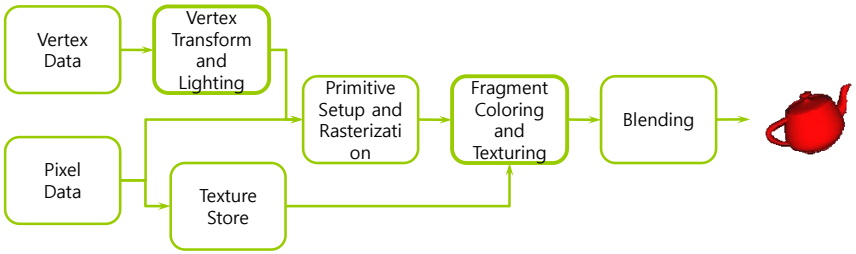
## OpenGL 1.0

- OpenGL 1.0은 1994년 7월 1일 발표
- OpenGL 1.0의 그래픽스 파이프라인은 fixed-function으로 OpenGL 1.1부터 OpenGL 2.0 (2004/09)까지 사용



## OpenGL 2.0 Programmable Pipeline

- OpenGL 2.0은 공식적으로 Programmable Shader를 추가
  - Vertex Shader
  - Fragment Shader
- 그러나, Fixed-function pipeline도 사용 가능



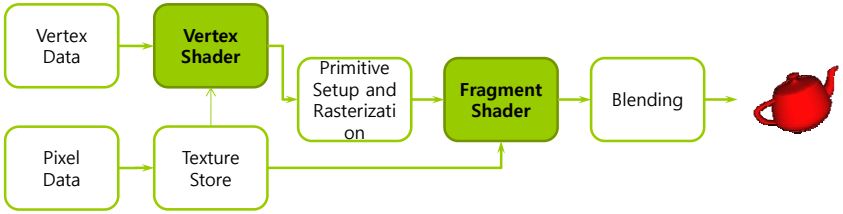

## OpenGL 3.0

- OpenGL 3.0은 *deprecation model* 을 소개
  - OpenGL 3.0 이전 버전까지는 backward compatibility를 지원하기 위하여 함수를 추가해왔지만 OpenGL 3.0부터 더 이상 backward compatibility를 지원하지 않고 새로운 함수를 제공함
- OpenGL 3.1 (2009/03/24)까지는 그래픽스 파이프라인이 동일하게 적용
- OpenGL 3.1부터 forward compatible contexts를 지원

| Context Type | Description |
|---|---|
| Full | Includes all features (including those marked deprecated) available in the current version of OpenGL |
| Forward Compatible | Includes all non-deprecated features (i.e., creates a context that would be similar to the next version of OpenGL) |

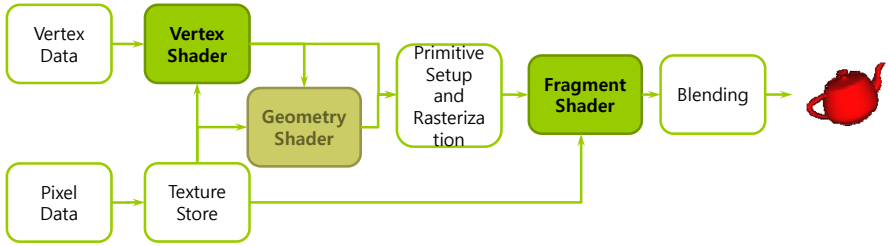
## Exclusively Programmable Pipeline

- OpenGL 3.1은 Fixed-function pipeline을 제거함
  - 프로그램은 오로지 shader를 사용해야 함
- 추가적으로, 모든 데이터는 GPU에 보내어져서 사용함
  - 모든 정점 데이터는 buffer objects를 사용하여 보내어짐




