

# Chapter 03 - 동명대학교 게임공학과 3D 그래픽스 프로그래밍

## 프리미티브



“나는 컴퓨터가 두렵지 않다. 그것이 없는 것이 두렵다.”

“I do not fear computers. I fear the lack of them”

— 아이작 아시모프 Isaac Asimov

### 이 장에서 생각할 문제

- ❖ OpenGL 프리미티브에 대한 이해
- ❖ 프리미티브를 변경하여 그리기 가능 연습해 보기
- ❖ 프리미티브 제어 프로그램 작성

### 3.1 프리미티브의 역할

프리미티브 primitive는 OpenGL이 제공하는 그리기 기본 요소인데, OpenGL에 제공되는 정점 데이터들을 어떻게 조합 assembly 할 것인지를 결정한다. 이 프리미티브를 사용하는 방법은 다음과 같다.

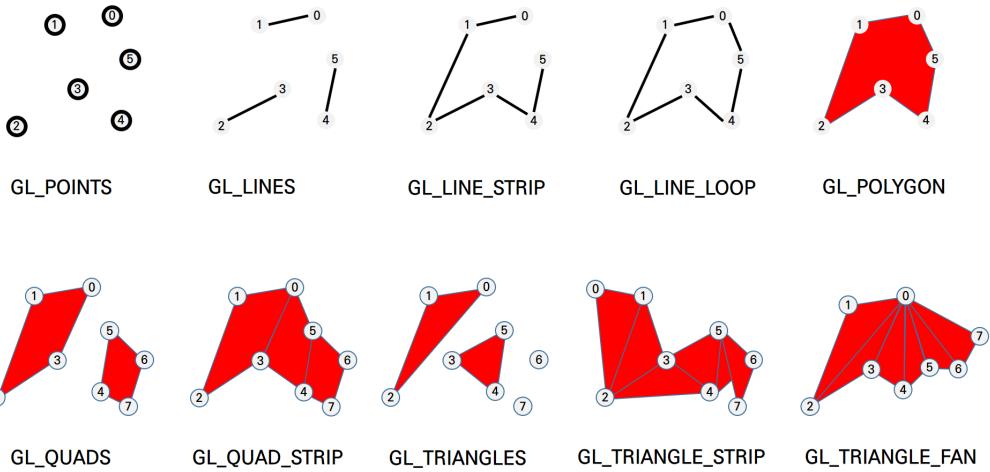
#### 프리미티브 사용 방법

```
glBegin(프리미티브 지정)  
    정점 정보 제공(glVertex3f 등을 이용)  
glEnd()
```

그래픽 하드웨어는 프로그래머가 지정한 프리미티브에 따라 정점의 리스트에서 정점을 몇 개 단위로 끊어서 그릴지 결정한다. 예를 들어 삼각형을 그리기 위해서는 GL\_TRIANGLES라는 프리미티브를 사용하는데, 이때는 입력된 정보를 읽고 정점 3 개씩 묶어 삼각형을 그린다. 프리미티브는 다음과 같은 것들이 존재한다.

- GL\_POINTS – 입력된 정점을 하나씩 점으로 가시화
- GL\_LINES – 입력된 정점을 두 개씩 묶어 선분으로 표현
- GL\_LINE\_STRIP – 입력된 정점을 차례로 연결하여 하나의 폴리라인 polyline 구성
- GL\_LINE\_LOOP – 입력된 정점을 차례로 연결한 뒤에 마지막 점을 시작점으로 연결
- GL\_TRIANGLES – 입력된 정점을 세 개씩 묶어 삼각형을 그림
- GL\_TRIANGLE\_STRIP – 처음 세 개 정점으로 삼각형을 그린 뒤, 정점이 추가될 때마다 삼각형을 직전 두 개 정점과 연결하여 삼각형 추가
- GL\_TRIANGLE\_FAN – 부채 모양으로 삼각형을 추가해 나감
- GL\_QUADS – 정점 네 개씩을 묶어 사각형 그리기
- GL\_QUAD\_STRIP: 처음 네 개 정점으로 사각형 그리고, 이후 두 개씩 묶어 직전 두 개 정점과 함께 사각형 그리기
- GL\_POLYGON: 입력된 모든 정점으로 다각형을 그림

아래 그림은 정점이 순서를 가지고 제공되었을 때 각각의 프리미티브가 정점을 어떤 방식으로 조합 assembly 하여 그림을 그리는지를 보이고 있다.



