

Chapter 02 - 동명대학교 게임공학관 3D 그래픽스 프로그래밍

OpenGL 소개와 구현환경 구축



“컴퓨터는 믿을 수 없이 빠르고, 정확하고, 명청한데, 인간은 믿을 수 없이 느리고, 부정확하고, 영리하다. 그래서 둘이 함께라면 상상을 뛰어넘을 정도로 강력하다.”
“Computers are incredibly fast, accurate and stupid; humans are incredibly slow, inaccurate and brilliant; together they are powerful beyond imagination.”

– 알베르트 아인슈타인 Albert Einstein

이 장에서 생각할 문제

- ❖ OpenGL에 대한 기본적인 이해
- ❖ OpenGL 프로그래밍을 위한 환경 구축
- ❖ 윈도우 환경에서 동작하는 간단한 OpenGL 프로그램의 구현

2.1 OpenGL 소개

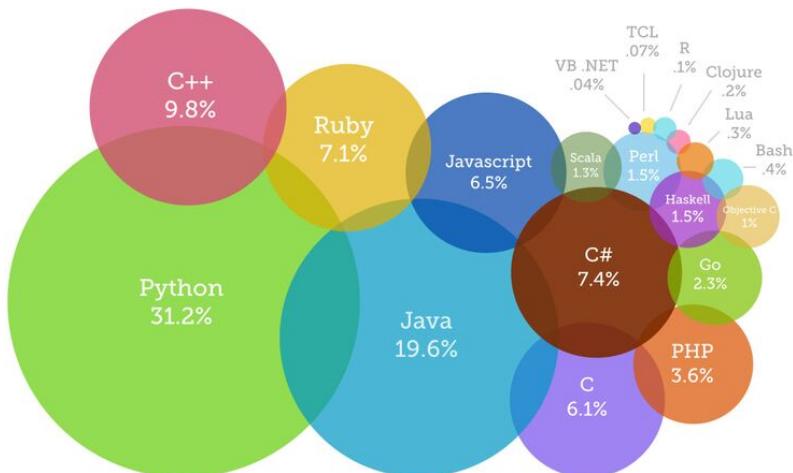
OpenGL은 특정한 하드웨어나 운영체제에 의존하지 않고 다양한 시스템에 이식^{porting}될 수 있는 개방형 라이브러리로 그래픽 처리 장치^{GPU}를 이용하여 기하 객체를 다루고 시각 정보로 변환하여 화면 등에 표시할 수 있도록 한다.

OpenGL은 실시간 그래픽스 환경에서 콘텐츠를 생성하는 것을 주요 목적으로 하고 있기 때문에 실사 수준의 고품질 렌더링보다는 빠르고 효율적인 방법으로 그래픽 정보를 생성하는 것을 목표로 한다. 그러나 최근의 하드웨어 발전이 가속함에 따라 매우 높은 수준의 사실성을 가진 영상을 실시간에 생성해 낼 수 있게 되었다.

이 강의에서는 이 OpenGL의 동작 원리를 이해하고, 활용할 수 있는 능력을 습득하여 향후 다양한 그래픽스 시스템이나 도구^{tool}을 다룰 때에 빠르게 적응하고 다룰 수 있는 기초를 닦는 것이다.

2.2 이 강의의 프로그래밍 환경

이 수업에서는 파이썬^{python}을 기본 프로그래밍 언어로 사용할 것이다. 세상에는 다양한 프로그래밍 언어가 존재하고, 프로그래머들은 각자 선호하는 언어가 있다. 대표적인 언어는 '파이썬', '자바', 'C', 'C++', 'JavaScript' 등이 있다. 아래 그림은 2019년 기준으로 가장 트렌디한 프로그래밍 언어로 선정된 언어들의 비중을 비교하고 있다.



2019년 가장 트렌디한 프로그래밍 언어, 자료출처: <https://learnworthy.net/>

파이썬 Python 은 **귀도 반 로섬** Guido van Rossum 이 1991년에 개발한 대화형 프로그래밍 언어로 최근 많은 인기를 얻고 있다. 반 로섬은 취미로 파이썬을 만들었지만 파이썬은 아주 인기 있는 범용 프로그래밍 언어가 되었다. 파이썬이 최근 전 세계적으로 각광을 받는 이유는 무엇일까? 가장 큰 이유는 파이썬의 생산성이 뛰어나기 때문이다. 파이썬을 이용하면 간결하면서도 효율적인 프로그램을 빠르게 작성할 수 있다. 파이썬은 오픈 소스여서 무료이고 패키지들이 계속 추가되고 있어서 매일 진화하는 언어이기도 하다. 예를 들어서 회사의 매출을 세련된 차트 chart 로 만들고 싶다면 맷플롯립 matplotlib 모듈을 가져다가 사용하면 된다.

