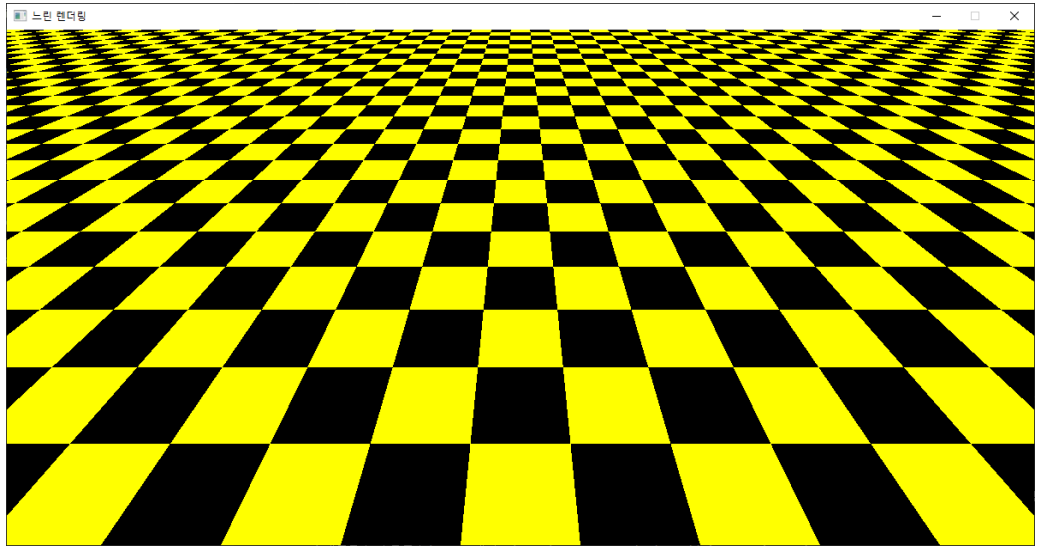
# 렌더링 속도를 개선하기

동명대학교 게임공학과
강영민

# 정점이 많은 기하 객체

```
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()
```



# 앞서 구현한 FPS 게임 카메라 실습과 연결

```
def drawPlane():
    glBegin(GL_LINES)
    for i in range(100):
        glVertex3f(i-50, 0,  -50)
        glVertex3f(i-50, 0,   50)
        glVertex3f( -50, 0, i-50)
        glVertex3f(  50, 0, i-50)
    glEnd()
```

```
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()
```

# 매우 느린 이동

왜 이런 속도가 나타날까?

매번 CPU가 정점 지정을 위한 glVertex3f 호출이 반복됨

```
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 paintGL(self):


        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(60, 2, 0.2, 100)

        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()
        gluLookAt( self.cam[0], self.cam[1] + 0.6, self.cam[2], # 카메라 위치
                   self.target[0], self.target[1], self.target[2], # 카메라가 쳐다보는 지점
                   0, 1, 0  # 카메라 위쪽 방향
        )

        drawPlane()
```

# Display list 사용하기

- 리스트 만드는 방법

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

  - nRange: 디스플레이 리스트의 개수
  - glGenLists: nRange 개수의 리스트를 생성할 수 있도록 함
    - 연속된 번호 중 첫 번호를 리턴
  - glNewList(리스트 번호, …
    - 리스트 번호에 해당하는 리스트 정의를 시작
  - glEndList
    - 만들고 있는 리스트를 완료함

# 리스트 호출

- 정의해 둔 리스트를 사용
  - 해당 그리기를 위해 준비한 리스트가 실행 됨
- 디스플레이 리스트가 빠른 이유
  - 명령 최적화
    - 렌더링 명령을 다시 하지 않고 이전에 정의된 명령을 다시 실행
  - 메모리 최적화
  - 상태 변경 감소
  - OpenGL 드라이버 최적화
- 주의점
  - 디스플레이 리스트는 동적인 데이터 렌더링에는 적합하지 않음