### 3.2 정점 정보 제공 방법

프리미티브가 설정되면 이 프리미티브에 따라 그림을 그릴 때에 필요한 정점 정보가 제공되어야 한다. 정점 데이터는 앞으로 여러 종류를 살펴볼 것인데, 대표적인 것은 정점의 위치, 해당 정점에서 면의 법선<sup>normal vector</sup>, 그리고 색상 등이다. 정점의 위치를 입력하는 OpenGL 명령어는 `glVertex[차원|자료형]`이다. 예를 들어 3차원 정점의 각 성분을 부동소수점 표현으로 입력하고 싶다면 `glVertex3f(x, y, z)`와 같이 입력이 가능하다. 따라서 다음과 같은 다양한 방법으로 정점 데이터를 입력할 수 있다.

#### 정점 정보 제공 방법

```

ix, iy, iz = -1, 0, 0
fx, fy, fz = 1.0, 0.0, 0.0
fverts = [fx-1.0, fy+1.0, fz]

glBegin(GL_TRIANGLES)
glVertex3i(ix, iy, iz)
glVertex3f(fx, fy, fz)
glVertex3fv(fverts)
glEnd()

```

모두 3차원 좌표를 사용하여 차원을 표시하는 숫자가 3으로 되어 있다. 정수 좌표를 제공하기 위해 `glVertex3i`를 사용했고, 부동소수점으로 표시된 좌표를 다루기 위해서 `glVertex3f`를 사용한 것을 확인할 수 있을 것이다. 마지막으로 사용한 함수에는 v가 접미사로 붙어 있는데, 이것은 벡터vector를 의미하며, 정점 정보가 배열이나 리스트와 같이 묶음으로 제공될 때 사용할 수 있다. 3fv라는 접미사는 결국 3차원 데이터를 표현하는 부동소수점 데이터 3 개가 묶음으로 제공된다는 의미이며, 여기서는 파이썬의 리스트를 이용하여 좌표값이 전달되었다.

#### Qt의 OpenGL 위젯인 QOpenGLWidget을 이용하여 프리미티브와 정점 정보 사용하기

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

import sys

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

class MyGLWindow(QOpenGLWidget):

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

    def initializeGL(self):
        # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
        glClearColor(0.8, 0.8, 0.6, 1.0)

    def resizeGL(self, width, height):
        # 카메라의 투영 특성을 여기서 설정
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()

    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()
```

```

glColor3f(0, 1, 1)

ix, iy, iz = -1, 0, 0
fx, fy, fz = 1.0, 0.0, 0.0
fverts = [fx-1.0, fy+1.0, fz]

glBegin(GL_TRIANGLES)
 glVertex3i(ix, iy, iz)
 glVertex3f(fx, fy, fz)
 glVertex3fv(fverts)
 glEnd()

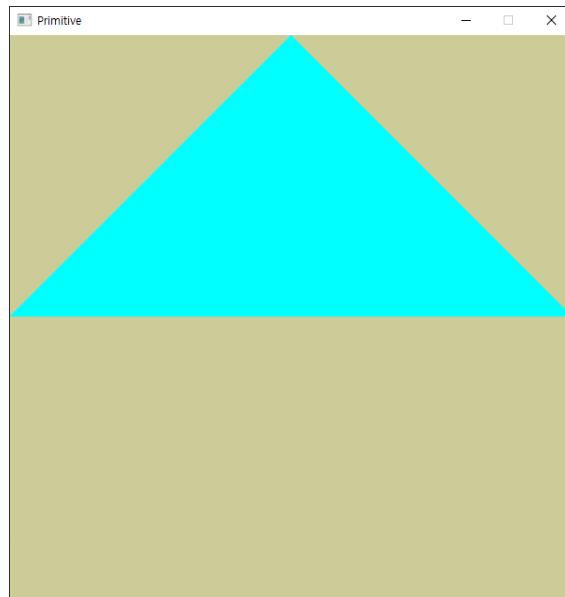
# 그려진 프레임버퍼를 화면으로 송출
glFlush()