파이썬은 무엇보다도 초보자의 프로그래밍 입문에 적합한 언어이다. 그 이유는 파이썬이 인터프리터 언어 interpreted language 이기 때문이다. 파이썬에서는 프로그래머가 한 줄의 문장을 입력하고 엔터키를 치면 명령 해석기인 인터프리터 interpreter 가 이것을 바로 실행한다. 다른 많은 언어는 실행하기 전에 코드를 컴퓨터가 이해할 수 있는 기계어로 번역하는 컴파일 과정이 필요한 경우가 많다. 그러나 파이썬 프로그래머는 자신이 작성한 문장의 결과를 입력 즉시 볼 수 있기 때문에 입문자도 간편하게 프로그램의 실행을 살펴볼 수 있다.

2.2.1 파이썬 프로그래밍 환경 구축하기

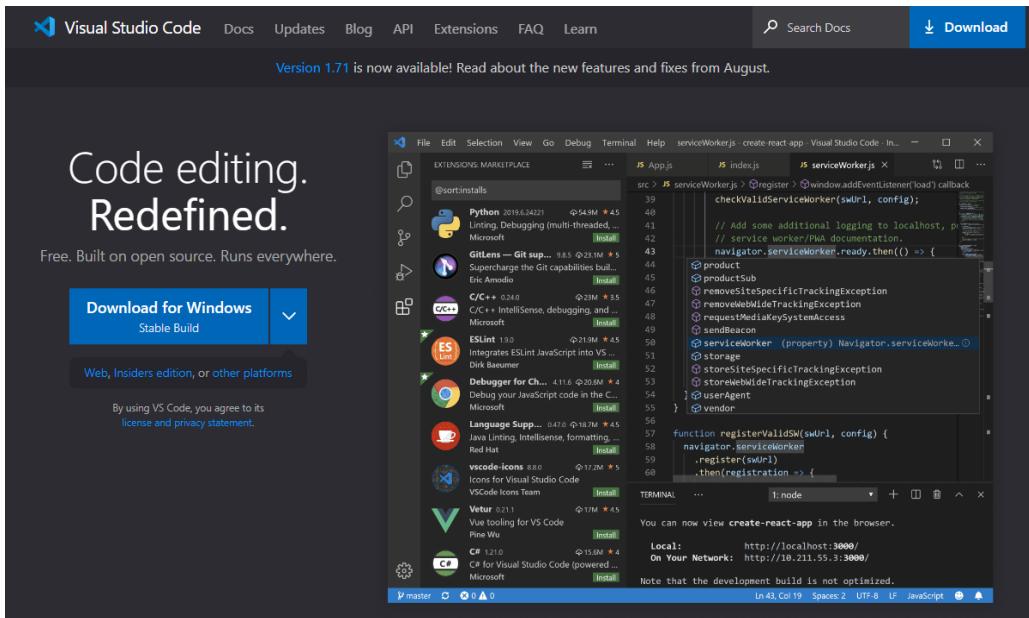
파이썬 프로그래밍 환경은 다양한 방법으로 구축할 수 있다. 여기서는 마이크로소프트 사에서 제공하는 비주얼 스튜디오 코드 Visual Studio Code 를 이용하여 파이썬 프로그래밍을 할 수 있는 환경을 구축해 보자.

비주얼 스튜디오 코드는 소스코드를 편집할 수 있는 편집기로 그 자체로는 프로그램 언어를 컴파일 compile 하거나 인터프리터 interpreter 로서의 기능을 제공하지는 않는다. 하지만 이 환경 내에서 파이썬이나 C, C++과 같은 언어를 사용할 수 있도록 하는 확장 extension 을 설치하면 다양한 언어를 쉽게 편집하고 실행할 수 있다.

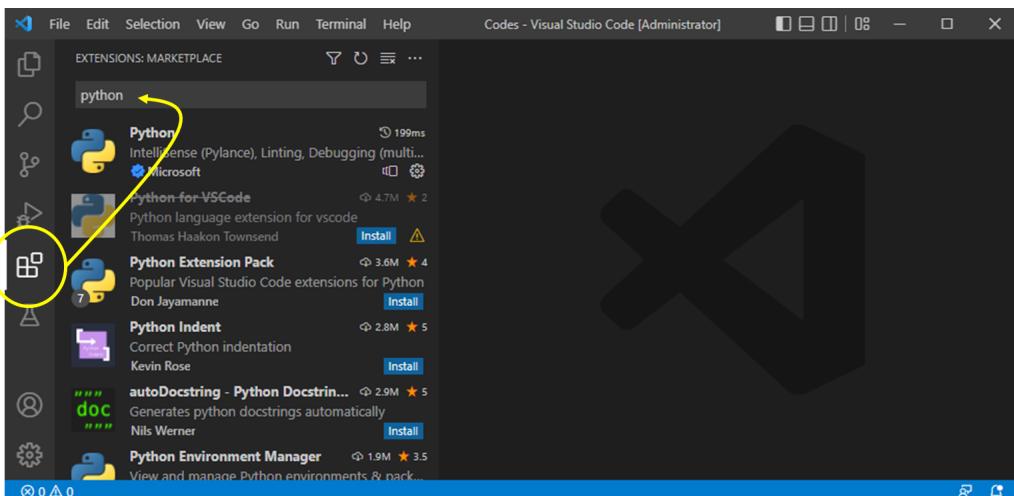
비주얼 스튜디오 코드를 설치하기 위해서는 공식 홈페이지에 접속하여 다운로드하면 된다. 비주얼 스튜디오 코드는 무료이며 공개 소프트웨어이다. 공식 홈페이지의 URL은 다음과 같다.

