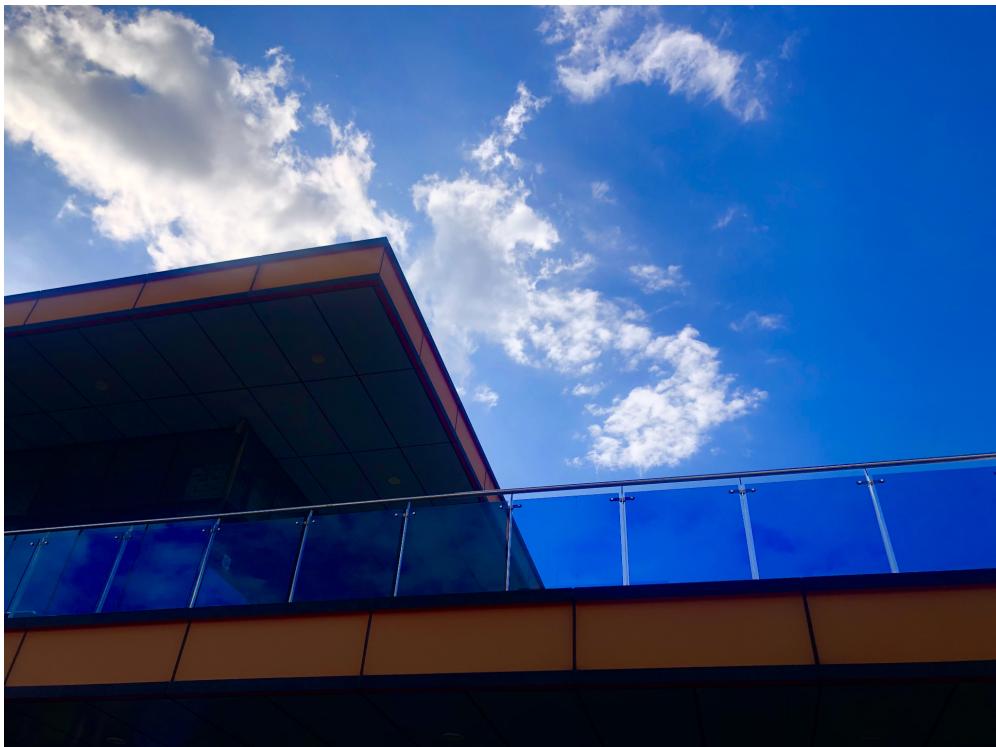


Chapter 05 -

동명대학교 게임공학과 3D 그래픽스 프로그래밍

그리기의 속도를 높이기



“기계를 빠르게 하는 건 하드웨어, 빠른 기계를 느리게 만드는 건 소프트웨어야.”
“It's hardware that makes a machine fast. It's software that makes a fast machine slow.”

— 크레이그 류캐슬 Craig Reucassel

이 장에서 생각할 문제

- ❖ OpenGL의 그리기 과정에서 CPU와 GPU가 담당하는 역할
- ❖ 그리기 성능을 개선하는 방법에 대한 이해
- ❖ 정점 배열을 활용한 그리기 기법 이해

3.1 그리기 속도를 높이는 방법 - 디스플레이 리스트

지금까지 우리는 `glBegin(primitive)`과 `glEnd()` 사이에 `glVertex*(...)` 등의 함수를 이용하여 정점 정보를 제공하는 방식으로 그림을 그렸다. 그런데 `glVertex*(...)`라는 함수를 호출한다는 것은 CPU가 함수 호출을 처리한다는 것을 의미한다. 이것은 병렬처리가 이루어지지 않고 순차적으로 처리될 수밖에 없다. 따라서 정점의 수가 매우 많은 경우 CPU의 함수 호출이 많아져서 그림을 그리는 일이 느려질 수밖에 없다.

예를 들어 화면에 정사각형을 x축과 z축 방향으로 한 칸씩 건너뛰며 그려 체스판과 같은 모양을 만들어 보자. 다음과 같은 그리기 함수를 만들면 될 것이다.

```
def drawPlane() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100       # 체스 판의 한쪽 면 길이
    d = w/(n-1)   # 인접한 두 정점 사이의 거리

    # 체스판을 그린다
    glBegin(GL_QUADS)
    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                glVertex3f(startX, 0, startZ)
                glVertex3f(startX, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ)
    glEnd()
```

여기에 사용되는 정점의 수는 250,000 개이다. 이 정점을 이용하여 만들어지는 격자 구조의 사각형은 모두 499×499 로 총 249,001개이며, 각각은 4 개의 점으로 이루어져 있어 996,004 개의 `glVertex*(...)` 호출이 필요하다. 이들 가운데 둘 중에 하나의 사각형은 그리지 않아도 되므로 총 498,002 번의 `glVertex*(...)` 호출이 필요하다. 상당히 많은 수의 정점 호출이다. 그리고, 사각형 내부를 칠하는 작업이 이루어져야 하므로 앞 장에서 다룬 선으로 만들어진 평면에 비해서 그리기 시간이 더 필요할 것이다. 따라서 이 코드를

사용하여 평면을 그리고 키보드를 이용하여 이 공간 위를 다니는 코드를 작성하면 매우 느린 동작을 보게 될 것이다.

