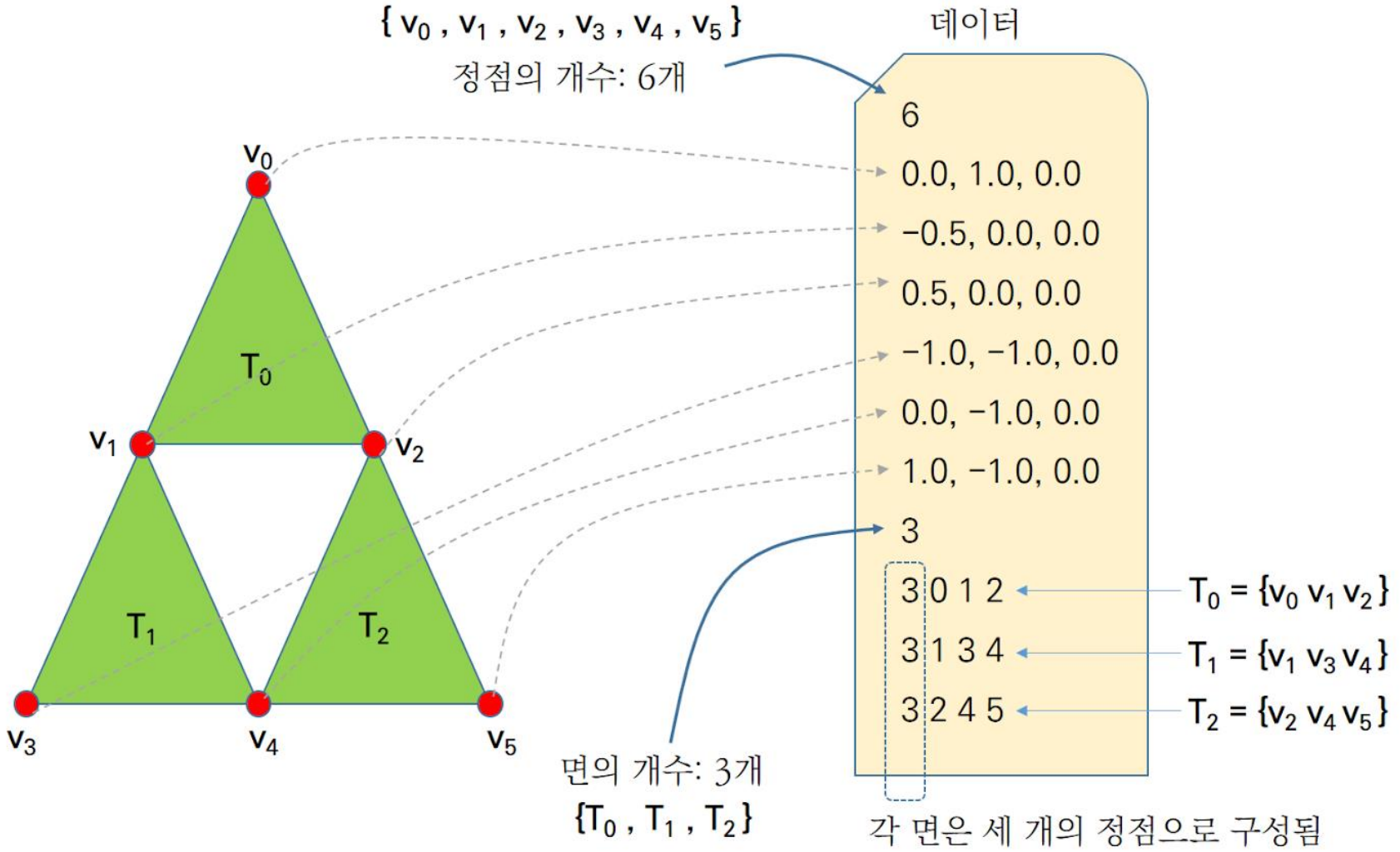
파일에 담긴 기하 객체 읽고 그리기

동명대학교 게임공학과
강영민

간단한 메시 데이터 포맷

데이터 양식	실제 데이터 예시
정점의 개수 (nV)	6
정점 0의 좌표	0.0, 1.0, 0.0
정점 1의 좌표	−0.5, 0.0, 0.0
정점 2의 좌표	0.5, 0.0, 0.0
...	−1.0, −1.0, 0.0
정점 nV−1의 좌표	0.0, −1.0, 0.0
면의 개수 (nF)	1.0, −1.0, 0.0
(면 0 구성 정점의 수 k) (정점 번호) ... k개	3
(면 1 구성 정점의 수 k) (정점 번호) ... k개	3 0 1 2
...	3 1 3 4
(면 nF−1 구성 정점의 수 k) (정점 번호) ... k개	3 2 4 5