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

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

```

이러한 방식으로 그리기 코드를 구현하여 실행하면 다음 그림과 같이 세 개의 정점을 가진 삼각형이 그려진다. 프리미티브로는 `GL_TRIANGLES`가 사용되었고, 이 프리미티브는 세 개의 정점을 모아 하나의 다각형을 그린다. 제공된 정점이 3 개이므로 이들을 이용하여 삼각형을 그리게 된다.



입력 즉시 볼 수 있기 때문에 입문자도 간편하게 프로그램의 실행을 살펴볼 수 있다.

### 3.3 정점 정보 제공 및 OpenGL 위치과 염계

임의의 정점을 입력하고, 이 정점을 다양한 프리미티브로 그림을 그려볼 수 있도록 해 보자.  
우선은 정점을 입력받는 프로그램을 작성해 보자.

#### 3.3.1 정점 입력을 처리하기 위한 기본 프로그램 작성

새롭게 사용할 클래스는 `QPainter`라는 클래스이다. 이 클래스는 위젯 위에 저수준 low-level 그리기 동작을 할 수 있게 하다. 우리의 예제에서는 마우스 클릭을 한 위치에 점을 그려 넣는 일을 할 것이다. 그리고 점들은 입력된 순서에 따라 선으로 이어지는데 이 연결선도 이 클래스를 이용하여 그릴 것이다. 그림을 그릴 때는 어떠한 방식으로 그릴지가 결정되어야 하는데 이를 결정하는 클래스가 `QPen`이다. 따라서 다음과 같이 새로운 클래스들을 `import` 한다.

```
from PyQt5.QtGui import QPainter, QPen
```

사용자가 입력한 정점은 프로그램 전체에서 POINTS라는 전역 변수에 담아 관리하자. 이 변수는 리스트인데, 하나의 원소는 2차원 좌표를 표현하는 리스트로 표현된다. 따라서 POINTS는 리스트의 리스트라고 할 수 있다. 기본적으로 세 개의 점은 입력된 것으로 시작할 것이다. 다음과 같이 POINTS를 초기화한다.

```
# 정점 데이터 (2차원 정점을 리스트로 표현하고, 이들의 리스트로 복수의 정점 관리)
POINTS = [[0, 0], [10, 10], [100, 50]]
```

하나의 윈도우에 수평 박스 레이아웃을 설정하고 왼쪽에는 정점을 초기화 하는 버튼을 두고, 오른쪽에는 정점 입력을 마우스로 받을 수 있는 위젯을 배치하려고 한다. 정점을 초기화하는 버튼이 있는 곳은 이 버튼 뿐만이 아니라 프리미티브를 선택하는 콤보 박스<sup>comboBox</sup> 등을 둘 것이므로 왼쪽에는 이러한 배치가 가능하도록 그룹 박스를 둘 것이다. 이것은 QGroupBox를 생성하고, 여기에 수직 박스 레이아웃을 단다. 이 레이아웃에는 여러 제어 위젯을 달 수 있는데, 우선 정점입력 초기화를 위한 버튼을 추가하는 형태로 구현해 보자. 다음과 같이 구현할 수 있다. 그리고 오른쪽에는 정점을 입력받고 그 정점들을 그리기 위한 Drawer 클래스의 객체인 위젯을 달자. Drawer 클래스는 우리가 만들 클래스로 QPainter 클래스의 객체를 멤버로 가진 클래스이다. 주요 윈도우가 될 MyWindow 클래스를 앞의 예제와 같이 만들고 다음과 같이 사용자 인터페이스를 구성할 수 있다.