## More Programmability

- OpenGL 3.2 (2009/09/03)은 Geometry Shader 단계를 추가함



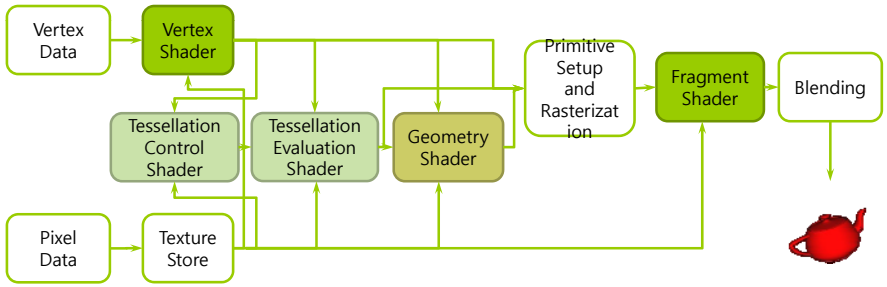
## More Evolution – Context Profiles

- OpenGL 3.2는 context profile 개념을 소개
  - Profile이란 응용프로그램에서 features를 선택하기 쉽게 하기 위하여 제공
  - 현재 2가지 profile이 존재: core & compatible

| Context Type | Profile | Description |
|---|---|---|
| Full | core | All features of the current release |
| | compatible | All features ever in OpenGL |
| Forward Compatible | core | All non-deprecated features |
| | compatible | Not supported |

## OpenGL 4.1 & 4.3

- OpenGL 4.1 (2010/07/25)에서는 tessellation-control shader와 tessellation-evaluation shader를 추가
- 가장 최근 버전은 OpenGL 4.3




## OpenGL Libraries

- OpenGL core library
  - 윈도우 시스템에서는 OpenGL32
  - 유닉스/리눅스 시스템에서는 libGL.a
- OpenGL Utility Library (GLU)
  - OpenGL core를 추가 지원함
- OpenGL Windows Toolkit
  - Different platforms have different ways to integrate OpenGL with their windowing environment
  - X Window System (GLX)
  - Apple (AGL)
  - Windows (WGL)
  - IBM OS/2 (PGL)
  - Cross-platform (GLUT – OpenGL Utility Toolkit)



## Freeglut & GLEW

- Freeglut는 GLUT를 개선한 버전
  - 초기 GLUT는 오래되고 더 이상 개선되지 않고 있음.
  - freeglut는 GLUT를 개선한 것으로 추가적인 기능 제공함.
- GLEW (OpenGL Extension Wrangler Library)
  - 하드웨어와 연관이 되어 있는 OpenGL은 사용하기 어려움. 함수포인터를 얻어내야 하며 Extension도 신경써야 함.
  - GLEW는 GLSL같은 OpenGL extension을 편리하게 사용할 수 있도록 함
  - OpenGL 응용프로그램에서 #include <glew.h>와 glewInit()를 불러서 초기화하면 됨

## OpenGL & GLSL

- 쉐이더 기반의 OpenGL
  - 대부분의 state variables, attibutes and related pre 3.1 OpenGL funtions 이 더 이상 사용되지 않음.
  - 모든 것이 쉐이더를 통해 이루어짐
- GLSL (OpenGL Shading Language)
  - Nvidia의 Cg나 Microsoft HLSL과 비슷한 C 언어와 흡사함
  - 코드는 쉐이더로 보내어져서 GPU를 통해 렌더링이 수행됨



