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 오픈소스 라이브러리인 GLEW와 FreeGLUT를 이용하여 모던 OpenGL 기반 그래픽스 개발환경을 구축한다.

1 모던 OpenGL(modern OpenGL)

OpenGL은 산업계 표준 삼차원 그래픽스 API 중 하나이다. OpenGL API는 플랫폼 독립적으로 설계되어 있어, OpenGL로 구성한 그래픽스 어플리케이션은 다양한 운영체제에서 특별한 수정없이 돌아갈 수 있다.

OpenGL 1.x로 대표되는 기존 OpenGL(legacy OpenGL)은 그래픽스 하드웨어의 주어진 기능을 활용하는데 초점이 맞춰져 있었다. 따라서, 기존 OpenGL은 하드웨어 설계로 인해 고정된 기능만 사용할 수 있는 고정 렌더링 파이프라인(fixed rendering pipeline)만 지원하였다.

OpenGL 2.x 이후의 모던 OpenGL(modern OpenGL)은 그래픽스 하드웨어의 기능을 프로그래밍적 접근을 통해 활용하는데 초점이 맞춰져 있다. 따라서, 모던 OpenGL은 하드웨어 설계의 제약을 일부 벗어나 프로그래머가 원하는 기능을 쉐이더(shader)를 통해 프로그래밍할 수 있는 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 GLEW와 FreeGLUT를 설치하는 방법을 학습한다.

2 모던 OpenGL 개발환경 설치

Ubuntu 16.04 LTS에서 다음과 같이 apt을 이용하여 C++ 개발환경, GLEW, FreeGLUT등을 설치하여 모던 OpenGL 개발환경을 손쉽게 구축할 수 있다.

```
$ sudo apt install build-essential     # C/C++ 개발환경 설치
$ sudo apt install git curl            # git 및 curl 설치

$ sudo apt install libglew-dev         # GLEW 라이브러리
$ sudo apt install freeglut3-dev       # FreeGLUT 라이브러리
```

GLEW와 FreeGLUT를 설치하여 모던 OpenGL 개발환경 설치를 끝내면 GLEW와 FreeGLUT관련 헤더파일과 라이브러리 파일이 Ubuntu 16.04 LTS시스템에 깔리게 된다.

- GLEW와 FreeGLUT헤더 파일: /usr/include/GL/ 디렉토리 밑에 GLEW와 FreeGLUT관련 헤더 파일들 추가됨
- GLEW와 FreeGLUT라이브러리 파일: /usr/lib/x86_64-linux-gnu/ 디렉토리 밑에 GLEW와 FreeGLUT관련 라이브러리 파일 추가됨

GLEW와 FreeGLUT를 깔아 생기는 헤더 파일과 라이브러리 파일은 이후 g++ 컴파일러를 이용해서 모던 OpenGL 기반 프로그램 작성 시 활용된다.

3 Hello World 작성

개발환경 설치가 끝났으면, 간단한 모던 OpenGL 기반 프로그램을 작성해 보자.

3.1 소스코드 작성

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

국민대학교 소프트웨어학부, junho@kookmin.ac.kr

```
///// main.cpp
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <iostream>

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");

        init();

        glutDisplayFunc(mydisplay);

        glutMainLoop();

        return        0;
}

void init()
{
        if (glewInit() != GLEW_OK)
                std::cout << "GLEW Init Error!" << std::endl;

        std::cout << glGetString(GL_VERSION) << std::endl;

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

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

        glutSwapBuffers();
}
```

Fig. 1. main.cpp 소스파일

3.2 소스컴파일 및 실행

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

```
$ g++ main.cpp -o hello -lGL -lGLEW -lglut
```

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, Ubuntu 16.04 LTS에서 정상적으로 모던 OpenGL 기반 개발환경이 구축된 것이다.

3.3 컴파일 과정 자세히 들어다보기

모던 OpenGL로 작성된 프로그램의 컴파일 옵션을 하나하나 살펴보면 다음과 같다.

- g++: C++ 컴파일러로 g++를 이용함

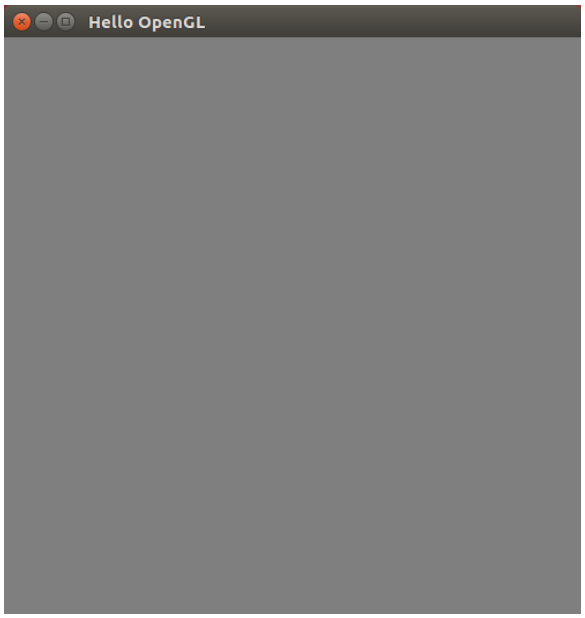

Fig. 2. 모던 OpenGL로 작성된 hello 프로그램

- main.cpp: 컴파일할 소스파일 이름 지정
- -o hello: 컴파일 후 만들어질 실행가능한 파일 이름을 hello로 설정
- -lGL: OpenGL 라이브러리 파일을 찾아 링크하도록 함
- -lGLEW: GLEW라이브러리 파일을 찾아 링크하도록 함
- -lglut: FreeGLUT라이브러리 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. Fig. 3과 같이 Makefile을 main.cpp가 있는 동일 디렉토리에 작성해서 컴파일 과정을 단순화 시키자.

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

Fig. 3. Makefile 파일

Makefile을 Fig. 3과 같이 작성한 후, 터미널에서 make라고 간단히 명령을 내리면 컴파일이 진행된다.

```
$ make
```

4 연습문제

소스파일 main.cpp를 수정해서 프로그램을 다음과 같이 변형해 보자.

1) 윈도우 크기를 800 x 800으로 바꿔보자.
2) 윈도우 타이틀 바에 'Hello OpenGL'가 아닌 자신의 영문이름이 찍히도록 바꿔보자.
3) 윈도우 내부색을 회색에서 흰색으로 바꿔보자.