<https://code.visualstudio.com>

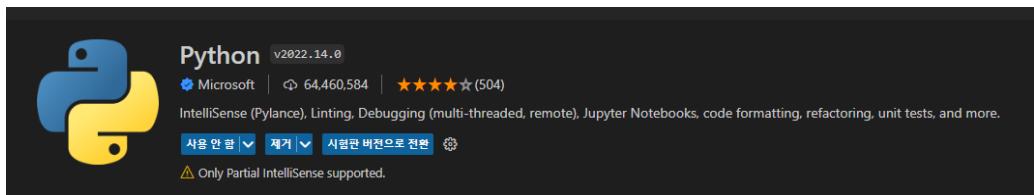
이 사이트에 접속하면 다음과 같은 화면이 나타난다.



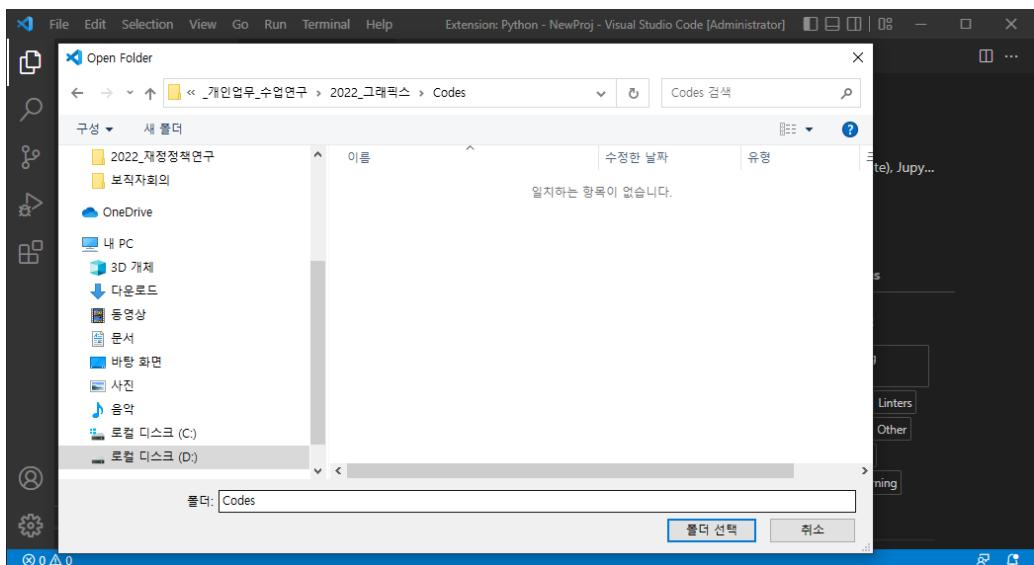
파이썬이 설치되고 나면 이제 파이썬을 사용할 수 있는 환경을 구현해 보자. 이를 위해서는 확장(extension)을 설치해야 한다. 화면의 왼쪽에 보면 확장을 표시하는 아이콘이 있다. 이를 선택하거나 Ctrl+Shift+x 키를 이용하여 확장을 검색할 수 있다. 검색창에 python을 입력하고 마이크로소프트에서 제공하는 Python 확장을 설치해 보자.



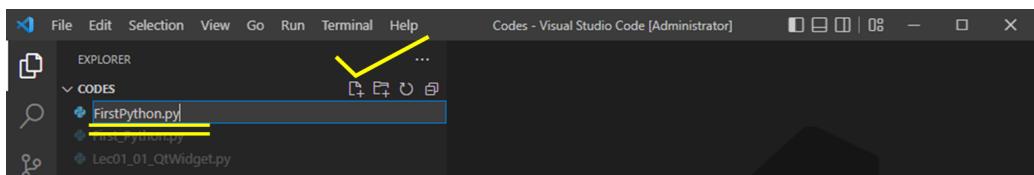
설치가 완료되면 다음과 같이 설치가 되었다는 표시가 나타날 것이다.