## OpenGL/GLUT Programming

- 단순히 윈도우를 여는 프로그램 예제

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <GL/freeglut_ext.h>
void display (void)
{
}
int main(int argc, char *argv[])
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGBA);
      glutInitWindowSize(512, 512);
      glutCreateWindow(argv[0]);
      glewInit();
      glutDisplayFunc(display);
      glutMainLoop();
      return 0;
}
```



## OpenGL/GLUT Programming

- 윈도우를 전부 하얀색으로 칠하는 프로그램 예제

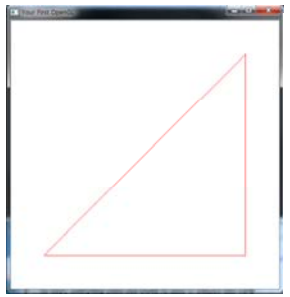
```
void display (void) {
      glClear(GL_COLOR_BUFFER_BIT);
      glFlush();
}
void init (void)  {
      glClearColor(1.0, 1.0, 1.0, 1.0);
}
int main(int argc, char *argv[])
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGBA);
      glutInitWindowSize(512, 512);
      glutCreateWindow(argv[0]);
      glewInit();
      init();
      glutDisplayFunc(display);
      glutMainLoop();
      return 0;
 }
```



## OpenGL/GLUT Program 작성 예

- glVertex를 이용한 OpenGL 2.x 프로그램

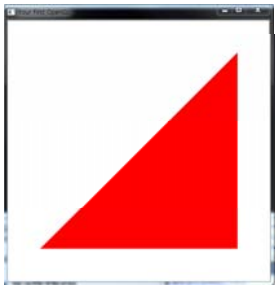
```
void display (void)
{
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1.0, 0.0, 0.0);
      glBegin(GL_LINE_LOOP);
      glVertex3f(-0.75, -0.75, 0.0);
      glVertex3f(0.75, -0.75, 0.0);
      glVertex3f(0.75, 0.75, 0.0);
      glEnd();
      glFlush();
}
void init (void)
{     /* 변경없음 */
}
int main(int argc, char *argv[])
{     /* 변경없음 */
}
```



## OpenGL/GLUT Programming

- glVertex를 이용한 OpenGL 2.x 프로그램

```
void display (void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex3f(-0.75, -0.75, 0.0);
    glVertex3f(0.75, -0.75, 0.0);
    glVertex3f(0.75, 0.75, 0.0);
    glEnd();
    glFlush();
}
```

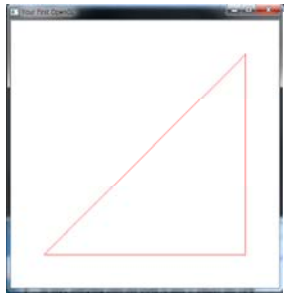




## OpenGL/GLUT Program 작성 예

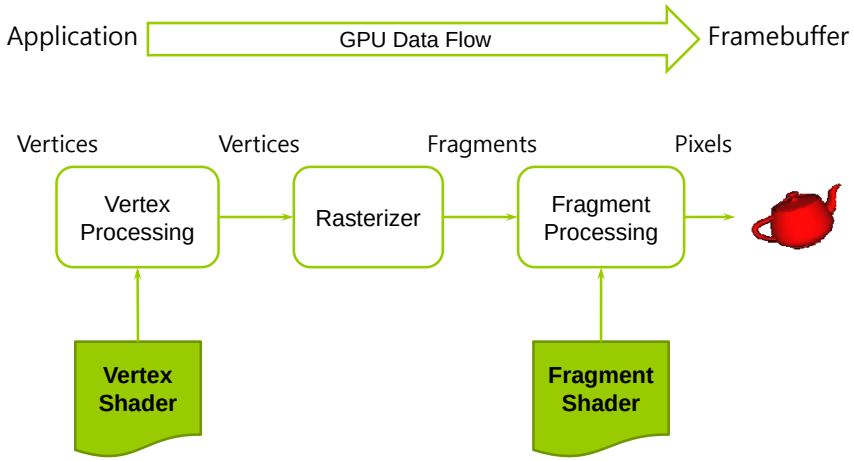
- OpenGL 3.x 프로그램

```
void display (void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    /* 이 부분을 쉐이더로 작성 */

    glFlush();
}
```





## OpenGL 3.x Simplified Rendering Pipeline




## Modern OpenGL Programming in a Nutshell

- Modern OpenGL (OpenGL 3.x) 프로그래밍은 다음과 같은 단계로 진행
  1. Shader 프로그램을 만든다.
  2. Vertex 자료를 Vertex Buffer Object (VBO)와 Vertex Array Object (VAO)를 만들고 이 자료를 쉐이더에 로딩한다.
  3. 이 자료의 위치와 쉐이더의 변수와 "연결(Connect)" 한다.
  4. 렌더링을 수행한다.

## OpenGL 3.x Program Structure

- OpenGL 3.x 프로그램의 일반적인 구조
  - **main()**: 관련 콜백 함수 지정 등
    - Specifies the callback functions
    - Opens one or more windows with the required properties
    - Enters event loop (last executable statement)
  - **init()**: 상태변수 (state variables) 지정
    - Viewing
    - Attributes
  - **initShader():** read, compile and link shaders
  - callbacks
    - Display function
    - Input and window functions

## OpenGL/GLUT Programming