속도가 매우 느린 평면 그리기

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

import sys

from PyQt6.QtWidgets import QApplication, QMainWindow
from PyQt6.QtWidgets import QWidget, QHBoxLayout
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
from PyQt6.QtCore import *
import math
import numpy as np

def drawPlane() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100       # 체스 판의 한쪽 면 길이
    d = w/(n-1)   # 인접한 두 정점 사이의 거리

    # 체스판을 그린다
    glBegin(GL_QUADS)
    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                glVertex3f(startX, 0, startZ)
                glVertex3f(startX, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ)
    glEnd()

class MyGLWidget(QOpenGLWidget):

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

        self.p = np.array([0.0,0.0,0.0], dtype=float)
        self.d = np.array([0.0,0.0,1.0], dtype=float)
        self.angle = 0.0
```

```

def initializeGL(self):
    # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
    glClearColor(0.0, 0.0, 0.0, 1.0)

def resizeGL(self, width, height):
    # 카메라의 투영 특성을 여기서 설정
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60, width/height, 0.1, 100)

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

    gluLookAt(self.p[0], self.p[1]+0.8, self.p[2],
              self.p[0]+self.d[0], self.p[1]+self.d[1], self.p[2]+self.d[2],
              0, 1, 0)

    glColor3f(1,1,0)
    drawPlane()
    glFlush()

class MyWindow(QMainWindow):

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

        self.glWidget = MyGLWidget()  # OpenGL Widget
        self.setCentralWidget(self.glWidget)

    def keyPressEvent(self, e):
        s = np.array([0.03])

        if e.key() == Qt.Key_W:
            self.glWidget.p += s*self.glWidget.d
        elif e.key() == Qt.Key_A:
            self.glWidget.angle += 0.05
            self.glWidget.d = (
                math.sin(self.glWidget.angle),

```

```

        0,
        math.cos(self.glWidget.angle))

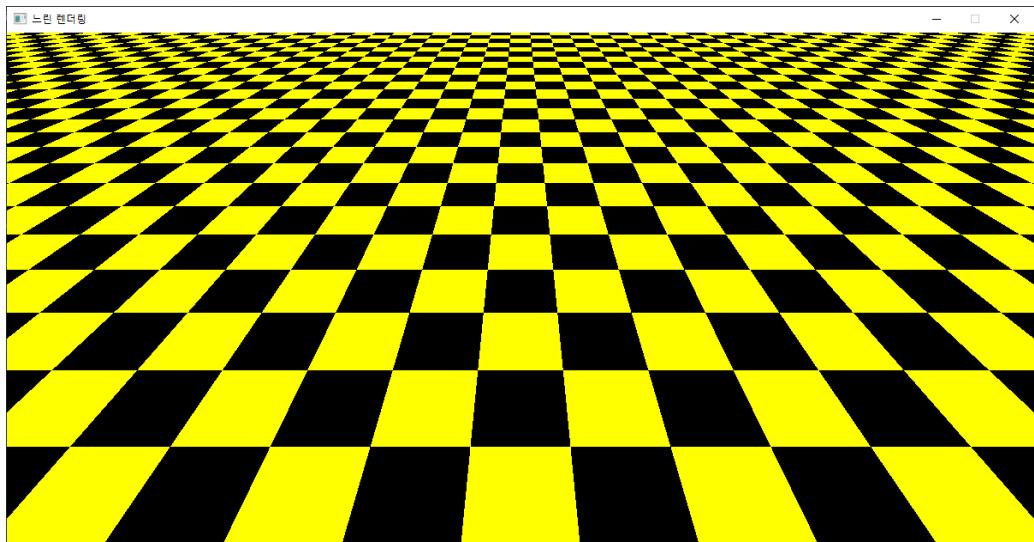
    self.glWidget.update()

def main(argv = []):
    app = QApplication(argv)
    window = MyWindow('Immediate Mode')
    window.setFixedSize(1200, 600)
    window.show()
    app.exec()

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

```

결과는 아래 그림 1과 같다. 이 프로그램은 W 키를 이용하여 카메라를 전진시키고, A 키를 눌러 카메라의 방향을 왼쪽으로 회전시킬 수 있게 하였다. 실제로 적용해 보면 렌더링에 소요되는 시간이 커서 카메라의 이동이 매우 느리다는 것을 확인할 수 있을 것이다. 어떻게 하면 렌더링 속도를 개선하여 카메라의 이동을 부드럽게 만들 수 있을까?



[그림 1] 많은 점점으로 그려진 평면의 예

많은 정점 정보를 사용할 때마다 `glVertex*(...)` 함수로 제공한다면 렌더링에 많은 시간이 소요될 수밖에 없다. 이러한 문제에 해결책이 될 수 있는 것이 디스플레이 리스트 `display list`이다.

디스플레이 리스트는 반복적으로 사용되는 그리기 명령어들을 캐시^{cache}하기 위한 방법이다. 디스플레이 리스트를 구성하는 그리기 명령들은 저장되어 있다가 필요할 때 불려진다. 이때 이 명령들을 미리 컴파일되고 최적화된 상태로 저장되어 있기 때문에 새롭게 함수 호출을 일일이 하는 것에 비해 훨씬 더 빠른 성능을 보인다.

디스플레이 리스트를 생성하는 방법은 `glGenLists(n)`과 `glNewList()`, 그리고 `glEndList()`를 이용하여 다음과 같이 구현한다.