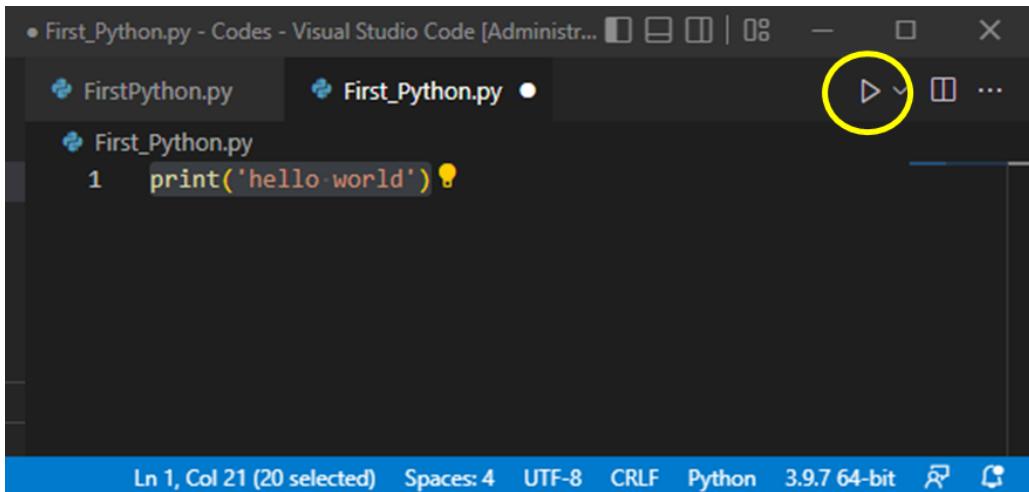
이제 새로운 폴더를 만들어 파이썬 프로그래밍을 시작해 보자. File 메뉴의 Open Folder를 선택하여 아래와 같이 원하는 작업 공간을 열도록 하자.



폴더 안에 First_Python.py라는 파일을 만들어 보았다.



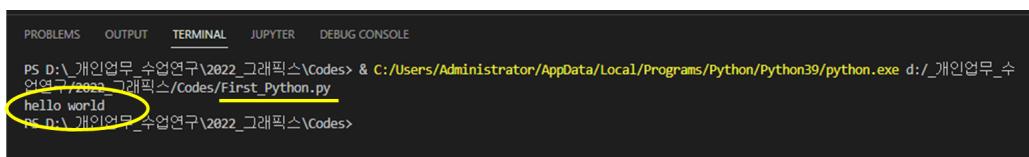
이 파일에 파이썬 코드를 작성하고 실행해 보자.



```
• First_Python.py - Codes - Visual Studio Code [Administrator] ⚡ ⚡ ⚡ | 0: - ⚡ ⚡ ⚡
  FirstPython.py First_Python.py •
  First_Python.py
  1   print('hello world') ⚡
```

Ln 1, Col 21 (20 selected) Spaces: 4 UTF-8 CRLF Python 3.9.7 64-bit ⚡ ⚡

터미널 창이 생성되고 실행 결과를 확인할 수 있을 것이다.



```
PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
PS D:\_개인업무_수업연구\2022_그래픽스\Codes> & C:/Users/Administrator/AppData/Local/Programs/Python/Python39/python.exe d:/_개인업무_수업연구\2022_그래픽스/codes/First_Python.py
hello world
PS D:\_개인업무_수업연구\2022_그래픽스\Codes>
```

이 터미널 창을 이용하여 바로 실행을 해 볼 수도 있다.



```
PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
PS D:\_개인업무_수업연구\2022_그래픽스\Codes> python .\First_Python.py
hello world
PS D:\_개인업무_수업연구\2022_그래픽스\Codes>
```

2.2.2 윈도우 생성 - Qt 사용하기

OpenGL은 그래픽 콘텐츠를 생성하므로, 문자가 아니라 이미지 정보를 생성한다. 이러한 이미지는 현재의 컴퓨터 운영체제에서 윈도우라는 객체 내에 표현된다. 우리가 사용하는 컴퓨터는 운영체제마다 각자의 윈도우 생성 시스템이 있다. 맥오에스^{MacOS}, 유닉스/리눅스^{Unix/Linux}, 마이크로소프트 윈도^{Windows} 등의 운영체제에서 윈도우 시스템을 생성하는 방식은 서로 다르다. 그러나 이렇게 각각의 운영체제에 따라 서로 다른 방식으로

윈도우를 생성하는 일을 하나의 프로그램 안에 다 담으려면 프로그래머가 여러 가지로 불편하다. 이를 해결하기 위해 운영체제에 독립적인 윈도우 생성 기술이 필요하다. 이러한 목적으로 다양한 도구를 사용할 수 있지만, 이 강의에서는 Qt를 사용할 예정이다.