```
#### GUI 설정
central_widget = QWidget()
self.setCentralWidget(central_widget)

gui_layout = QHBoxLayout() # CentralWidget의 수평 나열 레이아웃

# 배치될 것들 - 정점 입력을 받기 위한 위젯
central_widget.setLayout(gui_layout)

self.controlGroup = QGroupBox('정점 입력')
gui_layout.addWidget(self.controlGroup)

control_layout = QVBoxLayout()
self.controlGroup.setLayout(control_layout)

## 정점을 초기화하는 버튼을 추가. 버튼을 누르면 정점이 사라진다.
```

```

## 이 버튼을 누르면 resetPoints라는 이 메소드가 호출된다.
reset_button = QPushButton('정점 초기화', self)
reset_button.clicked.connect(self.resetPoints)

control_layout.addWidget(reset_button)

## 정점을 입력받기 위한 위젯을 만들고, pointInput이라는 멤버로 관리하자.
self.pointInput = Drawer(parent=self)
gui_layout.addWidget(self.pointInput)

```

입력 정점을 초기화하는 버튼이 눌리면 이 클래스의 `resetPoints` 메소드가 동작하도록 연결되어 있다. 이 함수는 다음과 같이 구현하였다. 구현 내용은 간단하다. `POINTS`라는 이름의 리스트 객체를 원소가 없는 리스트로 만들어 버린다. 그리고 나선 `self.pointer`를 호출한다.

```

# 정점 초기화 버튼을 눌렀을 때 호출되는 메소드
def resetPoints(self, btn):
    global POINTS
    POINTS = []
    self.pointInput.update()

```

`Drawer` 클래스는 `QPainter` 클래스를 가지고 있는 위짓이다. 따라서 다음과 같이 초기화 함수를 구현할 수 있다.

```

##### 정점 입력을 위해 사용되는 위짓으로 QPainter를 활용한다.
class Drawer(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.parent = parent
        # QPainter 멤버 준비
        self.painter = QPainter()

```

이 `Drawer` 클래스 객체는 두 가지 이벤트에 반응한다. 하나는 사용자가 마우스를 이용하여 점의 위치를 입력하는 이벤트이며, 다른 하나는 위짓으로서 그리기 동작을 수행해야 하는 이벤트이다.

위짓에 마우스가 눌리쳤을 때 해당 이벤트를 처리하는 처리기는 `mousePressEvent`라는 메소드를 구현하여 오버라이딩 overriding 할 수 있다. 우리는 마우스의 위치를 점의 위치로

보고, 이를 2개의 원소를 가진 리스트로 만들어 POINTS 리스트에 추가<sup>append</sup>하는 동작을 구현하면 된다.

```
# 정점을 입력하는 방법은 마우스 이벤트 발생시에 좌표를 읽어
# 이 정점을 표현하는 2차원 정보를 리스트로 만들어 정점 리스트 POINTS에 추가
def mousePressEvent(self, event):
    POINTS.append([event.x(), event.y()])
    print(event.x(), event.y())
    self.update()
```

이) Draw 클래스가 화면에 그려질 때는 QPainter 클래스의 객체가 동작하여 그림을 그리도록 하자. QPainter는 begin() 메소드와 end() 메소드가 호출되는 사이에 drawPoint, drawLine과 같은 메소드를 이용하여 기하 객체를 그릴 수 있는 클래스이다. 입력된 정점 POINTS를 출력하는 방법은 다음과 같이 구현할 수 있다.

```
# QPainter를 이용하여 입력된 정점을 출력한다.
def paintEvent(self, event):
    global POINTS

    self.painter.begin(self)
    self.painter.setPen(QPen(Qt.red, 6))

    for i in range(len(POINTS)):
        self.painter.drawPoint(POINTS[i][0], POINTS[i][1])