```
#include <GL/glew.h>
#include <GL/freeglut.h>          ← includes gl.h
#include <GL/freeglut_ext.h>

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);   ← specify window properties
    glutInitWindowSize(512,512);
    glutInitWindowPosition(0,0);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);     ← display callback
    glewInit();
    init();                       ← set OpenGL state and initialize shaders
    glutMainLoop();               ← enter event loop
}
```

## OpenGL/GLUT Programming

- void glutInit(int *argc, char **argv)
  - GLUT와 OpenGL 환경을 초기화. 인수에는 main의 인수를 그대로 건네줌.
- void glutInitWindowSize(int width, int height)
  void glutInitWindowPosition(int x, int y)
  - 윈도우 크기를 지정함. 윈도우 위치를 지정함.
- int glutCreateWindow(char *name)
  - 윈도우를 여는 함수. 인수 name은 그 윈도우의 이름이 타이틀 바에 표시됨.
- void glutDisplayFunc(void (*func)(void))
  - 인수 func는 열린 윈도우 내에 디스플레이하는(즉, 그림을 그리는) callback함수 포인터. 윈도우가 열리거나 다른 윈도우에 의해 숨겨진 윈도우가 다시 디스플레이될 때 이 함수가 실행
- void glutMainLoop(void)
  - GLUT 루프. 이 함수의 호출로 프로그램은 이벤트를 기다리는 상태임.

## OpenGL/GLUT Programming

- void glutInitDisplayMode(unsigned int mode)
  - 디스플레이의 표시모드를 설정. Mode에 GLUT_RGBA를 지정했을 경우는 색의 지정을 RGB로 사용함을 지정. 그 밖에 인덱스 칼라모드 (GLUT_INDEX)를 지정하면 효율을 향상시킬 수 있음.
- void glClearColor(GIclampf R, GIclampf G, GIclampf B, GIclampf A)
  - 윈도우를 전부 칠할 때의 색을 지정. R, G, B, A는 0~1 사이의 값을 가짐. (0, 0, 0, 1)을 지정하면 검정색의 불투명을 그림.
- void glClear(Glbitfield mask)
  - 윈도우를 전부 칠함. Mask에는 전부 칠하는 버퍼를 지정한다. OpenGL이 관리하는 화면상의 버퍼(메모리)에는 color buffer, depth buffer, stencil buffer, overlay buffer, 등이 겹쳐서 존재함. GL_COLOR_BUFFER를 지정했을 때는 컬러버퍼만 전부 칠해짐.
- void glFlush(void)
  - 이 함수는 아직 실행되지 않은 OpenGL명령을 전부 실행.

## Immediate Mode Graphics

- Geometry는 정점 (vertices)으로 지정
  - 2차원 또는 3차원 공간에 한 지점에 정점으로 지정
  - Points, lines, circles, polygons, curves, surfaces
- Immediate mode
  - 응용프로그램에서 매번 정점이 지정되고, 그 위치 값이 GPU에 보내어짐
  - **과거에 glVertex를 사용했었음**
  - CPU와 GPU 간에 병목 현상이 발생함
  - OpenGL 3.1 부터 사라짐

## Retained Mode Graphics

- 모든 정점(vertex)과 속성(attribute) 정보를 배열(array)로 지정
- 이 배열을 GPU에 보내어 저장하고 렌더링에 사용함

## Display Callback

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    glFlush();
}
```

- glDrawArrays를 사용하여 그리기를 수행함.
- 이때, 배열(arrays)는 정점 배열(vertex arrays)을 가진 버퍼 객체(buffer objects)임