```
idxFirstList = glGenLists(nRange)
for i in range(nRange):
    glNewList(idxFirstList+i, GL_COMPILE)
    # 리스트를 구성하는 그리기 동작
    glEndList()
```

이때 `nRange`은 새롭게 생성할 디스플레이 리스트의 개수를 의미한다. 이 리스트들은 연속된 번호를 부여받는데, 그 첫 번호가 반환된다. 위의 코드에서는 `idxFirstList`로 저장했다. 빈 리스트가 만들어지면 그 내용을 채울 수가 있는데 위의 코드에서는 모두 `nRange` 개수의 리스트를 만들 수 있다. `for` 구문을 이용하여 `nRange` 개의 리스트를 만든다고 가정해 보자. `glNewList()`와 `glEndList()`는 새로운 리스트의 내용을 채울 때에 시작과 끝을 나타낸다. 이 둘 사이의 있는 그리기 함수들이 리스트로 저장된다.

하나의 리스트를 생성하고, 여기에 평면 그리기를 담는다고 하면 다음과 같이 OpenGL 위젯의 `initializeGL(self)` 메소드 내에 평면 그리기 함수를 호출하여 리스트에 담으면 될 것이다. 이렇게 구현하면 다음에 평면 그리기가 필요할 때 `self.planeList`를 호출하기만 하면 된다.

```
def initializeGL(self):
    # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
    glClearColor(0.0, 0.0, 0.0, 1.0)

    self.planeList = glGenLists(1)
    glNewList(self.planeList, GL_COMPILE)
    drawPlane()
    glEndList()
```

리스트를 호출할 때에는 `glCallList(list_numer)`를 호출하면 된다. 따라서 `paintGL(self)` 메소드는 다음과 같이 평면 그리기 함수를 바로 호출하지 않고 리스트를 호출하는 방식으로 개선할 수 있다.

```
def paintGL(self):  
  
    glClear(GL_COLOR_BUFFER_BIT)  
    glMatrixMode(GL_MODELVIEW)  
    glLoadIdentity()  
  
    gluLookAt(self.p[0], self.p[1]+0.8, self.p[2],  
              self.p[0]+self.d[0], self.p[1]+self.d[1], self.p[2]+self.d[2],  
              0, 1, 0)  
  
    glColor3f(1,1,0)  
    glCallList(self.planeList) #drawPlane()  
    glFlush()
```

키보드를 누를 때마다 이 `paintGL(self)` 메소드는 계속해서 호출이 되며, 평면을 그리는 동작도 반복적으로 실행된다. 이때 OpenGL 코드를 다시 호출하여 그리는 것이 아니라 디스플레이 리스트에 담겨 있는 내용을 호출하는 방식을 사용하면 훨씬 더 빠르게 렌더링이 이루어진다.

키보드의 L 키를 이용하여 디스플레이 리스트를 사용하는 방식과, 사용하지 않는 방식을 번갈아 적용할 수 있는 코드를 작성하면 아래 코드와 같다.

속도가 매우 느린 평면 그리기

```
from OpenGL.GL import *  
from OpenGL.GLU import *  
  
import sys  
  
from PyQt6.QtWidgets import QApplication, QMainWindow  
from PyQt6.QtWidgets import QWidget, QHBoxLayout  
from PyQt6.QtOpenGLWidgets import QOpenGLWidget  
from PyQt6.QtCore import *  
import math  
import numpy as np
```

```

def drawPlane() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100       # 체스 판의 한쪽 면 길이
    d = w/(n-1)  # 인접한 두 정점 사이의 거리

    # 체스판을 그린다
    glBegin(GL_QUADS)
    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                glVertex3f(startX, 0, startZ)
                glVertex3f(startX, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ)
    glEnd()

class MyGLWidget(QOpenGLWidget):

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

        self.p = np.array([0.0,0.0,0.0], dtype=float)
        self.d = np.array([0.0,0.0,1.0], dtype=float)
        self.angle = 0.0
        self.listMode = False

    def initializeGL(self):
        # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
        glClearColor(0.0, 0.0, 0.0, 1.0)
        glEnable(GL_DEPTH_TEST)

        self.drawPlaneList = glGenLists(1)
        glColor3f(0,1,1)
        glNewList(self.drawPlaneList, GL_COMPILE)
        drawPlane()
        glEndList()

    def resizeGL(self, width, height):
        # 카메라의 투영 특성을 여기서 설정
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(60, width/height, 0.1, 100)

```

```

def paintGL(self):

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    gluLookAt(self.p[0], self.p[1]+0.8, self.p[2],
              self.p[0]+self.d[0], self.p[1]+self.d[1], self.p[2]+self.d[2],
              0, 1, 0)

    if self.listMode:
        glColor3f(0,1,1)
        glCallList(self.drawPlaneList)
    else :
        glColor3f(1,1,0)
        drawPlane()

    glFlush()

class MyWindow(QMainWindow):

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

        self.glWidget = MyGLWidget() # OpenGL Widget
        self.setCentralWidget(self.glWidget)

    def keyPressEvent(self, e):

        s = np.array([0.03])
        angle = self.glWidget.angle

        if e.key() == Qt.Key.Key_L:
            if self.glWidget.listMode:
                self.glWidget.listMode = False
            else:
                self.glWidget.listMode = True
            print(self.glWidget.listMode)

        if e.key() == Qt.Key.Key_W:
            self.glWidget.p += s*self.glWidget.d
        elif e.key() == Qt.Key.Key_S:
            self.glWidget.p -= s*self.glWidget.d

```

```

    elif e.key() == Qt.Key.Key_A:
        angle += 0.05
        self.glWidget.d = (math.sin(angle), 0, math.cos(angle))
    elif e.key() == Qt.Key.Key_D:
        angle -= 0.05
        self.glWidget.d = (math.sin(angle), 0, math.cos(angle))