간단한 메시 데이터 포맷


{ v0 , v1 , v2 , v3 , v4 , v5 }
정점의 개수: 6개
데이터
6
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
3
3 0 1 2
3 1 3 4
3 2 4 5
T0 = {v0 v1 v2 }
T1 = {v1 v3 v4 }
T2 = {v2 v4 v5 }
v0
v1
v2
v3
v4
v5
T0
T1
T2
면의 개수: 3개
{T0 , T1 , T2 }
각 면은 세 개의 정점으로 구성됨

메쉬를 읽는 클래스

```
class MeshLoader:
    def __init__(self):
      ...
    def loadData(self, filename) :
      ...
    def draw(self):
       ...
```

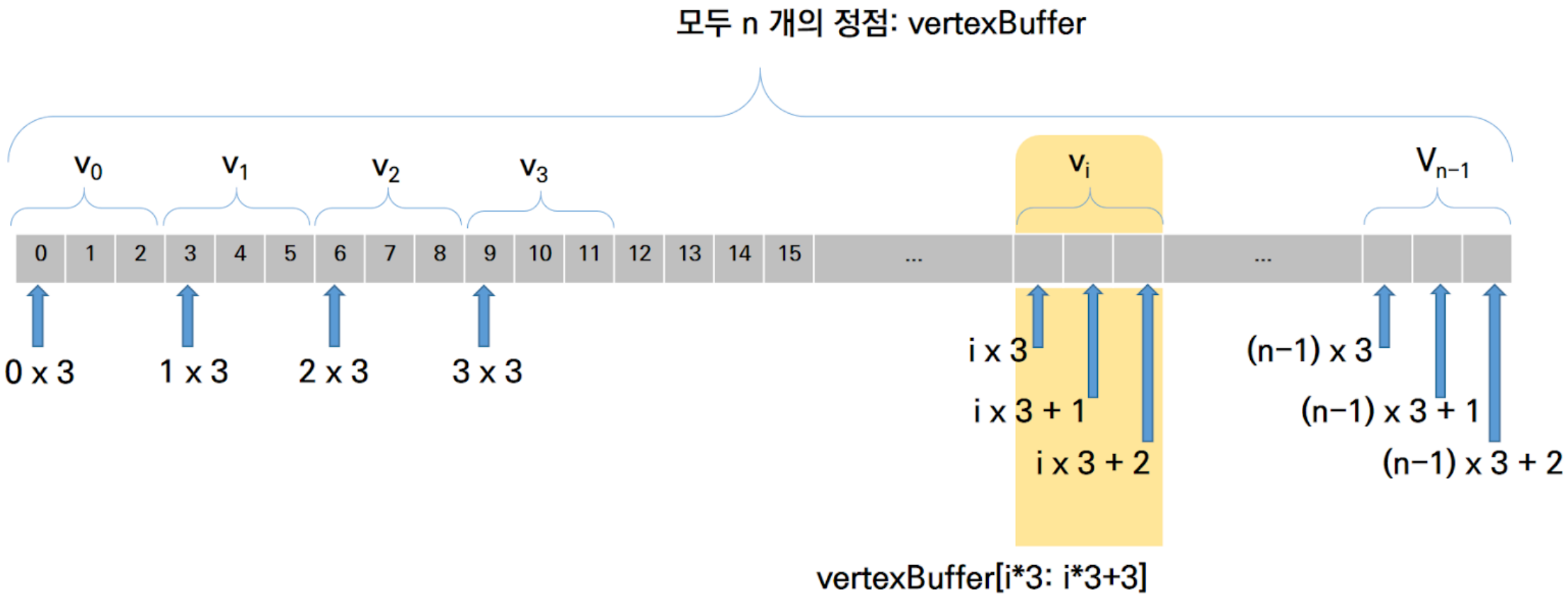
메쉬를 읽는 클래스 – 생성자

```
def __init__(self):
    self.nV = 0
    self.nF = 0
    self.vertexBuffer = None
    self.idxBuffer = None
```

메쉬를 읽는 클래스 – 데이터 읽기 준비 (버퍼준비)

```
def loadData(self, filename) :
    with open(filename, "rt") as inputfile:

        self.nV = int(next(inputfile)) # 정점의 개수
        self.vertexBuffer = np.zeros(shape = (self.nV*3, ),
                                     dtype=float)
```