## Vertex Arrays

- 정점 (Vertices) 속성들
  - Position
  - Color
  - Texture Coordinates
  - Application data

```
const float vertexPositions[] =
{
   -0.75f, -0.75f, 0.0f, 1.0f,
   0.75f, -0.75f, 0.0f, 1.0f,
   0.75f, 0.75f, 0.0f, 1.0f,
};
```

## Vertex Array Object

- Vertex Array Object는 모든 정점 자료(positions, colors, ..)를 묶어줌.
  - 하나의 VAO에 여러 개의 Buffer Object을 가질 수 있음. 예로, 위치 정보 버퍼와 색상정보를 위한 버퍼를 VAO에 사용.
  - 보통 하나의 Mesh마다 하나의 VAO를 사용함
- VAO를 생성하고, bind함.

```
GLuint vao;          // vertex array object
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
```

  - 이것은 Vertex Array만을 가진다는 의미로 아직 실제 데이터(content)는 없음.

## Buffer Object

- Buffer object는 대용량 자료를 GPU에 보내줄 수 있음.
- 버퍼를 생성하고, bind하고, 자료를 넣어줌.

```
GLuint buffer;       // vertex buffer object
glGenBuffers(1, &buffer);
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertexPositions),
                              vertexPositions, GL_STATIC_DRAW);
```

- 그리고 glVertexAttribPointer를 호출하여 현재 바인드 된 버퍼가 몇 번째 Attribute 버퍼인지 어떤 속성을 갖는지 알려줌. 다음 glEnableVertexAttribArray(index)를 호출해야 해당 버퍼의 정보가 렌더링 됨.

```
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
```

## Initialization

- **init()** 에서 Vertex Array Objects와 Buffer Objects 지정
- 또한 일반적으로 GL의 clear color와 parameters도 지정
- 또한 shaders도 지정
  - Read
  - Compile
  - Link

## Shader Functions

- glCreateProgram – 프로그램 (vertex shader, geometry shader, fragment shader를 포함하는 파이프라인) 생성
- glCreateShader – GL에서 쉐이더 공간을 만들어 달라고 요청. 단지 공간을 생성하기 위한 함수
- glShaderSource – 할당한 쉐이더 공간과 쉐이더 소스 파일을 읽어들인 버퍼를 연결
- glCompileShader – 쉐이더를 컴파일
- glAttachShader – 컴파일된 쉐이더를 프로그램에 붙임
- glLinkProgram – 링크하고 에러 체크. 링크가 완료되면 프로그램이 준비 완료된 것임
- glUseProgram – 이 프로그램을 사용

## OpenGL/GLUT Programming

```
// Create a NULL-terminated string by reading the provided file
static char* readShaderSource(const char* shaderFile)
{
    FILE* fp = fopen(shaderFile, "r");
    if ( fp == NULL ) { return NULL; }

    fseek(fp, 0L, SEEK_END);
    long size = ftell(fp);

    fseek(fp, 0L, SEEK_SET);
    char* buf = new char[size + 1];
    fread(buf, 1, size, fp);

    buf[size] = '₩0';
    fclose(fp);

    return buf;
}
```

## OpenGL/GLUT Programming

```
// Create a GLSL program object from vertex and fragment shader files
GLuint InitShader(const char* vShaderFile, const char* fShaderFile)
{
    struct Shader { const char* filename; GLenum type; GLchar* source; }
        shaders[2] = { { vShaderFile, GL_VERTEX_SHADER, NULL },
                       { fShaderFile, GL_FRAGMENT_SHADER, NULL } };
    GLuint program = glCreateProgram();
    for ( int i = 0; i < 2; ++i ) {
        Shader& s = shaders[i];
        s.source = readShaderSource( s.filename );
        if ( shaders[i].source == NULL ) {
            std::cerr << "Failed to read " << s.filename << std::endl;
            exit( EXIT_FAILURE );
        }
        GLuint shader = glCreateShader( s.type );

        glShaderSource( shader, 1, (const GLchar**) &s.source, NULL );
        glCompileShader( shader );
```