Qt는 크로스플랫폼^{cross-platform} 그래픽 사용자 인터페이스 생성 소프트웨어로 다양한 운영체제에서 동작하는 윈도우 기반 응용 프로그램을 편리하게 생성할 수 있도록 한다. 최초 공개는 1995년에 이루어졌으며 지금도 다양한 분야에서 사용되고 있다. 현재 Qt 6.3 버전까지 나와 있다.

이 Qt를 이용하여 윈도우를 생성하고 이 윈도우에 OpenGL 그리기를 수행할 것이다. 이를 위해서는 파이썬용 Qt인 pyQt를 설치해야 하며, 파이썬용 OpenGL 패키지인 pyOpenGL도 설치해야 할 것이다.

비주얼 스튜디오 코드의 터미널 창에서 다음과 같이 pyQt의 버전 6.x를 설치해 보자.

pip를 이용한 pyQt6 설치

```
PS D:\_개인업무_수업연구\2022_그래픽스\Codes> pip install pyQt6
Collecting pyQt6
  Downloading PyQt6-6.3.1-cp37-abi3-win_amd64.whl (6.3 MB)
  ━━━━━━━━━━━━━━━━ 6.3/6.3 MB 5.9 MB/s eta
0:00:00
Collecting PyQt6-Qt6>=6.3.0
  Downloading PyQt6_Qt6-6.3.1-py3-none-win_amd64.whl (46.2 MB)
  ━━━━━━━━━━━━━━ 46.2/46.2 MB 8.5 MB/s eta
0:00:00
Collecting PyQt6-sip<14,>=13.4
  Downloading PyQt6_sip-13.4.0-cp39-cp39-win_amd64.whl (72 kB)
  ━━━━━━━━━ 72.9/72.9 kB 4.2 MB/s eta
0:00:00
Installing collected packages: PyQt6-Qt6, PyQt6-sip, pyQt6
Successfully installed PyQt6-Qt6-6.3.1 PyQt6-sip-13.4.0 pyQt6-6.3.1
PS D:\_개인업무_수업연구\2022_그래픽스\Codes>
```

pyQt6를 사용할 수 있는 상태가 되었다. 먼저 Qt를 이용하여 간단한 윈도를 생성해 보자. 아래 코드는 Qt를 이용하여 윈도우를 생성하는 예제이다.

Qt를 이용한 윈도우 생성 프로그램

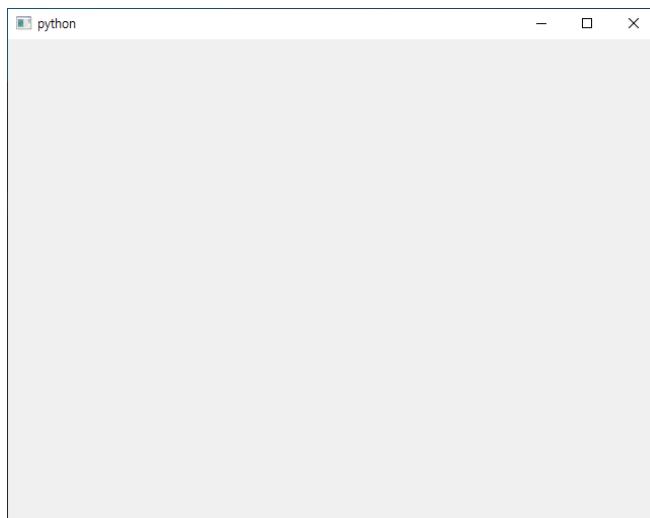
```
from PyQt6.QtWidgets import QApplication, QWidget

# 콘솔 명령으로 인자를 넘겨줄 때 필요 - 당분간 우리는 사용하지 않음
import sys

# 애플리케이션은 하나의 QApplication 인스턴스가 필요
# sys.argv에는 명령행에서 입력된 인자를 담고 있음 이를 QApplication 인자에 넘김
# 인자를 사용하지 않을 경우 sys.argv 대신 빈 리스트인 []를 넘길 수도 있음
# app = QApplication([])
app = QApplication(sys.argv)

# Qt 위젯을 생성 - 우리가 윈도우로 사용할 첫 위젯
window = QWidget()
window.show() # 기본적으로 화면에 나타나지 않게 설정되어 있어 보이게 변경

# 애플리케이션이 이벤트 루프에 들어가게 만든다.
app.exec()
```



윈도우에는 사용자와 상호작용하는 인터페이스 위젯^{widget}을 부착하여야 한다. 다음의 코드는 이를 위해 QApplication을 하나 생성하고, 윈도우는

QMainWindow를 상속받아 MainWindow라는 클래스로 새롭게 정의하고 있다. 이 클래스 안에는 레이블label과 라인 에디트line edit 위젯이 부착되는데, 수직vertical 배치를 위해 QVBoxLayout을 적용하는 코드이다. 코드를 살펴보면 쉽게 이해할 수 있을 것이다.