메쉬를 읽는 클래스 – 데이터 채우기



메쉬를 읽는 클래스 – 데이터 읽기 메소드

정점 읽기의 완성

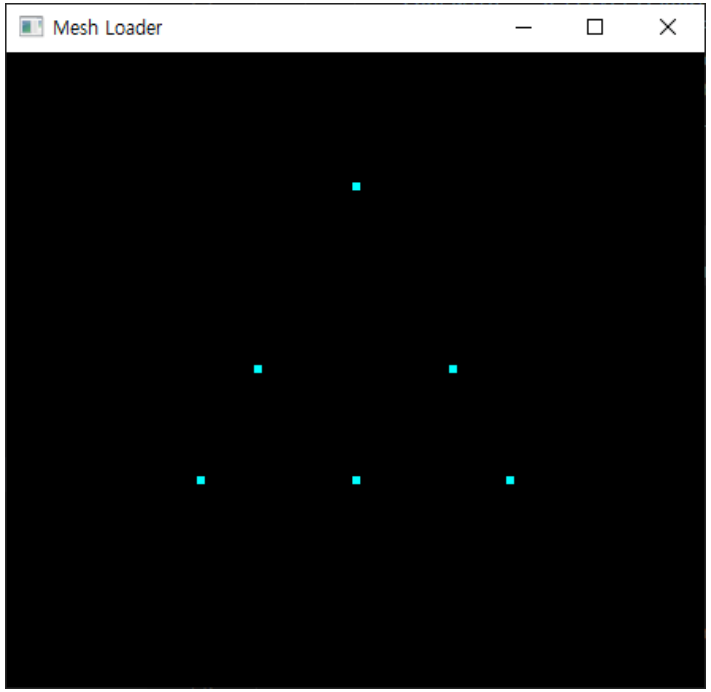
```
def loadData(self, filename) :
    with open(filename, "rt") as inputfile:

        self.nV = int(next(inputfile)) # 정점의 개수
        self.vertexBuffer = np.zeros(shape = (self.nV*3, ), dtype=float)

        for i in range(self.nV) :
            self.vertexBuffer[i*3: i*3+3] = next(inputfile).split()
            # 한 줄을 읽어 공백으로 분리 저장

        coordMin = self.vertexBuffer.min()
        coordMax = self.vertexBuffer.max()
        scale = max([coordMin, coordMax], key=abs)
        self.vertexBuffer /= scale
```

메쉬 그리기 – 정점 출력

```
def draw(self):
    glBegin(GL_POINTS)
    for i in range(self.nV):
        glVertex3fv(self.vertexBuffer[i*3:i*3+3])
    glEnd()
```

메쉬 읽고 그리기 – MeshLoader 활용

```
class MyGLWidget(QOpenGLWidget):

    ...

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

        self.meshLoader = MeshLoader()
        self.meshLoader.loadData('mesh.txt')

    ...
    def paintGL(self):
        ...

        gluLookAt(2,0.5,2, 0,0,0, 0,1,1)
        self.meshLoader.draw()
```



면 정보 읽기

```
def loadData(self, filename) :
    with open(filename, "rt") as inputfile:

        ...


        self.nF = int(next(inputfile))
        self.idxBuffer = np.zeros(shape = (self.nF*3, ), dtype = int)
```