# 실제 사용예

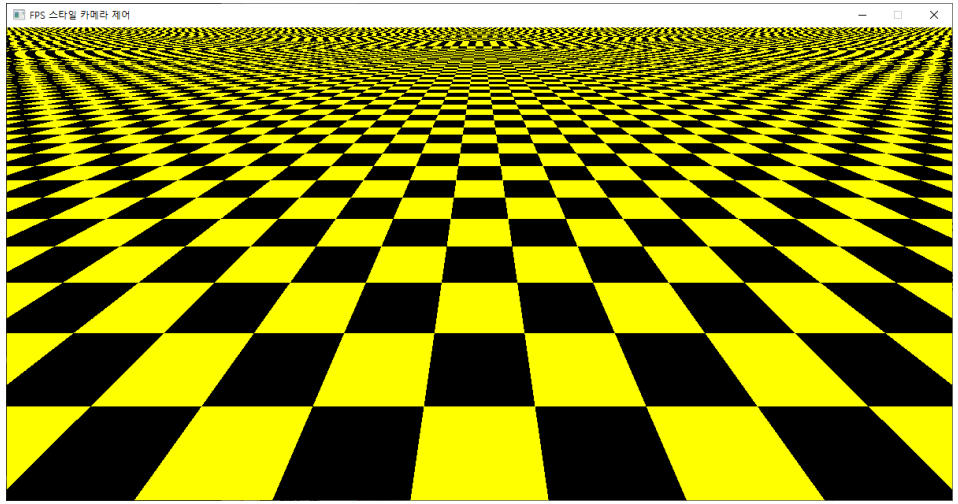
```
class MyGLWidget(QOpenGLWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        # 멤버 변수를 선언한다
        # 카메라의 위치
        self.cam = np.array([0, 0, 0], dtype = float)
        self.camDir = np.array([0, 0, 1], dtype = float)
        self.target = self.cam + self.camDir
        self.angle = 0.0

    def initializeGL(self):
        self.drawPlaneList = glGenLists(1)
        glColor3f(0,1,1)
        glNewList(self.drawPlaneList, GL_COMPILE)
        drawPlane()
        glEndList()
```

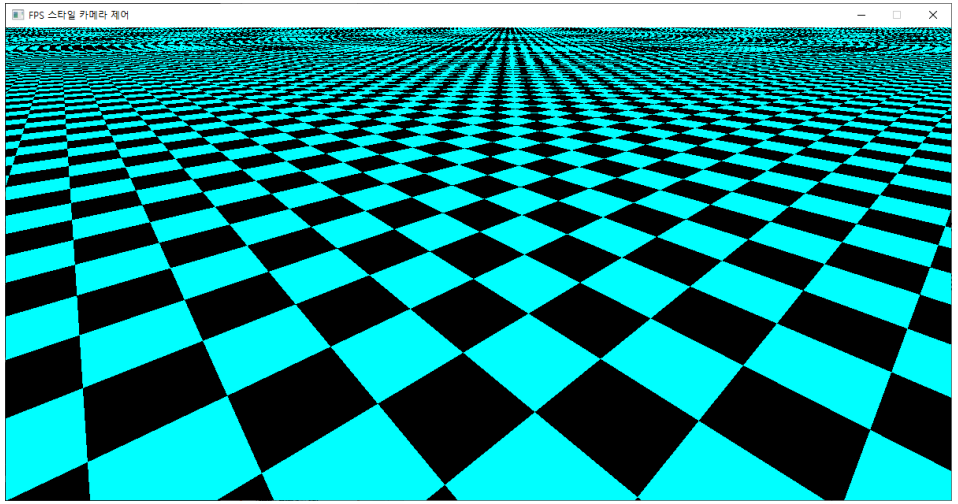
# 실제 사용예

```
    def paintGL(self):


        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(60, 2, 0.2, 100)

        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()
        gluLookAt( self.cam[0], self.cam[1] + 0.6, self.cam[2], # 카메라 위치
                   self.target[0], self.target[1], self.target[2], # 카메라가 쳐다보는 지점
                   0, 1, 0  # 카메라 위쪽 방향
        )

        glCallList(self.drawPlaneList)
```

# [실습] 키보드로 렌더링 방법 전환하여 속도 비교

- 키보드를 이용하여 평면을 그리는 방식을
  - Immediate 모드 ←→ display list 모드로 전환하여
  - 카메라 이동의 속도를 비교해 보자
  - 그림을 그릴 때, 렌더링 모드에 따라 평면의 색이 다르게 하자



# 정점 배열 그리기

- glDrawArrays
  - 정점의 정보를 담고 있는 배열을 통째로 넘기는 방식

- 준비
  - glEnableClientState(array)
    - array: GL_VERTEX_ARRAY, GL_COLOR_ARRAY, GL_NORMAL_ARRAY
    - client vs Server in OpenGL
      - CPU – client
      - GPU – server
  - glVertexArray(…
    - GPU가 접근해야 할 버퍼의 정보를 제공

# parameters

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.