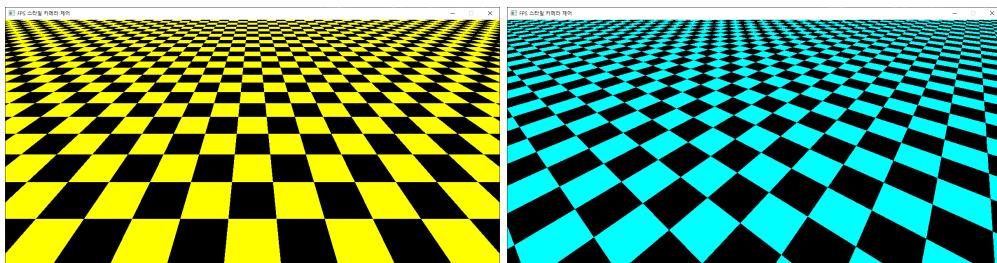
    self.glWidget.angle = angle
    self.glWidget.update()

def main(argv = []):
    app = QApplication(argv)
    window = MyWindow('FPS 스타일 카메라 제어')
    window.setFixedSize(1200, 600)
    window.show()
    app.exec()

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

```

그림 2는 위의 코드를 실행했을 때, L 키를 눌러 디스플레이 리스트를 사용하거나 사용하지 않을 때를 구분한 결과이다. 노란색으로 그려질 때는 디스플레이 리스트를 사용하지 않은 경우이고 카메라 이동이 매우 느리게 이루어진다. 그러나 L 키를 누르면 바닥이 하늘색으로 그려지고 이 그리기는 디스플레이 리스트를 이용하여 이루어진다. 그리고 카메라를 조작해 보면 훨씬 더 빨라진 것을 확인할 수 있을 것이다.



【그림 2】 디스플레이 모드를 사용하지 않은 경우와 사용한 경우의 색을 달리하여 비교

3.2 정점 배열 ^{vertex array}을 `glDrawArrays`로 그리기

지금까지 OpenGL의 `glBegin()`과 `glEnd()` 사이에 정점 정보를 제공하여 그림을 그리는 방식을 사용하였다. 이 절에서 살펴볼 방법은 `glVertex*`(...) 함수 등을 불러 정점의 정보를 제공하는 방식이 아니라 정점의 정보가 담겨 있는 배열을 GPU에 통째로 넘겨서 처리하게 하는 방법이다. 정점 뿐만 아니라, 색상이나 각 정점 위치에서 기하 객체의 표면이 바라보는 법선normal 벡터의 배열도 전달할 수 있지만, 우선은 정점 배열만을 사용해 보자.

정점 배열을 사용하기 위해서는 `glEnableClientState(array)` 함수를 사용해야 한다. `array` 인자는 어떤 배열을 사용할 수 있도록 활성화하는지를 지정하는데, 정점 배열을 활성화할 때는 `GL_VERTEX_ARRAY`, 색상 배열을 활성화할 때는 `GL_COLOR_ARRAY`, 법선 벡터 배열을 활성화할 때는 `GL_NORMAL_ARRAY`를 사용한다.

OpenGL에서 CPU는 그래픽 처리의 클라이언트client이고, GPU는 서버server가 된다. GPU 쪽의 처리에서 필요한 기능을 활성화할 때는 `glEnable`을 사용하는데, 정점 배열을 사용하는 것은 CPU 쪽에서 활성화할 일이기 때문에 `glEnableClientState()` 함수를 사용하였다.

정점 배열을 활성화하였다면, GPU가 사용할 정점 배열을 지정해야 한다. 이때 사용하는 것이 `glVertexPointer(...)` 함수이다. 이 함수는 다음과 같은 함수 원형을 갖는다.

`glVertexPointer(size, type, stride, pointer)`

- `size`: 하나의 정점을 구성하는 좌표의 수. 2, 3, 또는 4 중에 하나이다.
- `type`: 정점 배열에 들어 있는 숫자의 자료형으로 `GL_INT`, `GL_FLOAT` 등이 가능하다.
- `stride`: 배열 내에서 정점들 사이의 간격을 바이트 단위로 표시한 것. 0이라면 사용하는 정점이 배열 내에서 간격이 0인 상태로 배치되어 있다는 의미이다.
- `pointer`: 배열의 주소. C 언어의 경우 포인터 변수나 배열 이름이 되고, 파이썬에서는 리스트가 참조값으로 다루어지므로 리스트 변수를 그대로 사용하면 된다.

이를 활용하기 위해서는 정점 좌표 정보가 리스트로 저장되어야 한다. 평면을 그리는 함수에서 `glVertex3f(...)` 함수가 이 정점 정보를 전달하는 역할을 했다. 따라서 이때

결정된 정점 정보를 리스트에 추가하는 다음과 같은 코드로 정점 버퍼를 만들 수 있을 것이다. 앞 절에서 사용했던 `drawPlane()` 함수는 그리기를 실행하는 코드였고, 여기서는 정점 정보를 `vertexBuffer`에 계속해서 누적하는 일을 하고, 그 결과 리스트를 반환하는 일을 한다. 이 리스트가 정점 배열로 사용될 것이다. 이때 `drawPlane`의 좌표와 비교했을 때 y 값이 0이 아니라 0.3으로 조금 더 높은 곳에 있도록 했다. 둘을 다 그렸을 때 다른 곳에 위치하여 모두 관찰할 수 있도록 하기 위해서이다.