윈도우 내에 위치 배치하기

```
from PyQt6.QtWidgets import QApplication, QMainWindow
from PyQt6.QtWidgets import QLabel, QLineEdit, QVBoxLayout, QWidget
import sys

class MainWindow(QMainWindow):
    def __init__(self):
        # 슈퍼클래스의 초기화를 실행
        super().__init__()

        self.setWindowTitle("My App")

        # QLabel과 QLineEdit 클래스의 객체를 각각 생성한다
        self.label = QLabel()
        self.input = QLineEdit()

        # QLineEdit 객체의 값이 변경되면 QLabel로 전달된다.
        self.input.textChanged.connect(self.label.setText)

        # 생성해 놓은 두 위치 객체를 수직으로 배치할 레이아웃 객체 생성
        layout = QVBoxLayout()

        # 레이아웃 객체에 두 위치 객체를 추가하여 담기게 한다.
        layout.addWidget(self.label)
        layout.addWidget(self.input)

        # 윈도우에 표시될 위치를 담는 컨테이너 객체 생성
        container = QWidget()

        # 컨테이너에 레이아웃 객체를 지정한다. 그 안에 담긴 위치도 함께
        container.setLayout(layout)

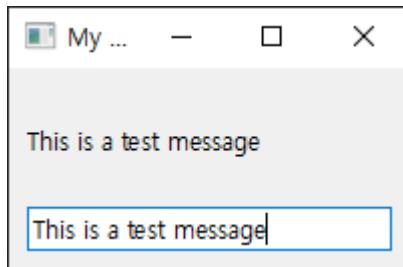
        # 윈도우의 중심 위치를 우리가 만든 것들이 담긴 컨테이너로 지정
        self.setCentralWidget(container)

app = QApplication(sys.argv)
```

```
window = MainWindow()
window.show()

app.exec()
```

이 코드를 실행하면 아래 그림과 같이 편집 위젯에 타이핑한 내용이 화면에 출력된다.



2.2.3 무턱대고 OpenGL 그리기를 시작해 보자.

OpenGL을 사용하기 위해서는 OpenGL 라이브러리를 설치하여야 한다. PyQt를 설치한 것과 같은 방법으로 다음과 같은 명령을 사용하면 된다.

```
$ pip install pyOpenGL
```

이제 OpenGL을 사용할 수 있는 환경이 되었다. 지금부터 작성할 코드는 main() 함수를 진입 지점으로 사용하는 C나 C++의 형태를 닮도록 고쳐보자. main() 함수 위에는 실행되는 코드를 두지 않고 정의만 두는 것을 기본으로 삼자. 그리고 main() 함수 다음에 이 코드가 다른 코드에 임포트^{import}되지 않고 인터프리터에 의해 직접 실행되었을 때에만 main() 함수를 호출하도록 하는 방식을 사용할 것이다. 이 코드가 인터프리터에서 직접 실행되었다면 __name__ 전역 변수가 '__main__'으로 지정된다. 따라서 아래 코드는 다른 코드에 임포트되지 않고 인터프리터가 바로 이

코드를 실행할 때에만 main() 함수가 호출되고, 이 함수 안에서 MyGLWindow 클래스의 객체를 생성하여 윈도우로 삽는다. 이때 앞의 코드는 QMainWindow를 상속받은 클래스를 사용했지만, 이 클래스는 QOpenGLWidget을 상속받아 사용한다. 이 위짓은 OpenGL 그리기가 가능한 위짓이다.

```
from OpenGL.GL import *
from OpenGL.GLU import *

from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget
from PyQt6.QtOpenGLWidgets import QOpenGLWidget

import sys

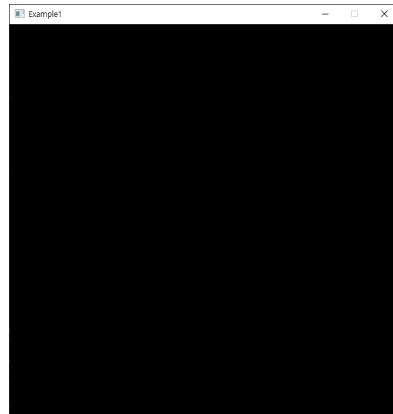
class MyGLWindow(QOpenGLWidget):

    def __init__(self, parent=None):
        super(MyGLWindow, self).__init__(parent)

    def main(argv = []):
        app = QApplication(argv)
        window = MyGLWindow()
        window.setWindowTitle('Example1')
        window.setFixedSize(600, 600)
        window.show()
        sys.exit(app.exec())

    if __name__ == '__main__':
        main(sys.argv)
```

실행을 하고 나면 다음과 같은 윈도가 생성된다. OpenGL 위짓에 아무런 코드도 담지 않았기에 아무런 그림도 그려지지 않는다.



OpenGL 위짓이 동작하도록 하는 데에는 3 가지 기본 콜백 함수가 필요하다. 그것은 다음과 같다.

