GLEW 기반 모던 OpenGL 개발환경 구축하기

김준호

Abstract

OpenGL 2.x 이후 버전부터는 프로그래머가 쉐이더(shader)를 통해 그래픽카드의 작동방식을 자신이 원하는 방식대로 동작할 수 있도록 프로그래밍할 수 있는 '프로그래밍 가능한 렌더링 파이프라인(programmable rendering pipeline)'을 지원한다. 쉐이더를 쓸 수 없었던 OpenGL 1.x 프로그래밍 방식과 대비하여, 쉐이더를 통해 렌더링 파이프라인을 프로그래밍 할 수 있는 OpenGL 2.x 이후의 프로그래밍 방식을 모던 OpenGL(modern OpenGL)이라 부른다. 여기서는 Ubuntu 16.04 LTS에서 FreeGLUT, GLEW등의 오픈소스 라이브러리를 이용하여 모던 OpenGL 기반 그래픽스 개발 환경을 구축해 본다.

1 모던 OpenGL (modern OpenGL)

OpenGL 2.x 이후의 모던 OpenGL(modern OpenGL)은 기존 OpenGL 1.x가 사용하는 고정 렌더링 파이프라인(fixed rendering pipeline)을 버리고 프로그래밍 가능한 렌더링 파이프라인(programmable rendering pipeline)을 지원한다.

여기서는 국민대학교 소프트웨어학부의 공식 실습환경인 Ubuntu 16.04 LTS에서 프로그래밍 가능한 렌더링 파이프라인을 지원하는 모던 OpenGL 개발 환경을 구축하기 위해 GLEW를 설치하는 방법을 학습한다.

2 FreeGLUT

FreeGLUT개발환경은 'FreeGLUT 기반 OpenGL 개발환경 구축하기'를 참고하여 설치하도록 한다.

3 GLEW를 이용한 모던 OpenGL 개발환경 구축

nVidia나 AMD와 같은 그래픽스 하드웨어 업체들이 매년 새롭게 출시하는 그래픽카드는 기존에 없는 새로운 하드웨어 가속기능을 제공하고 있다. 구매자 입장에서는 이러한 그래픽카드의 하드웨어적인 성능을 'OpenGL 4.4 지원, DirectX 11.1 지원' 등과 같은 방식으로 얼마나 최신 그래픽스 API 지원하는가로 체감하게 된다.

3.1 OpenGL API 버전의 의미

OpenGL API를 관리하는 비영리 재단인 크로노스 그룹(Khronos group)에서는 nVidia나 AMD 등과 같은 그래픽스카드 제조사로부터 새로운 그래픽스 하드웨어 기능을 지원하는 API 요청을 보고 받고 이를 지원하는 OpenGL API를 새롭게 정의한다. 이렇게 새롭게 정의되는 OpenGL API는 버전업을 거쳐, 보통 3월에 개최되는 GDC나 8월에 개최되는 Siggraph을 통해 대중에게 발표되고 있다.

3.2 그래픽스 드라이버

시중에 출시된 다양한 종류의 그래픽카드들은 발매일, 가격대 별로 제공하는 하드웨어적인 성능 차이가 있다. 예를 들어, 최근 출시된 nVidia의 GeForce GTX980 그래픽카드는 OpenGL 4.4를 지원하지만 몇년 전 출시된 nVidia의 GeForce GTX580 그래픽카드는 OpenGL 4.2를 지원한다. 그런데 내가 가지고 있는 PC에 GTX980이나 GTX580 중 어떤 그래픽카드를 꽂아 쓰더라도 모두 동일한 방식으로 최신 nVidia의 그래픽스 드라이버를 설치하면, GTX980의 경우는 OpenGL 4.4까지 활용할 수 있지만 GTX580의 경우는 OpenGL 4.2까지만 활용할 수 있다. 도대체 왜 이런 현상이 생기는 걸까?