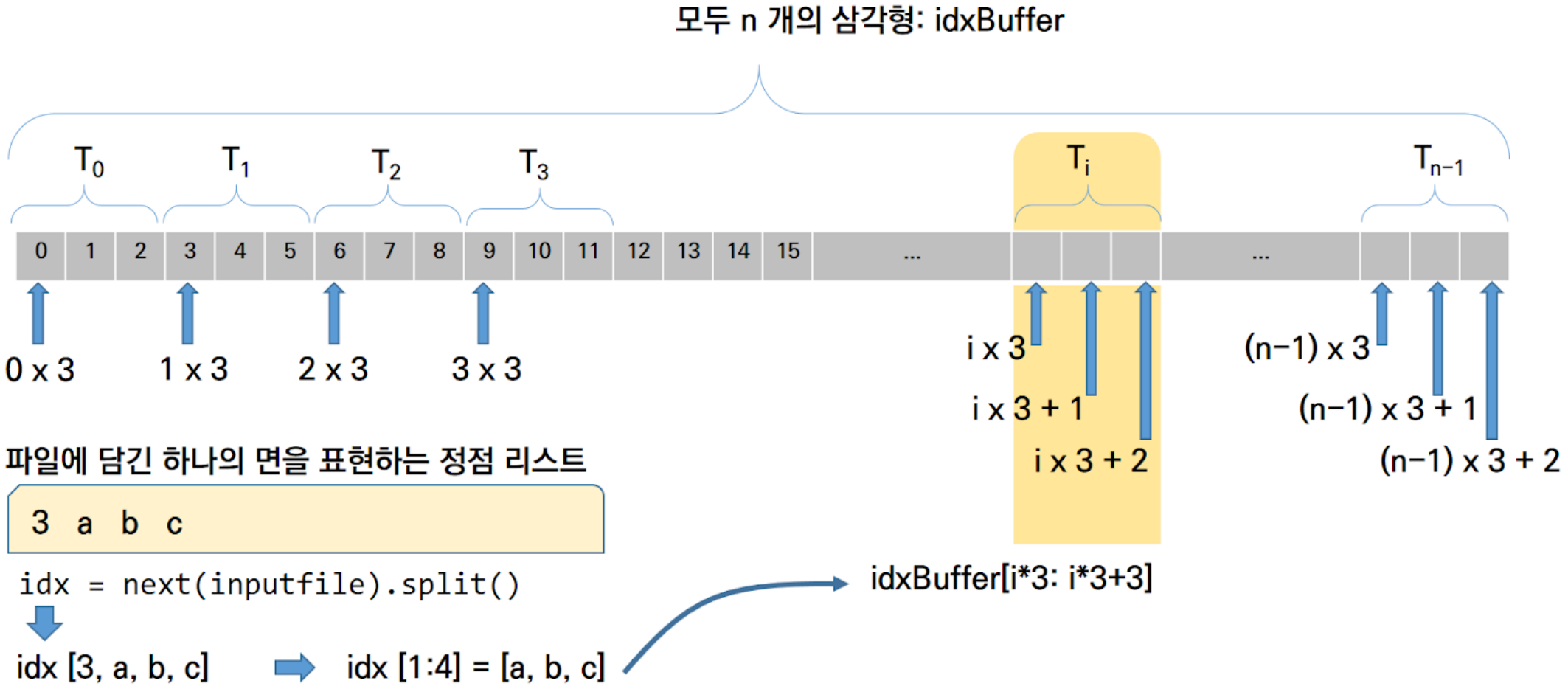
하나의 면을 구성하는 정점의 개수가 3 개가 아니고 4 개나 더 많을 수도 있지만, 설명을 간단히 하기 위해 3개의 정점, 즉 삼각형으로 각각의 면이 이루어진다고 가정하자. 그러면 면의 개수가 nF 개이며, 필요한 정점은 nF*3 개가 된다

면 정보 읽기

```
def loadData(self, filename) :
    with open(filename, "rt") as inputfile:

        ...

        for i in range(self.nF):        # 삼각형의 개수
            idx = next(inputfile).split()
            self.idxBuffer[i*3: i*3+3] = idx[1:4]
            # 한 줄을 읽어 공백으로 분리해 첫 원소 제외하고 저장
```

하나의 면을 구성하는 정점의 개수가 3 개가 아니고 4 개나 더 많을 수도 있지만, 설명을 간단히 하기 위해 3개의 정점, 즉 삼각형으로 각각의 면이 이루어진다고 가정하자. 그러면 면의 개수가 nF 개이며, 필요한 정점은 nF*3 개가 된다

면 정보 읽기


모두 n 개의 삼각형: idxBuffer
T0
T1
T2
T3
Ti
Tn-1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ... ...
0 x 3
1 x 3
2 x 3
3 x 3
i x 3
i x 3 + 1
i x 3 + 2
(n−1) x 3
(n−1) x 3 + 1
(n−1) x 3 + 2
파일에 담긴 하나의 면을 표현하는 정점 리스트
3 a b c
idx = next(inputfile).split()
idx [3, a, b, c]
idx [1:4] = [a, b, c]
idxBuffer[i*3: i*3+3]