- 초기화^{initialize} – OpenGL 콘텐츠의 초기화 콜백
- 크기 변경^{resize} – 윈도우가 생성되거나 크기가 변경될 때 호출되는 콜백
- 디스플레이^{display} – 화면에 그림을 그리는 일이 필요할 때 호출되는 콜백

이 세 가지 콜백을 OpenGL 위짓에 구현해 보도록 하자. 각각의 함수 명칭은 `initializeGL`, `resizeGL`, `paintGL`이라는 이름을 가진다. 그리고 OpenGL을 사용하기 위해서는 `pyOpenGL`에서 GL과 GLU를 사용할 수 있도록 해야 한다.

```
from OpenGL.GL import *
from OpenGL.GLU import *

from PyQt6.QtWidgets import QApplication, QWidget
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
import sys

#### MyWindow 클래스의 시작 #####
class MyGLWindow(QOpenGLWidget) : # QOpenGLWidget 상속

    def __init__(self):
        super().__init__() # 슈퍼클래스 QMainWindow 생성자 실행

        self.setWindowTitle('나의 첫 OpenGL 앱')

    def initializeGL(self) :
```

```

glClearColor(0.1, 0.7, 0.3, 1.0)

def resizeGL(self, w: int, h: int) :
    pass

def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    glBegin(GL_TRIANGLES)
    glColor3f(1, 1, 0)
    glVertex3f(-1, 0, 0)
    glVertex3f( 1, 0, 0)
    glVertex3f( 0, 1, 0)
    glColor3f(0, 1, 1)
    glVertex3f(-1, 0.5, 0)
    glVertex3f( 1, 0.5, 0)
    glVertex3f( 0,-0.5, 0)
    glEnd()

##### MyGLWindow 클래스의 끝 #####
#####

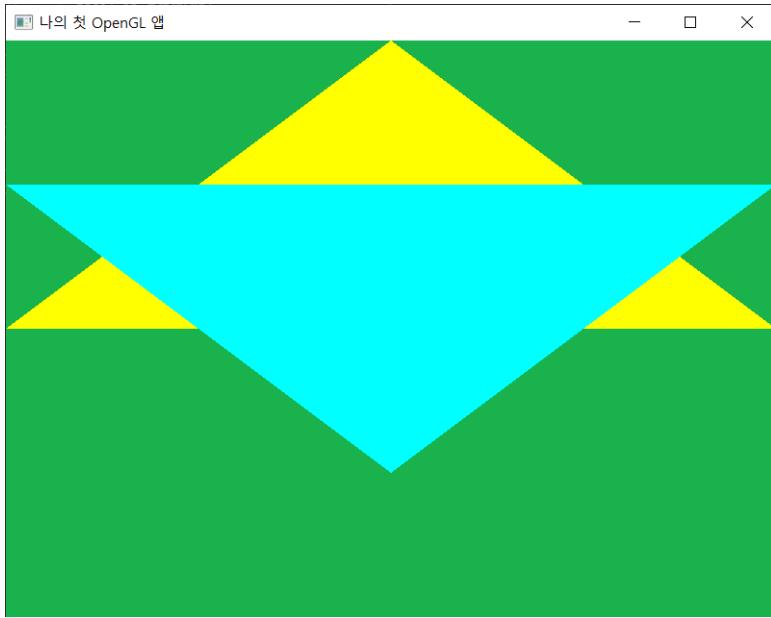
def main(argv = sys.argv) :
    ## 원도우 생성하기
    app = QApplication(argv)
    window = MyGLWindow()
    window.show()

    app.exec()

if __name__ == '__main__':
    main(sys.argv)

```

이렇게 세 가지 콜백 함수를 구현하고 실행해 보면, 아래와 같이 배경을 칠하고, 삼각형을 그리는 결과를 얻게 될 것이다. 이제 이러한 코드를 개선하여 OpenGL에 대한 이해를 점점 더 높여가도록 하자.



2.1 OpenGL의 특징과 관례

OpenGL은 실시간 그래픽 라이브러리로서 사실상 ^{de facto} 산업 표준이다. 이 라이브러리는 플랫폼에 독립적인 특징을 유지하려고 노력하고 있으며, OpenGL을 이용하여 작성한 그래픽 프로그램은 여러 종류의 다른 운영체제를 가진 시스템에 쉽게 이식할 수 있다.

OpenGL은 상태 기계 ^{state machine}으로서 OpenGL의 동작은 현재의 상태에 의해 결정된다. 이 상태는 변경하지 않는다면 계속해서 유지된다. OpenGL에는 많은 종류의 상태 변수가 있는데, 선의 굵기나 색부터 버퍼를 제어하는 변수까지 매우 다양하다. 이 상태는 한 번 설정하면 계속해서 설정된 상태를 유지하다가 변경 명령을 만나면 그 명령에 따라 다른 상태로 바뀐다. 예를 들어 선의 두께를 결정했다면 이후에 별도의 변경을 하지 않는 이상 선을 그릴 때는 계속해서 이 두께를 갖도록 그림을 그린다.