```
def drawPlane_vertexBuffer():
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100      # 체스 판의 한쪽 면 길이
    d = w/(n-1)  # 인접한 두 정점 사이의 거리

    vertexBuffer = []
    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                vertexBuffer.append([startX, 0.3, startZ])
                vertexBuffer.append([startX, 0.3, startZ+d])
                vertexBuffer.append([startX+d, 0.3, startZ+d])
                vertexBuffer.append([startX+d, 0.3, startZ])
    return vertexBuffer
```

정점 배열을 생성하기 위해서는 이 함수를 호출하여 정점 정보가 담긴 배열 혹은 리스트를 생성하고, 정점 배열 사용을 활성화한 뒤에 생성된 리스트를 정점 배열로 넘긴다. 이 과정은 OpenGL 위젯의 초기화 함수, 즉 `initializeGL`에서 이루어지면 될 것이다.

구현의 예는 다음과 같다.

```
def initializeGL(self):
    ...
    self.vBuffer = drawPlane_vertexBuffer()
    glEnableClientState(GL_VERTEX_ARRAY)          # 정점 버퍼 사용
    glVertexPointer(3, GL_FLOAT, 0, self.vBuffer) # 정점 버퍼 설정
    # 서버(GPU)는 지정된 정점 버퍼의 내용을 가져다가 사용하게 됨
```

정점 버퍼의 내용을 그릴 때는 `glDrawArrays(...)` 함수를 사용한다. 이 함수의 원형은 다음과 같다.

- `glDrawArrays(primitive_mode, start_index, count)`

`primitive_mode`: 활성화된 정점 배열을 이용하여 그림을 그릴 때 어떤 방식으로 조합하여 그릴지를 결정하는 프리미티브 설정. GL_POINTS, GL_LINES 등이 가능하다.

`start_index`: 활성화된 정점 배열에서 그리기를 시작할 첫번째 정점의 인덱스

`count`: 사용할 정점의 총 개수

따라서 다음과 같이 `glDrawArrays()` 함수를 사용하면 정점 배열에 있는 정점들을 이용하여 그림을 그릴 수 있게 된다. `glDrawArrays()` 함수의 첫 인자는 정점 배열에 있는 정점들을 4 개씩 묶어 사각형들을 그리도록 하는 `GL_QUADS`가 제공되었다. 그리고 정점 배열의 가장 첫 원소부터 사용하기 위해 0을 시작 인덱스로 제공하였다. 우리가 만든 정점 배열 `self.vBuffer`는 하나의 정점이 세 개의 원소를 가진 리스트로 구성되고, 이 리스트들의 리스트로 구성되기 때문에 `len(self.vBuffer)`는 정점의 개수를 반환한다. 이 값이 마지막에 제공된다.

```
def paintGL(self):
    ...
    glColor3f(0,1,0)
    glDrawArrays(GL_QUADS, 0, len(self.vBuffer))
```

이러한 내용을 종합하여 디스플레이 리스트로 그려진 평면은 하늘색, 정점 배열을 이용하여 그려진 평면은 녹색으로 화면에 표시해 보자. 두 방식 모두 빠르게 렌더링을 진행하기 때문에 둘을 모두 사용하여도 키보드를 이용한 카메라 이동이 부드럽게 일어날 것이다. 다음은 이를 테스트하기 위한 코드이다.

glDrawArrays를 이용한 빠른 그리기

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

import sys

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

```

from PyQt6.QtCore import *

import math
import numpy as np

def drawPlane() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100       # 체스 판의 한쪽 면 길이
    d = w/(n-1)   # 인접한 두 정점 사이의 거리

    # 체스판을 그린다
    glBegin(GL_QUADS)
    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                glVertex3f(startX, 0, startZ)
                glVertex3f(startX, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ+d)
                glVertex3f(startX+d, 0, startZ)
    glEnd()

def drawPlane_vertexBuffer() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100       # 체스 판의 한쪽 면 길이
    d = w/(n-1)   # 인접한 두 정점 사이의 거리

    vertexBuffer = []

    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                vertexBuffer.append([startX, 0.3, startZ])
                vertexBuffer.append([startX, 0.3, startZ+d])
                vertexBuffer.append([startX+d, 0.3, startZ+d])
                vertexBuffer.append([startX+d, 0.3, startZ])

    return vertexBuffer

class MyGLWidget(QOpenGLWidget):

    def __init__(self, parent=None):

```

```

super().__init__(parent)

self.p = np.array([0.0,0.0,0.0], dtype=float)
self.d = np.array([0.0,0.0,1.0], dtype=float)
self.angle = 0.0
self.listMode = False


def initializeGL(self):
    # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
    glClearColor(0.0, 0.0, 0.0, 1.0)

    self.planeList = glGenLists(1)
    glNewList(self.planeList, GL_COMPILE)
    drawPlane()
    glEndList()

    self.vBuffer = drawPlane_vertexBuffer()
    ## 그래픽 처리에서 클라이언트는 CPU / 서버는 GPU
    ## glEnable은 서버 상태를 변경하는 것
    ## 정점 배열 사용은 CPU 일이므로 glEnable 대신 glEnableClientState
    glEnableClientState(GL_VERTEX_ARRAY)          # 정점 버퍼 사용
    glVertexPointer(3, GL_FLOAT, 0, self.vBuffer) # 정점 버퍼 설정
    # 서버(GPU)는 지정된 정점 버퍼의 내용을 가져다가 사용하게 됨


def resizeGL(self, width, height):
    # 카메라의 투영 특성을 여기서 설정
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60, width/height, 0.1, 100)