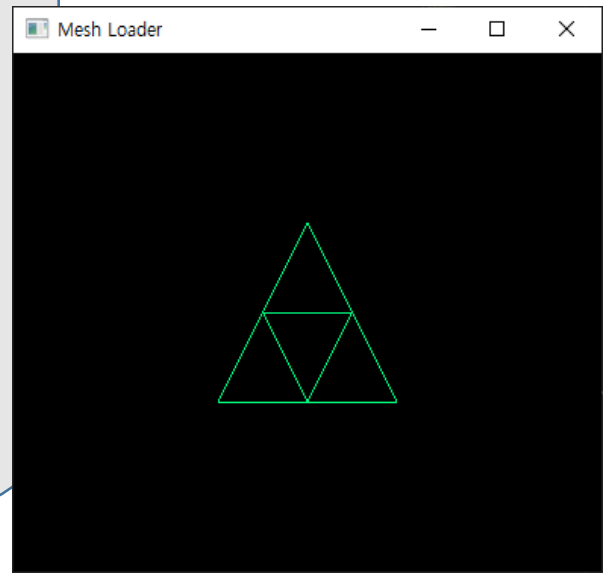
메쉬 로드 함수 완성

```
def loadData(self, filename) :
    with open(filename, "rt") as inputfile:

        self.nV = int(next(inputfile)) # 정점의 개수
        self.vertexBuffer = np.zeros(shape = (self.nV*3, ),
                                     dtype=float)

        for i in range(self.nV) :
            self.vertexBuffer[i*3: i*3+3] = next(inputfile).split()
            # 한 줄을 읽어 공백으로 분리 저장

        coordMin = self.vertexBuffer.min()
        coordMax = self.vertexBuffer.max()
        scale = max(coordMin, coordMax)
        self.vertexBuffer /= scale


        self.nF = int(next(inputfile))
        self.idxBuffer = np.zeros(shape = (self.nF*3, ), dtype = int)

        for i in range(self.nF):        # 삼각형의 개수
            idx = next(inputfile).split()
            self.idxBuffer[i*3: i*3+3] = idx[1:4]
            # 한 줄을 읽어 공백으로 분리해 첫 원소 제외하고 저장
```

그리기 함수의 개선

각 면(삼각형)을 GL_LINE_LOOP을 이용하여 그린다.
면을 구성하는 정점 인덱스 3개: v0, v1, v2
- 이 인덱스를 이용하여 좌표를 버텍스 버퍼에서 찾아 온다: vertexBuffer[v0*3: v0*3+3], …

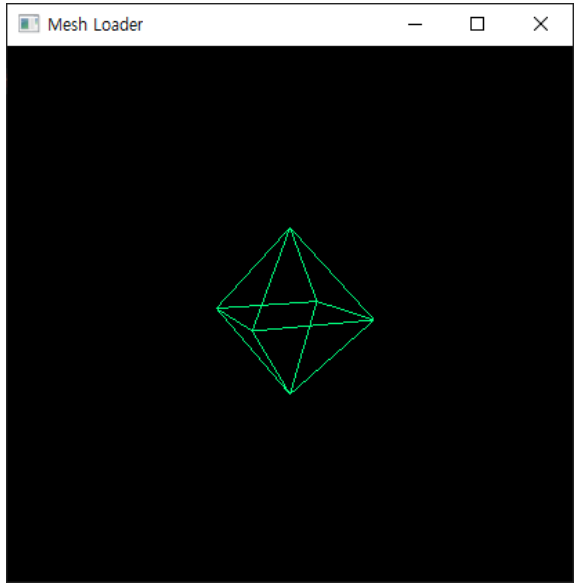
```
def draw(self):
    glColor3f(0,1,0.5)
    for i in range(self.nF):
        glBegin(GL_LINE_LOOP)
        vIdx = self.idxBuffer[i*3: i*3+3]
        v0, v1, v2 = vIdx[0], vIdx[1], vIdx[2]
        glVertex3fv(self.vertexBuffer[v0*3: v0*3+3])
        glVertex3fv(self.vertexBuffer[v1*3: v1*3+3])
        glVertex3fv(self.vertexBuffer[v2*3: v2*3+3])
        glEnd()
```



입체 객체 읽기

메시 파일 이름만 바꾸면 됨

```
    def initializeGL(self):
        glColor3f(0,1,1)
        glPointSize(5)
        self.myLoader.loadData('./Lab06_Meshes/octahedron.txt')
```



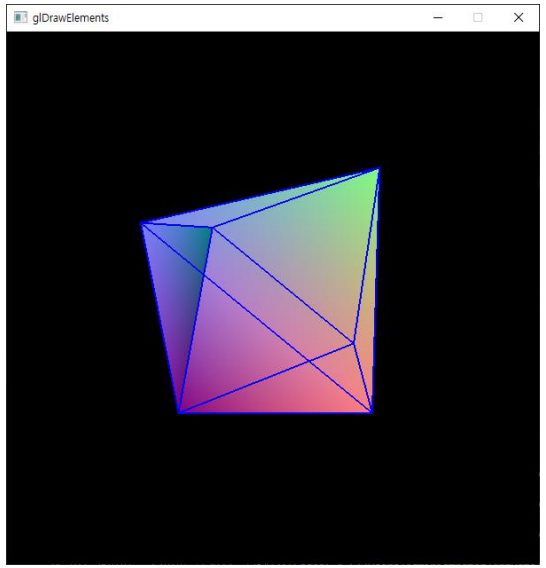
octahedron.txt

```
6
1 0 0
0 -1 0
-1 0 0
0 1 0
0 0 1
0 0 -1
8
3 4 0 1
3 4 1 2
3 4 2 3
3 4 3 0
3 5 1 0
3 5 2 1
3 5 3 2
3 5 0 3
```