    self.painter.setPen(QPen(Qt.blue, 2))
    for i in range(len(POINTS) - 1):
        self.painter.drawLine(POINTS[i][0], POINTS[i][1],
                             POINTS[i + 1][0], POINTS[i + 1][1])
    self.painter.end()
```

기타 애플리케이션을 실행하는 코드를 추가하여 다음과 같이 완성할 수 있다. 이 프로그램은 사용자의 마우스 입력에 따라 정점을 추가하거나, 초기화하는 기능을 갖고 있다.

#### 정점을 입력받기 위한 사용자 인터페이스 구현

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout,
QHBoxLayout, QWidget
```

```

from PyQt5.QtWidgets import QGroupBox, QPushButton
from PyQt5.QtCore import *
from PyQt5.QtGui import QPainter, QPen
import sys
import numpy as np

# 정점 데이터 (2차원 정점을 리스트로 표현하고, 이들의 리스트로 복수의 정점 관리)
POINTS = [[0, 0], [10, 10], [100, 50]]


class MyWindow(QMainWindow):

    def __init__(self, title=''):
        QMainWindow.__init__(self) # QMainWindow 슈퍼 클래스의 초기화
        self.setWindowTitle(title)

        ### GUI 설정
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        gui_layout = QHBoxLayout() # CentralWidget의 수평 나열 레이아웃

        # 배치될 것들 - 정점 입력을 받기 위한 위치
        central_widget.setLayout(gui_layout)

        self.controlGroup = QGroupBox('정점 입력')
        gui_layout.addWidget(self.controlGroup)

        control_layout = QVBoxLayout()
        self.controlGroup.setLayout(control_layout)

        ## 정점을 초기화하는 버튼을 추가. 버튼을 누르면 정점이 사라진다.
        ## 이 버튼을 누르면 resetPoints라는 이 메소드가 호출된다.
        reset_button = QPushButton('정점 초기화', self)
        reset_button.clicked.connect(self.resetPoints)

        control_layout.addWidget(reset_button)

        ## 정점을 입력받기 위한 위치를 만들고, pointInput이라는 멤버로 관리하자.
        self.pointInput = Drawer(parent=self)
        gui_layout.addWidget(self.pointInput)

    # 정점 초기화 버튼을 눌렀을 때 호출되는 메소드
    def resetPoints(self, btn):
        global POINTS
        POINTS = []
        self.pointInput.update()

```

```

##### 정점 입력을 위해 사용되는 위짓으로 QPainter를 활용한다.
class Drawer(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.parent = parent
        # QPainter 멤버 준비
        self.painter = QPainter()

    # QPainter를 이용하여 입력된 정점을 출력한다.
    def paintEvent(self, event):
        global POINTS

        self.painter.begin(self)
        self.painter.setPen(QPen(Qt.red, 6))

        for i in range(len(POINTS)):
            self.painter.drawPoint(POINTS[i][0], POINTS[i][1])

        self.painter.setPen(QPen(Qt.blue, 2))
        for i in range(len(POINTS) - 1):
            self.painter.drawLine(POINTS[i][0], POINTS[i][1],
                                POINTS[i + 1][0], POINTS[i + 1][1])
        self.painter.end()

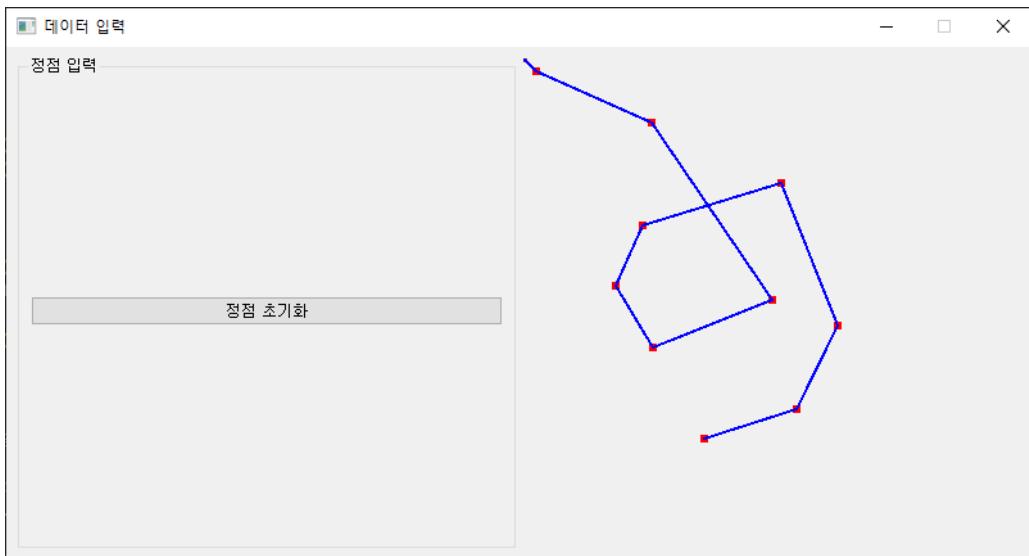
    # 정점을 입력하는 방법은 마우스 이벤트 발생시에 좌표를 읽어
    # 이 정점을 표현하는 2차원 정보를 리스트로 만들어 정점 리스트 POINTS에 추가
    def mousePressEvent(self, event):
        POINTS.append([event.x(), event.y()])
        print(event.x(), event.y())
        self.update()

def main(argv=[]):
    app = QApplication(argv)
    window = MyWindow('데이터 입력')
    window.setFixedSize(800, 400)
    window.show()
    sys.exit(app.exec_())

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