# parameters

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

# 정점 버퍼 준비

- 배열, 리스트 등의 자료형에 담으면 됨

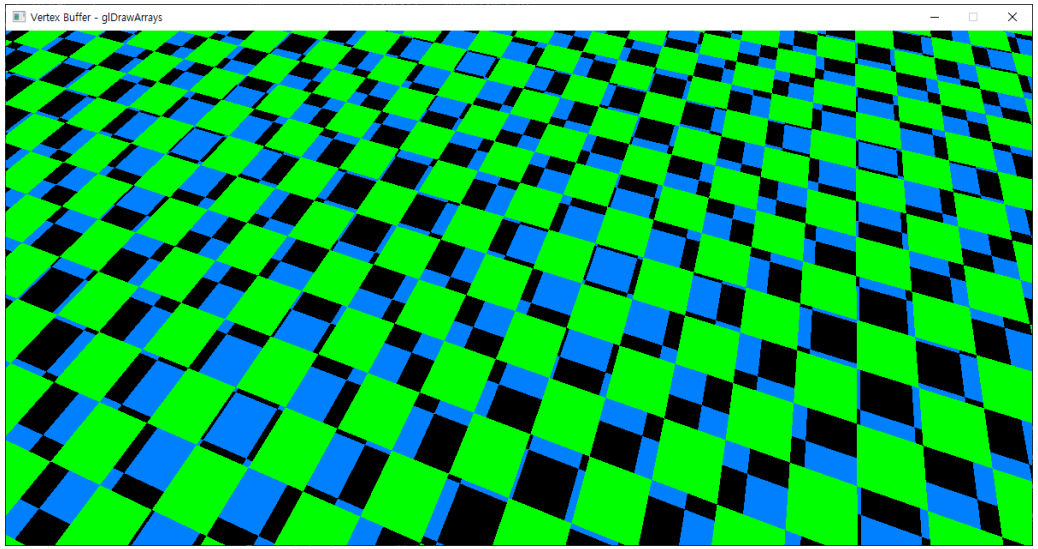
```
def drawPlane_vertexBuffer() :
    n = 500      # 체스 판을 구성하는 한 면에 놓일 정점의 수
    w = 100      # 체스 판의 한쪽 면 길이
    d = w/(n-1)  # 인접한 두 정점 사이의 거리

    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 버퍼를 GPU에게 제공

```
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
```

# 버퍼를 이용한 그리기

```
def paintGL(self):
        ...
        glColor3f(0,0.5,1)
        glCallList(self.planeList)