def paintGL(self):
    global t

    glClear(GL_COLOR_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    gluLookAt(self.p[0], self.p[1]+1.8, self.p[2],
              self.p[0]+self.d[0], self.p[1]+self.d[1], self.p[2]+self.d[2],
              0, 1, 0)

    glColor3f(0,0.5,1)
    glCallList(self.planeList)

    glColor3f(0,1,0)
    glDrawArrays(GL_QUADS, 0, len(self.vBuffer))

```

```
glFlush()

class MyWindow(QMainWindow):

    def __init__(self, title=''):
        QMainWindow.__init__(self) # QMainWindow 슈퍼 클래스의 초기화
        self.setWindowTitle(title)
        self.glWidget = MyGLWidget() # OpenGL Widget
        self.setCentralWidget(self.glWidget)

    def keyPressEvent(self, e):

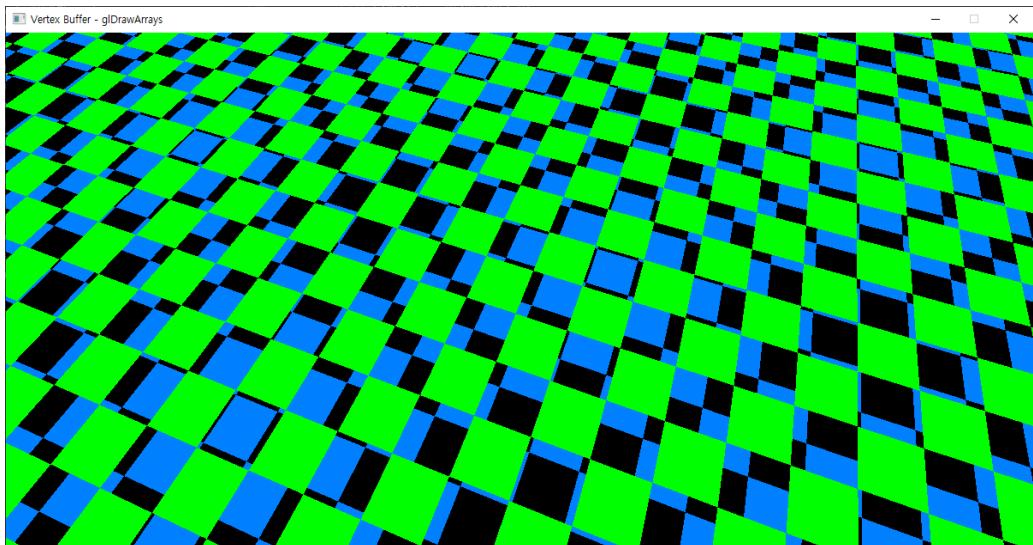
        s = np.array([0.03])
        angle = self.glWidget.angle
        if e.key() == Qt.Key.Key_L:
            if self.glWidget.listMode:
                self.glWidget.listMode = False
            else:
                self.glWidget.listMode = True
            print(self.glWidget.listMode)
        if e.key() == Qt.Key.Key_W:
            self.glWidget.p += s*self.glWidget.d
        elif e.key() == Qt.Key.Key_S:
            self.glWidget.p -= s*self.glWidget.d
        elif e.key() == Qt.Key.Key_A:
            angle += 0.05
            self.glWidget.d = (math.sin(angle), 0, math.cos(angle))
        elif e.key() == Qt.Key.Key_D:
            angle -= 0.05
            self.glWidget.d = (math.sin(angle), 0, math.cos(angle))

        self.glWidget.angle = angle
        self.glWidget.update()

def main(argv = []):
    app = QApplication(argv)
    window = MyWindow('느린 렌더링')
    window.setFixedSize(1200, 600)
    window.show()
    app.exec()

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

결과는 그림 3과 같이 디스플레이 리스트로 그려진 파란색 평면 위에, `glDrawArrays`를 이용하여 그린 녹색 평면이 나타날 것이다. 둘 다 빠른 렌더링이 가능하기 때문에 키보드를 조작하면 부드럽게 카메라가 이동하는 것을 확인할 수 있다.



[그림 3] 디스플레이 리스트와 `glDrawArrays`를 이용한 렌더링을 동시에 적용한 결과

정점의 색상을 추가하여 보자. 이를 위해서는 색상 배열을 사용할 수 있다. 색상 배열을 추가하는 방법은 `drawPlane_vertexBuffer()` 함수를 개선하여 다음과 같이 구현할 수 있다. 이 코드에서는 색상 배열로 사용될 리스트를 하나 추가하고, 각 정점에 대해 대응하는 색상을 지정해 리스트에 차례로 추가한다.

```
def drawPlane_vertexBuffer() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100       # 체스 판의 한쪽 면 길이
    d = w/(n-1)   # 인접한 두 정점 사이의 거리

    vertexBuffer = []
    colorBuffer = []
    for i in range(n):
        for j in range(n):
            startX = -w/2 + i*d
            startZ = -w/2 + j*d
            # 행과 열의 번호 합이 짝수일 때만 그린다 (체스판)
            if (i+j)%2 == 0:
                colorBuffer.append([1.0, 0.0, 0.0])
```

```

    vertexBuffer.append([startX, 0.3, startZ])
    colorBuffer.append([0.0, 1.0, 0.0])
    vertexBuffer.append([startX, 0.3, startZ+d])
    colorBuffer.append([0.0, 0.0, 1.0])
    vertexBuffer.append([startX+d, 0.3, startZ+d])
    colorBuffer.append([1.0, 1.0, 0.0])
    vertexBuffer.append([startX+d, 0.3, startZ])

return vertexBuffer, colorBuffer