입체 면 그리기 – GL_TRIANGLES

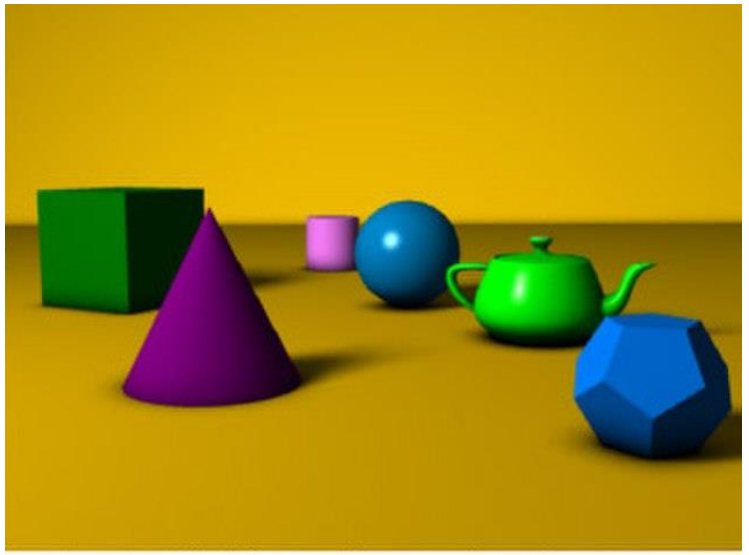
- 각 면을 구성하는 정점의 좌표 [-1, 1]
  - 이 좌표를 색상으로 이용하여 그려보자
    - 색상의 범위는 [0, 1] : 좌표에 1을 더하여 2로 나누면 이 범위에 들어옴

```
glBegin(GL_TRIANGLES)
for i in range(self.nF):
    vIdx = self.idxBuffer[i*3: i*3+3]
    v0, v1, v2 = vIdx[0], vIdx[1], vIdx[2]
    glColor3fv((self.vertexBuffer[v0*3: v0*3+3]+np.array([1]))/2)
    glVertex3fv(self.vertexBuffer[v0*3: v0*3+3])
    glColor3fv((self.vertexBuffer[v1*3: v1*3+3]+np.array([1]))/2)
    glVertex3fv(self.vertexBuffer[v1*3: v1*3+3])
    glColor3fv((self.vertexBuffer[v2*3: v2*3+3]+np.array([1]))/2)
    glVertex3fv(self.vertexBuffer[v2*3: v2*3+3])
glEnd()
```

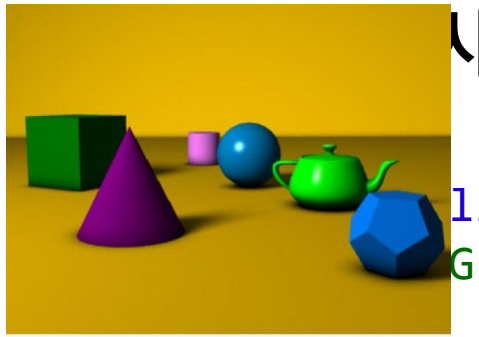


입체 면 그리기

- 깊이 버퍼 사용하기
  - z-버퍼라고도 불림
  - 영상을 생성할 때에 사용되는 여러 버퍼 가운데 하나
  - 대표적인 두 버퍼는 색상 버퍼와 깊이 버퍼
  - 색상 버퍼는 위쪽과 같이 우리 눈에 관찰되는 이미지
  - 색상 버퍼에 픽셀을 그릴 것인지 말 것인지를 결정
  - 픽셀의 깊이값을 참고해야 함
  - 새로운 픽셀이 기록된 깊이에 비해 더 얕은 곳에 있으면
    - 픽셀을 그리는 일이 실제로 이루어짐
    - 이 경우 색상 버퍼와 깊이 버퍼의 값이 바뀜