```

이 프로그램을 실행하면 아래와 같이 우측에 마우스 클릭을 하면 해당 위치의 좌표에 정점 데이터가 추가되고 입력된 데이터들을 화면에 보여주는 프로그램이 동작할 것이다.



### 3.3.2 OpenGL 위젯과 연결하여 각종 프리미티브 테스트하기

다음으로 우리가 할 일은 이 윈도에 OpenGL을 이용하여 그림을 그리는 위젯을 추가하고, 이 OpenGL 위젯 안에 입력된 정점을 이용하여 그림을 그리는 것이다. OpenGL의 기능을 사용하고, 이 기능의 결과가 표현될 수 있는 OpenGL 위젯을 사용하기 위해 `QOpenGLWidget`, 그리고 프리미티브를 선택할 수 있도록 `QComboBox`도 필요할 것이다. 따라서 다음과 같은 추가적인 임포트를 수행한다.

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

#### 추가로 필요한 패키지
from PyQt5.QtWidgets import QOpenGLWidget, QComboBox
```

목표로 하는 시스템은 콤보 박스를 이용하여 OpenGL 프리미티브를 선택하고, 이 프리미티브로 정점을 그리는 것이다. 이를 위해서 콤보 박스에 표현될 문자열의 리스트를 `PRIMITIVES`에 리스트로 담고, 선택된 프리미티브에 해당하는 실제 OpenGL

프리미티브를 담을 리스트를 PRIMITIVE\_VALUES라는 이름으로 준비한다. 그리고 선택된 프리미티브의 인덱스<sup>index</sup>는 selected라는 변수로 관리하자. 지금까지 준비한 모든 변수들을 전역 변수<sup>global variable</sup>이다.

```
##### 프리미터 선택을 위한 데이터
PRIMITIVES = ['GL_POINTS', 'GL_LINES', 'GL_LINE_STRIP', 'GL_LINE_LOOP',
               'GL_TRIANGLES', 'GL_TRIANGLE_STRIP', 'GL_TRIANGLE_FAN',
               'GL_QUADS', 'GL_QUAD_STRIP', 'GL_POLYGON']

PRIMITIVE_VALUES = [GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP,
                     GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN,
                     GL_QUADS, GL_QUAD_STRIP, GL_POLYGON]
selected = 0
```

이제 MyWindow 클래스 객체로 생성될 윈도우 객체에 붙을 OpenGL 위짓을 준비하자. 이 위짓은 MyGLWidget 클래스로 구현할 것이다. 이 내부에서 수행하는 동작은 선택된 프리미티브를 glBegin에 설정하고, 정점 정보를 제공하는 것이 전부이다. 다만 주의할 점은 resizeGL 이벤트 처리기 메소드에서 glOrtho라는 OpenGL 명령어를 사용하여 6 개의 숫자를 이용하여 무엇인가를 설정하는 작업이 앞의 OpenGL 구현과 차이가 있다. 이것은 다음 장에서 살펴볼 직교투영<sup>orthographic projection</sup>에서 관측 가능한 공간을 설정하는 것으로, 정점이 입력되는 공간의 윈도우 좌표계를 담을 수 있도록 설정되었다. 상세한 설명은 투영에서 다루기로 하자. 여기서는 마우스 입력이 이루어지는 공간과 비슷한 좌표계를 설정하는 것이라고 이해하면 된다. 아래와 같이 구현할 수 있을 것이다.