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



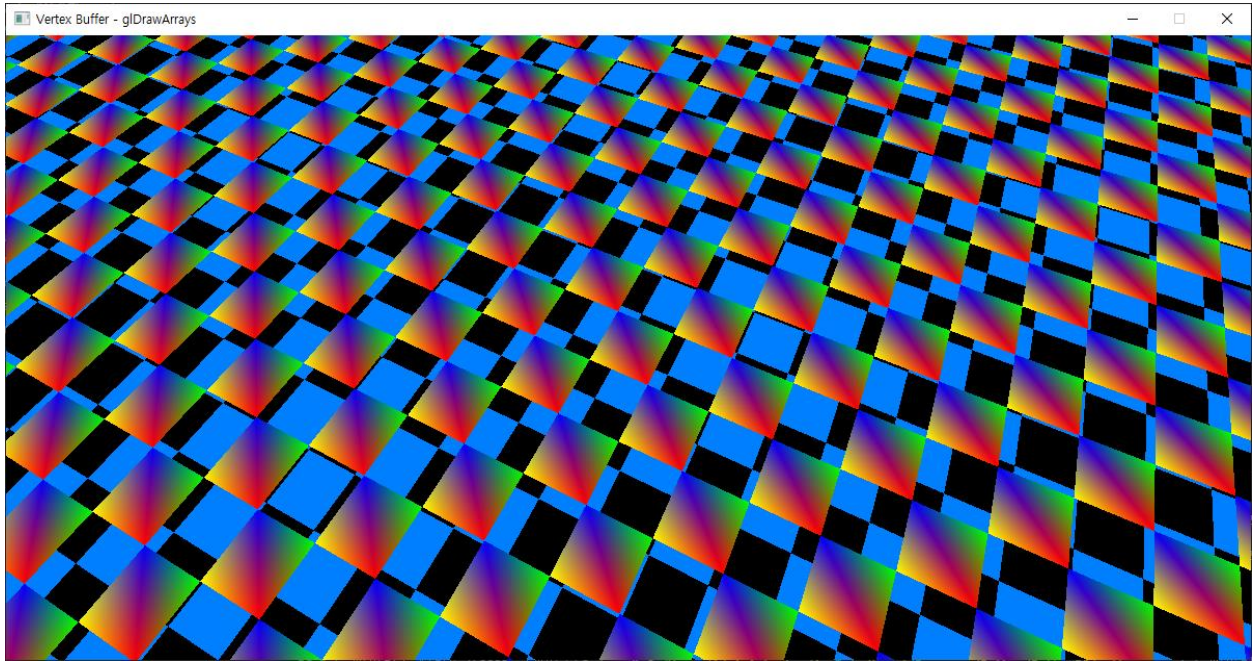

디스플레이 리스트와
glDrawArrays를 이용한 렌더링을
동시에 적용한 결과

# 정점에 색상 추가

```
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
```

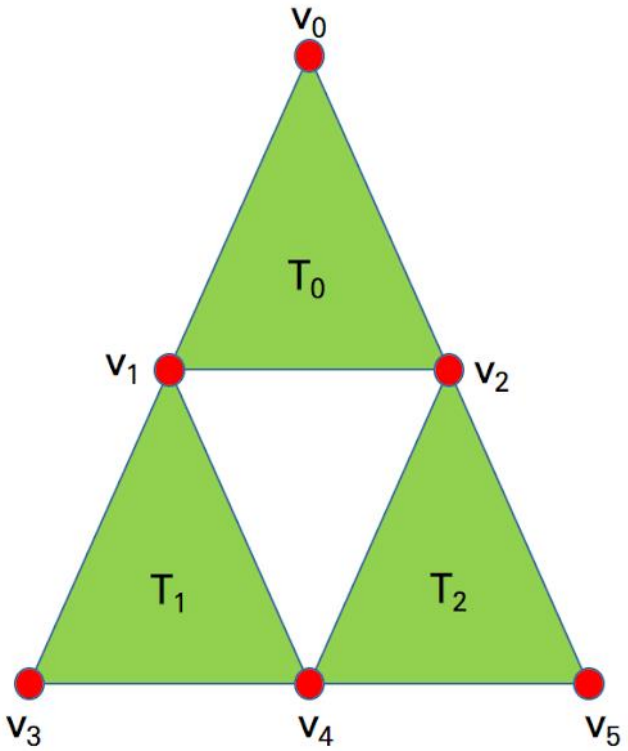
# 정점에 색상 추가

```
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
```



# 정점 배열을 glDrawElements로 그리기

- glDrawArray 방식의 문제

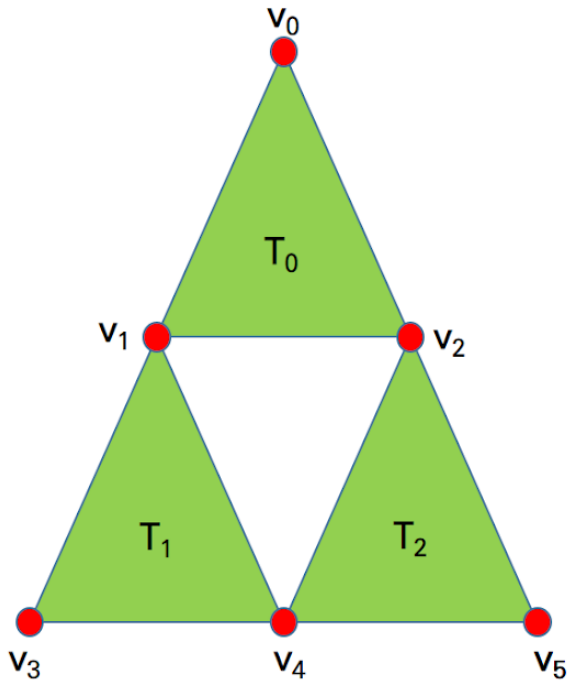



vertex_array = [ [$v_0^x$, $v_0^y$, $v_0^z$ ], [$v_1^x$, $v_1^y$, $v_1^z$ ], [$v_2^x$, $v_2^y$, $v_2^z$ ],
[$v_1^x$, $v_1^y$, $v_1^z$ ], [$v_3^x$, $v_3^y$, $v_3^z$ ], [$v_4^x$, $v_4^y$, $v_4^z$ ],
[$v_2^x$, $v_2^y$, $v_2^z$ ], [$v_4^x$, $v_4^y$, $v_4^z$ ], [$v_5^x$, $v_5^y$, $v_5^z$ ]]

```
# 정점 배열 사용 가능하게 설정
glEnableClientState(GL_VERTEX_ARRAY)
glVertexPointer(3, GL_FLOAT, 0, vertex_array)

# 정점 배열 그리기
glDrawArrays(GL_TRIANGLES, 0, 9)
```

# glDrawElements로 개선

- Indexed Facet




vertex_array = [ [$v_0^x$, $v_0^y$, $v_0^z$ ], [$v_1^x$, $v_1^y$, $v_1^z$ ], [$v_2^x$, $v_2^y$, $v_2^z$ ],
[$v_3^x$, $v_3^y$, $v_3^z$ ], [$v_4^x$, $v_4^y$, $v_4^z$ ], [$v_5^x$, $v_5^y$, $v_5^z$ ]]

```
Index = [ 0, 1, 2,   # T1
          1, 3, 4,   # T2
          2, 4, 5]   # T3
```

```
# 정점 배열 사용 가능하게 설정
glEnableClientState(GL_VERTEX_ARRAY)
glVertexPointer(3, GL_FLOAT, 0, vertex_array)

# 인덱스를 이용하여 그리기
glDrawElements(GL_TRIANGLES, 9, GL_FLOAT, index)
```

# glDrawElements: mesh 그리기에 유용