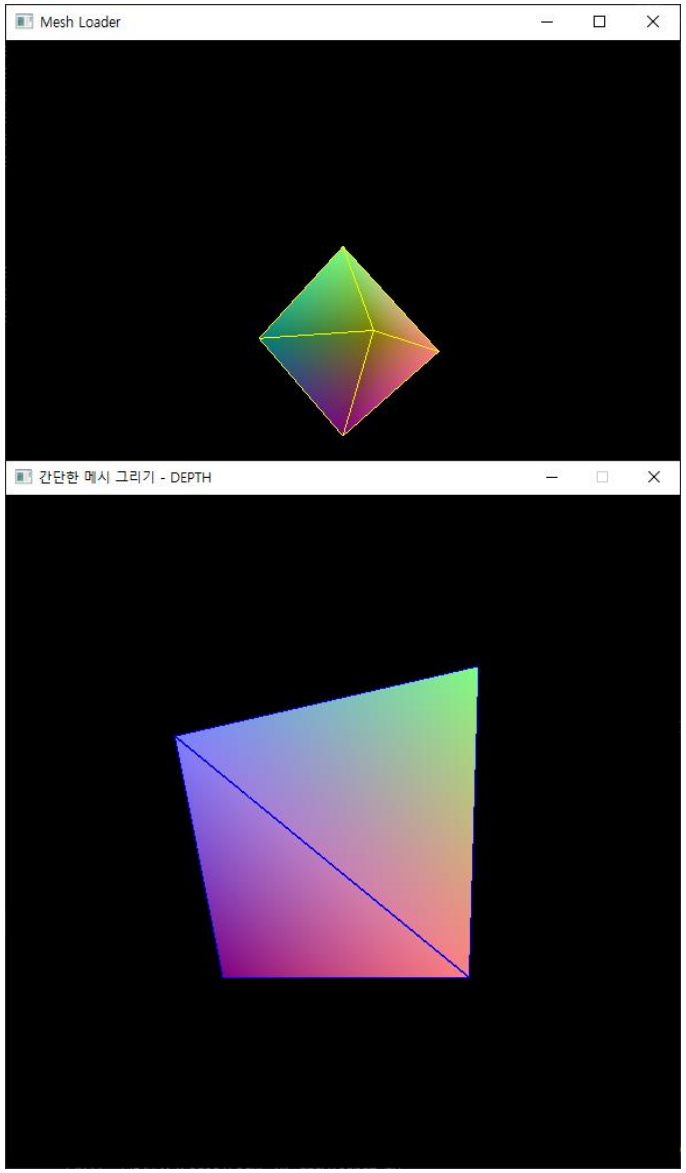


입체 면 그리기



사용하기

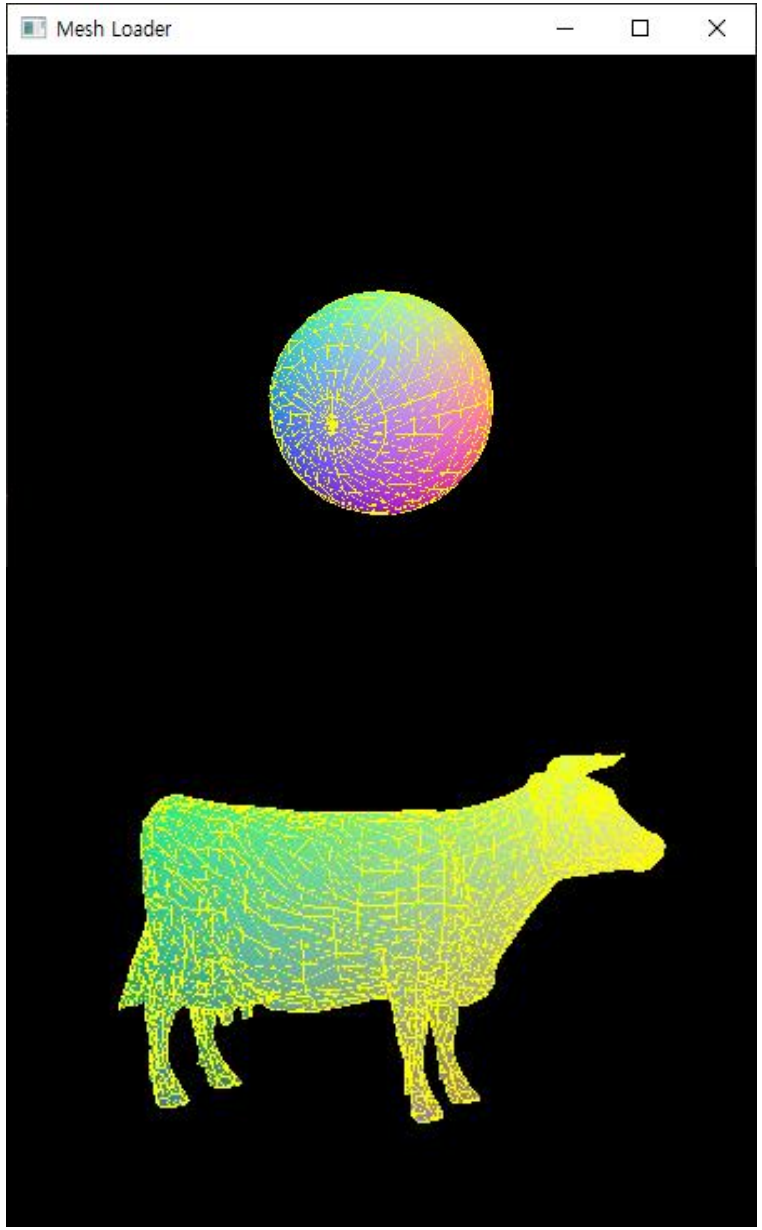
```
lizeGL(self):
GL 그리기를 수행하기 전에 각종 상태값을 초기화

le(GL_DEPTH_TEST)

L(self):

r(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
...
```