```

이 함수를 호출하여 얻을 수 있는 것은 정점 배열과 색상 배열이다. 이를 각각 `self.vBuffer`와 `self.cBuffer`로 받을 수 있도록 OpenGL 위젯의 `initializeGL` 함수를 수정한다. 그리고

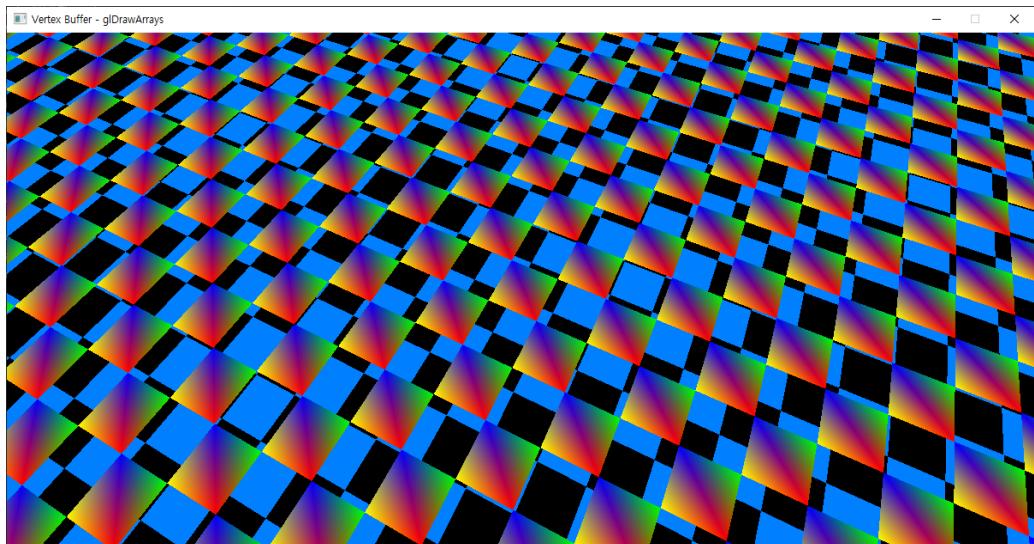
```

self.vBuffer, self.cBuffer = drawPlane_vertexBuffer()

glEnableClientState(GL_VERTEX_ARRAY)           # 정점 배열 사용
glEnableClientState(GL_COLOR_ARRAY )          # 색상 배열 사용
glVertexPointer(3, GL_FLOAT, 0, self.vBuffer) # 정점 배열 설정
glColorPointer(3, GL_FLOAT, 0, self.cBuffer) # 색상 배열 설정

```

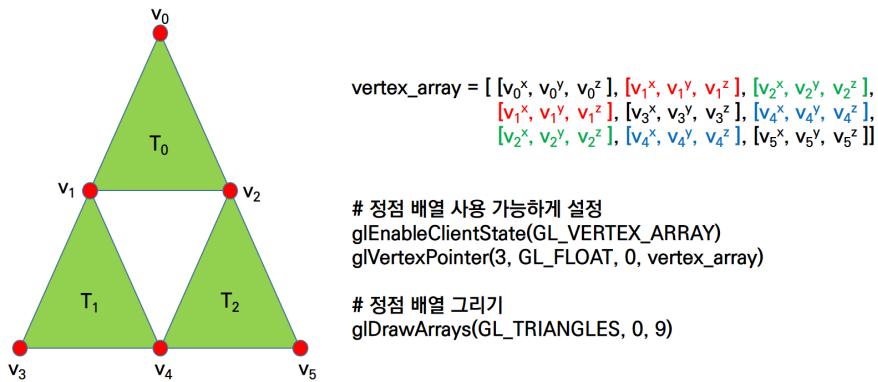
그리고 나서 앞서와 동일한 배열 그리기를 실행하면, 정점 배열과 함께 색상 배열을 함께 사용하게 되어 각 정점의 색상이 새롭게 지정된 다음과 같은 결과를 얻을 수 있다.



[그림 4] 색상 배열을 사용하여 정점의 색상도 조절한 결과

3.3 정점 배열 vertex array을 glDrawElements로 그리기

정점 배열을 이용하여 아래 그림 5와 같이 세 개의 삼각형을 그린다고 가정해 보자. 각각의 삼각형에 3 개의 정점이 필요하므로 총 9 개의 정점이 필요하다. 그런데 그림의 v_1 은 삼각형 T_1 과 T_2 에 중복하여 사용되고, v_2 는 삼각형 T_1 과 T_3 에 중복하여 사용되고 있다. 또한 v_4 역시 삼각형 T_2 과 T_3 에 중복된다. 따라서 실제로 필요한 정점은 6 개 뿐이다. 그러나 앞에서 사용한 방법대로 구현하면 그림의 오른쪽 코드와 같다.