그래픽스 드라이버는 하드웨어 가속기능을 구동할 수 있는 그래픽스 API 함수들의 집합체로 볼 수 있다. 다만 현재 내 PC에 탑제되어 있는 하드웨어의 한계로 인해 특정 그래픽스 API 함수를 지원하지 못한다면, 해당 함수가 불려도 아무런 동작이 되지 않는 것이 그래픽스 드라이버의 특징이다.

3.3 GLEW를 이용한 모던 OpenGL 개발환경 구축

기본적으로 OpenGL API는 고정 렌더링 파이프라인(fixed rendering pipeline)을 지원하는 OpenGL 1.x만 지원한다. 모던 OpenGL를 기반으로 개발하기 위해서는 OpenGL 1.x 이후에 새롭게 추가된 함수들을 활용해야 하는데, GLEW는 이를 손쉽게 해주기 위해 만들어진 공개 라이브러리이다.

OpenGL은 새롭게 추가되는 API 함수를 확장함수(extension function)로 관리함으로써 지속적인 기능 확장을 지원한다. OpenGL 확장함수를 관리하는 방법은 다름 아니라, 자신의 시스템에 설치된 그래픽스 드라이버에 지원하는 확장함수의 함수포인터(function pointer)를 얻어와서 활용하는 것이다. 그런데 확장함수를 얻는 방법이 운영체제마다 조금씩 다르고, OpenGL이 지속적으로 발전되는 동안 확장함수의 수가 굉장히 많아졌기 때문에, 프로그래머가 일일이 확장함수에 대한 함수포인터를 얻어와 프로그래밍하는 일이 매우 귀찮은 작업이 되었다.

GLEW는 OpenGL 1.x 이후에 추가된 확장함수 활용에 필요한 귀찮은 작업들을 대신해 줌으로써, 프로그래머가 모던 OpenGL 기반 개발에 집중할 수 있도록 도움을 주는 라이브러리이다. GLEW와 유사한 일을 하는 라이브러리로는 GLee등이 있다.

국민대학교 소프트웨어학부, junho@kookmin.ac.kr

3.4 GLEW 개발환경 설치

Ubuntu 16.04 LTS에서 다음과 같이 apt-get을 이용하여 GLEW 기반 개발환경을 손쉽게 설치할 수 있다.

```
sudo apt-get install libglew-dev
```

GLEW 개발환경 설치가 끝나면 Ubuntu 16.04 LTS는 내부적으로 크게 다음의 두가지 내용이 업데이트 된다.

- /usr/include/GL/ 아래에 있는 헤더 파일들
- /usr/lib/x86_64-linux-gnu/ 아래에 있는 라이브러리 파일들 (* 64-bit Ubuntu 16.04 LTS의 경우)

GLEW를 깔아서 생기는 헤더 파일과 라이브러리 파일은 이후, g++ 컴파일러를 이용해서 모던 OpenGL을 지원하는 프로그램을 생성하고자 할 때 활용될 것이다.

4 HELLO WORLD 작성

OpenGL 개발환경과 FreeGLUT, GLEW 개발환경의 설치가 끝났으면, 간단한 모던 OpenGL 기반 프로그램을 작성해 보자.

4.1 소스코드 작성

에디터를 통해 C++ 파일(예: hello_modern_GL.cpp)을 Fig. 1과 같이 작성한다.

4.2 소스컴파일 및 실행

터미널에서 g++ 컴파일러를 이용해서 다음과 같이 hello_world.cpp 파일을 컴파일 해 보자.

```
g++ hello_modern_GL.cpp -o hello -lglut -lGL -lGLEW
```

컴파일이 성공적으로 끝나면 현재 디렉토리에 실행가능한 파일인 hello가 생성된다. 만일 컴파일 에러가 생겼다면 C++ 소스파일에 오타가 있거나 g++ 컴파일 옵션에 오타가 있는 경우이다. 특히 대소문자를 구분하니 오타가 있는지 여부를 잘 살펴보도록 하자.

컴파일 후 만들어진 실행파일을 다음과 같이 터미널에서 실행시키면 Fig. 2와 같은 화면이 뜬다.

```
./hello &
```

여기까지 성공적으로 실행되었다면, CentOS 7에서 정상적으로 OpenGL 기반 개발환경이 구축된 것이다.