깊이 버퍼 자동 사용

- 현재 pyQt6의 QOpenGLWidget은 깊이 버퍼 자동 사용
  - glEnable(GL_DEPTH_TEST)
    - 깊이 버퍼를 활용하여 깊이 검사를 하도록 함

- 여러 파일을 사용해 보자
  https://github.com/dknife/opendata/raw/main/mesh/sphere.txt
  https://github.com/dknife/opendata/raw/main/mesh/cow.txt



메시를 움직여 보기

- 타이머 사용하여 OpenGL Update
  - paintGL 호출됨

```
class MyWindow(QMainWindow):
    def __init__(self, title=''):
        super().__init__()
        self.setWindowTitle(title)
        ## OpenGL Widget 달기
        self.glWidget = MyGLWidget()
        self.setCentralWidget(self.glWidget)

        self.timer = QTimer(self)
        self.timer.setInterval(1)
        self.timer.timeout.connect(self.timeout)
        self.timer.start()

    def timeout(self):
        self.glWidget.update()
```

메시를 움직여 보기

- 각도를 변경하고, 물체를 회전시킴

```
class MyGLWidget(QOpenGLWidget):
...
    def initializeGL(self):
        ...
        self.angle = 0
...
    def paintGL(self):

        glRotatef(self.angle, 0, 1, 0)
        self.myLoader.draw()
        self.angle += 1
```

너무 느려!!!!!

속도 개선

- 사용 가능한 것들
  - Display List
  - glDrawArrays
  - glDrawElements

Display list

```
    def initializeGL(self):
        ...
        self.myLoader.loadData('./Lab06_Meshes/cow.txt')

        self.drawList = glGenLists(1)
        glNewList(self.drawList, GL_COMPILE)
        self.myLoader.draw()
        glEndList()

        self.angle = 0
```

```
    def paintGL(self):

        ...

        glRotatef(self.angle, 0, 1, 0)
        #self.myLoader.draw()
        glCallList(self.drawList)
        self.angle += 1
```

메시 로더에 모두 구현하는 법

```
def draw(self):
    glColor3f(1,1,1)
    glBegin(GL_TRIANGLES)
    for i in range(self.nF):
        vIdx = self.idxBuffer[i*3: i*3+3]
        v0, v1, v2 = vIdx[0], vIdx[1], vIdx[2]
        glColor3fv((self.vertexBuffer[v0*3: v0*3+3]+np.array([1]))/2)
        glVertex3fv(self.vertexBuffer[v0*3: v0*3+3])
        glColor3fv((self.vertexBuffer[v1*3: v1*3+3]+np.array([1]))/2)
        glVertex3fv(self.vertexBuffer[v1*3: v1*3+3])
        glColor3fv((self.vertexBuffer[v2*3: v2*3+3]+np.array([1]))/2)
        glVertex3fv(self.vertexBuffer[v2*3: v2*3+3])
    glEnd()
    glColor3f(0,0,1)
    for i in range(self.nF):
        glBegin(GL_LINE_LOOP)
        vIdx = self.idxBuffer[i*3: i*3+3]
        v0, v1, v2 = vIdx[0], vIdx[1], vIdx[2]
        glVertex3fv(self.vertexBuffer[v0*3: v0*3+3])
        glVertex3fv(self.vertexBuffer[v1*3: v1*3+3])
        glVertex3fv(self.vertexBuffer[v2*3: v2*3+3])
        glEnd()
```

```
    def make_displayList(self):
        self.list = glGenLists(1)
        glNewList(self.list, GL_COMPILE)
        self.draw()
        glEndList()

    def draw_list(self):
        glCallList(self.list)
```

정점 배열을 사용하는 방법

- 앞 장에서 배운 내용을 활용
- glDrawArrays
- glDrawElements