```
##### 정점 정보 그리기
class MyGLWidget(QOpenGLWidget):

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

    def initializeGL(self):
        # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
        glClearColor(0.8, 0.8, 0.6, 1.0)
        glPointSize(4)
        glLineWidth(2)

    def resizeGL(self, width, height):
        # 카메라의 투영 특성을 여기서 설정
        glMatrixMode(GL_PROJECTION)
```

```

glLoadIdentity()
glOrtho(0, 240, 380, 0, -1, 1)

def paintGL(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    # 프리미티브를 이용한 객체 그리기
    glBegin(PRIMITIVE_VALUES[selected])
    nPoints = len(POINTS)
    for i in range(nPoints):
        glVertex2fv(POINTS[i])
    glEnd()

    # 그려진 프레임버퍼를 화면으로 송출
    glFlush()

```

이제 이 위젯이 메인 윈도우에 포함되도록 하면된다. 다음과 같이 MyWindow 클래스 내에 우리가 구현한 OpenGL 위젯을 만들어 추가한다.

```

##### OpenGL Widget 추가
self.glWidget = MyGLWidget() # OpenGL Widget
gui_layout.addWidget(self.glWidget)

```

콤보 박스를 이용하여 프리미티브를 선택할 수 있도록 지원하는 위젯을 추가로 달아야 한다. 이것은 다음과 같은 방식으로 가능하다.

```

##### 프리미티브 선택 기능 추가
primitive_selection = QComboBox()
for i in range(len(PRIMITIVES)):
    primitive_selection.addItem(PRIMITIVES[i])

# ComboBox에 기능 연결
primitive_selection.currentIndexChanged.connect(
    self.selectPrimitive)

reset_button = QPushButton('reset vertices', self)
reset_button.clicked.connect(self.resetPoints)

control_layout.addWidget(primitive_selection)
control_layout.addWidget(reset_button)

```

위의 코드에는 프리미티브를 선택하는 콤보박스를 primitive\_selection이라는 이름으로 생성하였고, 이 콤보박스를 이용하여 프리미티브를 선택하면 selectPrimitive라는 메소드가 변경을 처리하도록 하고 있다. 이 메소드는 다음과 같이 구현된다.

```
##### Primitive 선택 #####
def selectPrimitive(self, text):
    global selected
    selected = int(text)
    self.glWidget.update()
```

추가적으로 마우스 입력이 있을 때 OpenGL 위젯에 변화가 반영되어 새로운 그림이 그려질 수 있도록 update 함수를 호출하는 것이 필요하다.

```
def mousePressEvent(self, event):
    POINTS.append([event.x(), event.y()])
    print(event.x(), event.y())
    #####
    self.parent.glWidget.update()
    #####
```

이러한 내용들을 종합하여 다음과 같은 코드가 완성된다.

#### 정점을 입력받기 위한 사용자 인터페이스 구현

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

#### 추가로 필요한 패키지
from PyQt5.QtWidgets import QOpenGLWidget, QComboBox
#####

from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout,
QHBoxLayout, QWidget
from PyQt5.QtWidgets import QGroupBox, QPushButton
from PyQt5.QtCore import *
from PyQt5.QtGui import QPainter, QPen
import sys
```

```

import numpy as np

# 정점 데이터 (2차원 정점을 리스트로 표현하고, 이들의 리스트로 복수의 정점 관리)
POINTS = [[0, 0], [10, 10], [100, 50]]

##### 프리미터 선택을 위한 데이터
PRIMITIVES = ['GL_POINTS', 'GL_LINES', 'GL_LINE_STRIP', 'GL_LINE_LOOP',
               'GL_TRIANGLES', 'GL_TRIANGLE_STRIP', 'GL_TRIANGLE_FAN',
               'GL_QUADS', 'GL_QUAD_STRIP', 'GL_POLYGON']

PRIMITIVE_VALUES = [GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP,
                     GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN,
                     GL_QUADS, GL_QUAD_STRIP, GL_POLYGON]
selected = 0
#####

##### 정점 정보 그리기
class MyGLWidget(QOpenGLWidget):

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

    def initializeGL(self):
        # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
        glClearColor(0.8, 0.8, 0.6, 1.0)
        glPointSize(4)
        glLineWidth(2)

    def resizeGL(self, width, height):
        # 카메라의 투영 특성을 여기서 설정
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(0, 240, 380, 0, -1, 1)

    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()

        # 프리미티브를 이용한 객체 그리기
        glBegin(PRIMITIVE_VALUES[selected])
        nPoints = len(POINTS)
        for i in range(nPoints):
            glVertex2fv(POINTS[i])
        glEnd()

        # 그려진 프레임버퍼를 화면으로 송출
        glFlush()