4.2.1 컴파일 과정 자세히 들어다보기

g++ 컴파일 옵션을 하나하나 살펴보면 다음과 같다.

- g++: C++ 컴파일러로 g++를 이용함
- hello_world.cpp: 컴파일할 소스파일 이름 지정
- -o hello: 컴파일 후 만들어질 실행가능한 파일 이름을 hello로 설정
- -lglut: FreeGLUT 라이브러리 파일을 찾아 링크하도록 함
- -lGL: OpenGL 라이브러리 파일을 찾아 링크하도록 함
- -lGLEW: GLEW 라이브러리 파일을 찾아 링크하도록 함

만일 소스코드가 수정되어 실행가능한 파일을 새로 만드려면 동일한 g++ 컴파일 과정을 거쳐야 한다. 일반적으로 g++ 컴파일 옵션이 매우 길기 때문에 이를 매번 타이핑하는 작업이 여간 귀찮은게 아니다. 동일한 디렉토리에 다음과 같이 Makefile을 작성해서 컴파일을 간단히 해보자.

```
all:
	g++ hello_modern_GL.cpp -o hello -lglut -lGL -lGLEW
```

Makefile을 위와 같이 작성한 후, 터미널에서 make라고 간단히 명령을 내리면 컴파일이 진행된다.

```
make
```

4.3 소스파일 자세히 들어다보기

소스파일 hello_modern_GL.cpp 중 대부분은 'FreeGLUT 기반 OpenGL 개발환경 구축하기' 내용과 거의 다를바가 없다. 다만 GLEW를 이용해서 OpenGL 2.x 버전 이상의 모던 OpenGL을 쓸 수 있도록 설정했다는 점에 주목하자.

4.3.1 주목해야할 GLEW 함수들

- glewInit(...): OpenGL 2.x 이후 제공되는 각종 함수들이 호출되게 함으로써, 모던 OpenGL 기반 프로그래밍 초기화.

```cpp
// hello_modern_GL.cpp
#include <GL/glew.h>      // GLEW header file
#include <GL/freeglut.h>
#include <iostream>
#include <cassert>        // assert(...)

void init();
void mydisplay();

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutCreateWindow("Hello OpenGL 2.x");

  init();
  glutDisplayFunc(mydisplay);
  glutMainLoop();

  return 0;
}


void init()
{
  GLenum res = glewInit();     // GLEW initialize
  if (res != GLEW_OK)
  {
    std::cerr << "Cannot initialize GLEW" << std::endl;
    assert(0);
  }

  glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
}

void mydisplay()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glutSwapBuffers();
}
```

Fig. 1. hello_modern_GL.cpp 소스파일

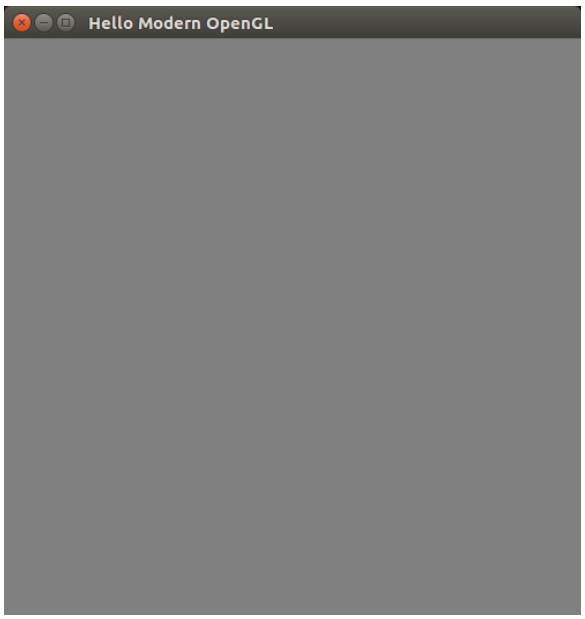

Fig. 2. FreeGLUT와 GLEW를 이용해 작성한 modern OpenGL 프로그램