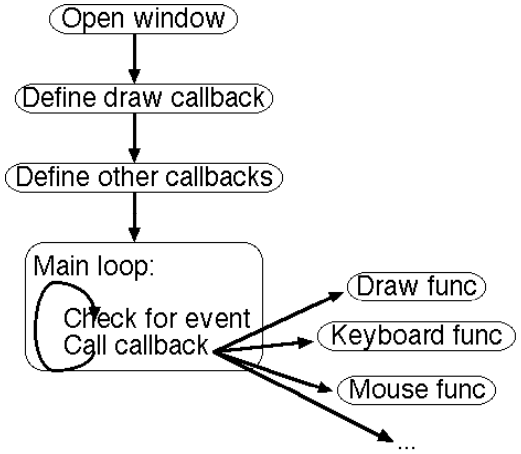
## OpenGL/GLUT Programming

```
        // 중간 생략...
        delete [] s.source;
        glAttachShader( program, shader );
    }
    glLinkProgram(program);
    GLint  linked;
    glGetProgramiv( program, GL_LINK_STATUS, &linked );
    if ( !linked ) {
        std::cerr << "Shader program failed to link" << std::endl;
        GLint  logSize;
        glGetProgramiv( program, GL_INFO_LOG_LENGTH, &logSize);
        char* logMsg = new char[logSize];
        glGetProgramInfoLog( program, logSize, NULL, logMsg );
        delete [] logMsg;
        exit( EXIT_FAILURE );
    }
    glUseProgram(program);
    return program;
}
```

## GLUT (OpenGL Utility Toolkit)

- Mark J. Kilgard가 개발한 portable windowing and interaction API
  - Prefix "glut"로 시작
  - 대부분의 window system에 보편적인 기능들을 wrapping한 상위 interface제공 (portable across all PC and workstation OS platforms)
  - OpenGL이 제공하는 범위보다 상위 수준의 utility function도 제공
  - UNIX/X-window에서 개발된 code를 그대로 재사용 가능
  - Win32, MFC, Xlib을 알 필요가 없음
  - 그러나 Window system의 기능을 제한적으로만 이용 가능

## GLUT Program Structure




## GLUT Callbacks

- GLUT defines a basic program structure - an *event loop*, with *callback functions*.
- *Callback,* a function that you provide for other code to call when needed; the "other code" is typically in a library
- GLUT uses callbacks for drawing, keyboard & mouse input, and other events.
- Whenever the window must be redrawn, your drawing callback is called.
- Whenever an input event occurs, your corresponding callback is called.

## GLUT Callbacks

- glutDisplayFunc
- glutPostRedisplay
- glutIdleFunc
- glutTimerFunc
- glutGetModifiers
- glutIgnoreKeyRepeat
- glutKeyboardFunc
- glutKeyboardUpFunc
- glutSpecialFunc
- glutSpecialUpFunc

## GLUT Callbacks

- Handling *Display* Callbacks
  - glutDisplayFunc(void (*func)(void))
    - Specifies a function that would draw the graphical contents
    - Argument: display function name
- Handling *Input* Events
  - glutReshapeFunc(void (*func)(int w, int h))
    - What action should be taken when the window is resized
  - glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))
  - glutMouseFunc(void (*func)(int button, int state, int x, int y))
    - Handle keyboard key and mouse button
  - glutMotionFunc(void (*func)(int x, int y))
    - What to do when the mouse is moved while a mouse button is pressed

## The *Reshape* Event Callback

- This callback sets up OpenGL to display images in a window of the new size
- The value w is the new width, and the value h is the new height

```
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
}
```

## The *Keyboard* Event Callback

- This callback simply causes the program to exit when the user hits the ESC key

```
void keyboard(unsigned char key, int x, int y)
{
  switch (key): /* ESC-key & q-key exits the program */
   {
      case 27:
      case 'q':
         exit(0);
   }
}
```

## GLUT Functions

- glutMainLoop(void)
  - Enter the GLUT event processing loop
- glutPostRedisplay()
  - Ensures that the window gets drawn at most once each time GLUT goes through the event loop.
  - In general, never call the display callback directly, but rather use the glutPostRedisplay() whenever the display needs to be redrawn.