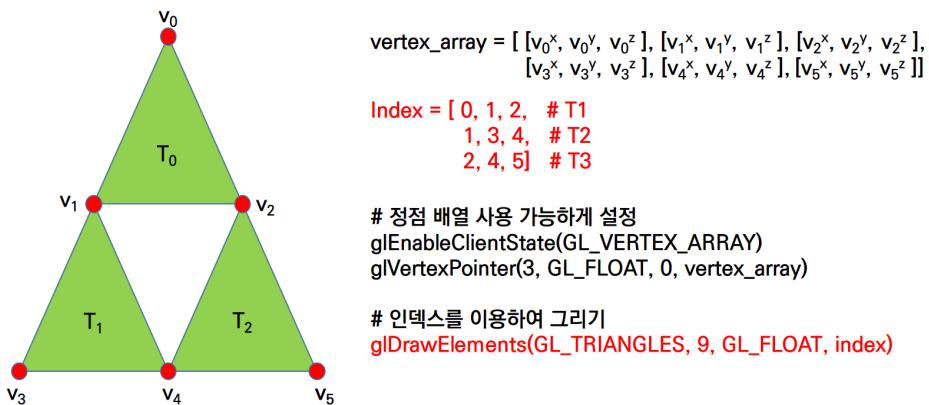


[그림 5] 정점 배열을 이용하여 정점을 공유하는 세 개의 삼각형 그리기

`vertex_array`의 첫 세 정점은 T_1 , 2행의 세 정점은 T_2 , 그리고 마지막 세 정점은 T_0 를 표현하는 정점인데, 검정색 좌표를 제외하고 빨간색, 녹색, 청색으로 표시된 좌표들은 중복하여 나타난다. 이것은 불필요한 메모리 사용이 될 수 있다.

이러한 문제를 해결하기 위해 `glDrawElements`를 사용할 수 있다. 이것은 그려지는 기하객체의 정점을 나열하는 것이 아니라, 정점을 중복없이 나열해 두고 그릴 때마다 정점의 인덱스^{index}를 사용하여 필요한 정점을 지정하는 것이다. 그림 6을 살펴보자. 정점은 중복없이 6 개만을 사용한다. 그리고 그리기의 대상이 되는 세 개의 삼각형을 표현하기 위해 사용하는 정점의 인덱스만 지정하면 된다. 이 인덱스를 배열 혹은 리스트로 담으면 된다.

`glDrawElements`는 인덱스를 사용하여 정점 배열의 점들에 접근하고 이를 이용하여 그림을 그린다. 예를 들어 그림 6은 `GL_TRIANGLES`를 프리미티브로 지정하여 삼각형 여러 개를 그리려고 한다. 이때 그려지는 삼각형의 개수는 3 개이고, 각각의 삼각형은 정점 인덱스 3 개가 필요하므로, 사용되는 인덱스는 모두 9개이다. 이 값이 다음 인자로 지정된다. 그리고, 인덱스를 표현하는 데이터의 형`type`은 `GL_UNSIGNED_INT`이다. 그리고, 인덱스가 담긴 배열을 지정하면 된다



[그림 6] 인덱스를 이용하여 정점 배열을 효율적으로 사용하는 방법

따라서 코드를 정리하면 다음과 같은 방식으로 삼각형을 그릴 수 있다. 이 방법은 복잡한 외형을 가진 메시^{mesh} 데이터를 읽고 그릴 때에 유용한 방법이다. 이는 나중에 살펴보자.

glDrawArrays를 이용한 빠른 그리기

```

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

import sys

from PyQt6.QtWidgets import QApplication, QMainWindow
from PyQt6.QtWidgets import QWidget, QHBoxLayout
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
from PyQt6.QtCore import *
import math
import numpy as np

```

```

import time

def createBuffers() :

    vertexBuffer = [ [0.0, 1.0, 0.0], [-0.5, 0.0, 0.0], [0.5, 0.0, 0.0] ,
                    [-1.0, -1.0, 0.0], [0.0, -1.0, 0.0], [1.0, -1.0, 0.0] ]

    indexBuffer = [ 0, 1, 2, 1, 3, 4, 2, 4, 5]
    return vertexBuffer, indexBuffer

class MyGLWidget(QOpenGLWidget):

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

        self.p = np.array([0.0,0.0,0.0], dtype=float)
        self.d = np.array([0.0,0.0,1.0], dtype=float)
        self.angle = 0.0
        self.listMode = False


    def initializeGL(self):
        # OpenGL 그리기를 수행하기 전에 각종 상태값을 초기화
        glClearColor(0.0, 0.0, 0.0, 1.0)

        self.vBuffer, self.iBuffer = createBuffers()
        glEnableClientState(GL_VERTEX_ARRAY)           # 정점 배열 사용
        glVertexPointer(3, GL_FLOAT, 0, self.vBuffer) # 정점 배열 설정


    def resizeGL(self, width, height):
        # 카메라의 투영 특성을 여기서 설정
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(60, width/height, 0.1, 100)

    def paintGL(self):

        glClear(GL_COLOR_BUFFER_BIT)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()

        gluLookAt(0,0,-3, 0,0,0, 0,1,0)

        glColor3f(0,1,0.5)
        glDrawElements(GL_TRIANGLES, 9, GL_UNSIGNED_INT, self.iBuffer)

        glFlush()

```

```
class MyWindow(QMainWindow):  
  
    def __init__(self, title=''):  
        QMainWindow.__init__(self) # QMainWindow 슈퍼 클래스의 초기화  
        self.setWindowTitle(title)  
  
        self.glWidget = MyGLWidget() # OpenGL Widget  
        self.setCentralWidget(self.glWidget)  
  
def main(argv = []):  
    app = QApplication(argv)  
    window = MyWindow('glDrawElements')  
    window.setFixedSize(600, 600)  
    window.show()  
    app.exec()  
  
if __name__ == '__main__':  
    main(sys.argv)
```

실행 결과는 다음 그림 7과 같다.



[그림 7] glDrawElements를 활용한 예제의 결과