```

```

#####
class MyWindow(QMainWindow):

    def __init__(self, title=''):
        QMainWindow.__init__(self) # QMainWindow 슈퍼 클래스의 초기화
        self.setWindowTitle(title)

        ### GUI 설정

        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        gui_layout = QHBoxLayout() # CentralWidget의 수평 나열 레이아웃

        # 배치될 것들 - 정점 입력을 받기 위한 위젯
        central_widget.setLayout(gui_layout)

        ##### OpenGL Widget 추가
        self.glWidget = MyGLWidget() # OpenGL Widget
        gui_layout.addWidget(self.glWidget)
#####

        self.controlGroup = QGroupBox('정점 입력')
        gui_layout.addWidget(self.controlGroup)

        control_layout = QVBoxLayout()
        self.controlGroup.setLayout(control_layout)

        ## 정점을 초기화하는 버튼을 추가. 버튼을 누르면 정점이 사라진다.
        ## 이 버튼을 누르면 resetPoints라는 이 메소드가 호출된다.
        reset_button = QPushButton('정점 초기화', self)
        reset_button.clicked.connect(self.resetPoints)

        ##### 프리미티브 선택 기능 추가
        primitive_selection = QComboBox()
        for i in range(len(PRIMITIVES)):
            primitive_selection.addItem(PRIMITIVES[i])

        # ComboBox에 기능 연결
        primitive_selection.currentIndexChanged.connect(
            self.selectPrimitive)

        reset_button = QPushButton('reset vertices', self)
        reset_button.clicked.connect(self.resetPoints)

        control_layout.addWidget(primitive_selection)

```

```

control_layout.addWidget(reset_button)
#####
## 정점을 입력받기 위한 위젯을 만들고, pointInput이라는 멤버로 관리하자.
self.pointInput = Drawer(parent=self)
gui_layout.addWidget(self.pointInput)

##### Primitive 선택 #####
def selectPrimitive(self, text):
    global selected
    selected = int(text)
    self.glWidget.update()
#####

# 정점 초기화 버튼을 눌렀을 때 호출되는 메소드
def resetPoints(self, btn):
    global POINTS
    POINTS = []
    self.pointInput.update()

##### 정점 입력을 위해 사용되는 위젯으로 QPainter를 활용한다.
class Drawer(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.parent = parent
        # QPainter 멤버 준비
        self.painter = QPainter()

    # QPainter를 이용하여 입력된 정점을 출력한다.
    def paintEvent(self, event):
        global POINTS

        self.painter.begin(self)
        self.painter.setPen(QPen(Qt.red, 6))

        for i in range(len(POINTS)):
            self.painter.drawPoint(POINTS[i][0], POINTS[i][1])

        self.painter.setPen(QPen(Qt.blue, 2))
        for i in range(len(POINTS) - 1):
            self.painter.drawLine(POINTS[i][0], POINTS[i][1], POINTS[i + 1][0], POINTS[i + 1][1])
        self.painter.end()

    # 정점을 입력하는 방법은 마우스 이벤트 발생시에 좌표를 읽어
    # 이 정점을 표현하는 2차원 정보를 리스트로 만들어 정점 리스트 POINTS에 추가
    def mousePressEvent(self, event):

```

```

POINTS.append([event.x(), event.y()])
print(event.x(), event.y())
#####
self.parent.glWidget.update()
#####
self.update()

def main(argv=[ ]):
    app = QApplication(argv)
    window = MyWindow('데이터 입력')
    window.setFixedSize(800, 400)
    window.show()
    sys.exit(app.exec_())

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

```

이 코드를 실행하면 다음 그림과 같이 사용자가 정점을 입력하고, 프리미티브를 바꾸어가면서 OpenGL의 렌더링 결과를 변경해 볼 수 있는 프로그램이 완성된다.