OpenGL의 API ^{application programming interface}는 `gl[ulut]-command-dimension-type`의 형태를 가진다. 예를 들어 `glVertex3f`는 `gl-Vertex-3-f`로 분리하여 이해할 수 있는데, `gl`은 이 명령이 `openGL`을 구성하는 GL, GLU, GLUT 등의 라이브러리 중에서 GL에 속하는 것이라는 의미하며, `Vertex`는 이 명령이 정점 ^{vertex}의 좌표를 지정하는 동작을 한다는 것을 의미하며, 이 명령의 정체성을 드러내는 것이다. 다음으로 3은 이 명령에서 사용하는 데이터의 차원을 의미하는 것으로 정점의 좌표를 3차원으로 표현할 것임을 나타낸다. 다음으로 f는 사용되는 데이터가 부동소수점 ^{floating point} 데이터라는 것을 의미한다. 파이썬은

자료형을 명시하지 않아도 자동으로 형 결정이 되지만, C나 C++을 이용하여 OpenGL 프로그래밍을 할 때는 이 자료형을 유의해서 다루어야 한다. OpenGL 명령어 마지막의 접미사가 나타내는 자료형에 대한 설명이 아래 표에 정리되어 있다.

접미사	자료형 명칭	C/C++ 기본 자료형	OpenGL 자료형
f	32 비트 부동소수점	float	GLfloat
d	64 비트 부동소수점	double	GLdouble
b	8 비트 정수	char	GLbyte
ub	8 비트 부호 없는 정수	unsigned char	GLubyte
i	32 비트 정수	int or long	GLint
ui	32 비트 부호 없는 정수	unsigned int long	GLuint, GLenum
s	16 비트 정수	short	GLshort

2.3 Qt Widget을 이용하여 OpenGL 렌더링 제어해 보기

```
from OpenGL.GL import *
from OpenGL.GLU import *

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QWidget
from PyQt6.QtOpenGLWidgets import QOpenGLWidget

from PyQt6.QtWidgets import QSlider ##### 
from PyQt6.QtCore import * #### Qt.orientation.Horizontal

import sys

#### MyWindow 클래스의 시작 #####
class MyGLWindow(QOpenGLWidget) : # QOpenGLWidget 상속

    def __init__(self):
```

```

super().__init__() # 슈퍼클래스 QOpenGLWidget 생성자 실행
self.r = 1.0
self.g = 0.0
self.b = 0.0

self.setWindowTitle('나의 첫 OpenGL 앱')

def initializeGL(self) :
    glClearColor(0.1, 0.7, 0.3, 1.0)

def resizeGL(self, w: int, h: int) :
    pass

def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    ##### ????? 아직은 모르지만 모든 OpenGL 코드가 이렇게 한다
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    glColor3f(self.r, self.g, self.b)
    glBegin(GL_TRIANGLES)
    glVertex3f(-1, 0, 0)
    glVertex3f( 1, 0, 0)
    glVertex3f( 0, 1, 0)
    glVertex3f(-1, 0.5, 0)
    glVertex3f( 1, 0.5, 0)
    glVertex3f( 0,-0.5, 0)
    glEnd()

def setR(self, val):
    self.r = val / 99
    self.update()

def setG(self, val):
    self.g = val / 99
    self.update()

def setB(self, val):
    self.b = val / 99
    self.update()
##### MyGLWindow 클래스의 끝 #####
class MyWindow(QMainWindow):

    def __init__(self) :
        super().__init__()

```

```
self.setWindowTitle('슬라이더로 색상 제어')

### Graphics User Interface 설정
centralWidget = QWidget()
self.setCentralWidget(centralWidget)

layout = QVBoxLayout()
centralWidget.setLayout(layout)

self.glWidget = MyGLWindow()
layout.addWidget(self.glWidget)

sliderR = QSlider(Qt.Orientation.Horizontal)
layout.addWidget(sliderR)
sliderG = QSlider(Qt.Orientation.Horizontal)
layout.addWidget(sliderG)
sliderB = QSlider(Qt.Orientation.Horizontal)
layout.addWidget(sliderB)

### 슬라이더 변경시에 연결된 동작 지정
sliderR.valueChanged.connect(self.glWidget.setR)
sliderG.valueChanged.connect(self.glWidget.setG)
sliderB.valueChanged.connect(self.glWidget.setB)

def main(argv = sys.argv) :
    ## 원도우 생성하기
    app = QApplication(argv)
    window = MyWindow()
    window.show()

    app.exec()

if __name__ == '__main__':
    main(sys.argv)
```

■ 슬라이더로 색상 제어

