# 컴퓨터애니메이션실습(CAL)

## HW1_Mesh_and_Eigen

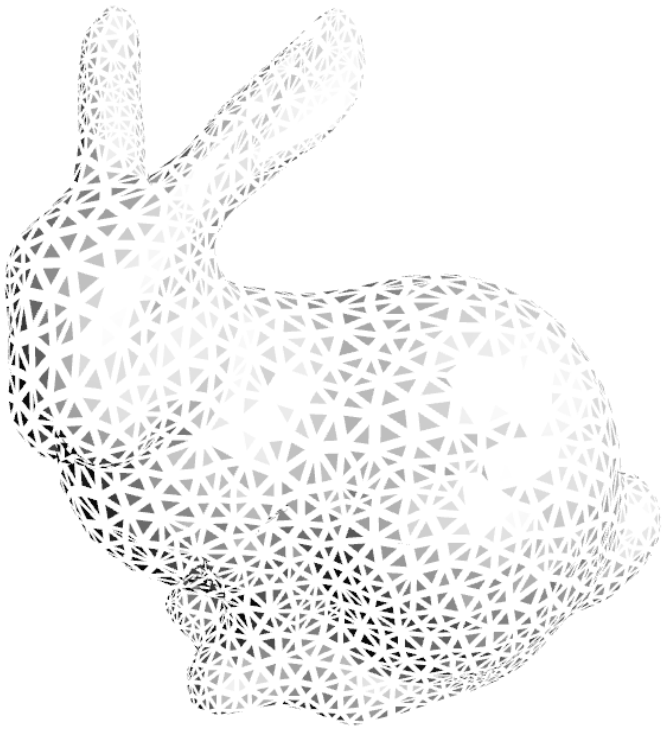

**Self-scoring table**

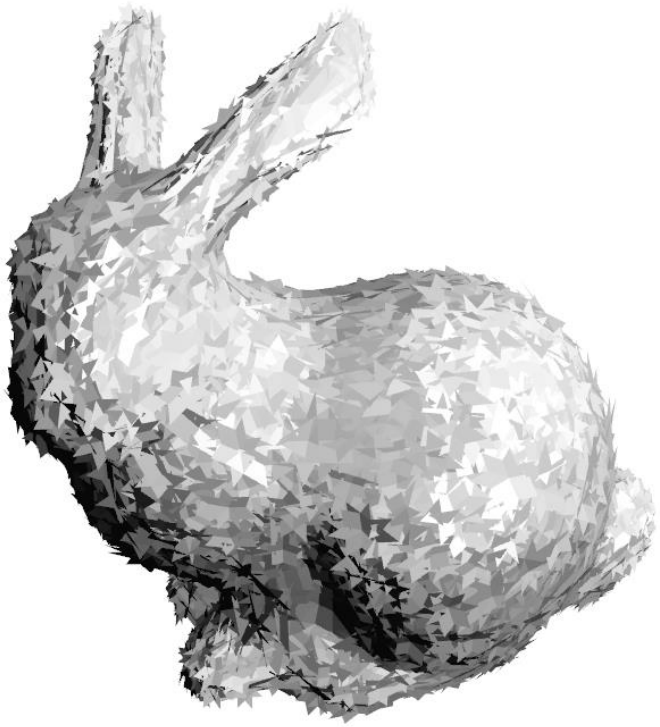
| | P1 | E1 | Total |
|---|---|---|---|
| **Score** | **1** | **1** | **2** |

2018707068 김경환

KwangWoon University

Practice 01. Build and draw a mesh with shrunken faces:





우선 왼쪽, 오른쪽과 같은 shrunken face의 경우에는 삼각형의 각 세 점을 삼각형의 무게중심 방향으로 이동시켜 새로운 삼각형을 그리는 식으로 만들 수 있다.

이는 buildShrunkenFaces 함수에서 const Vector3f& p = vertex.col(face(j, i)); faceVertex.col(3 * i + j) = p + gap * (center - p); 를 통해 만들어진다.

여기서 vertex.col(face(j, i))의 face(j, i)는 삼각형을 이루는 세 점을 표현하기 위해 0-2 사이값을 가지는 j와 각 삼각형을 표현하기 위해 0-face.cols() 즉, 면의 총 개수의 사이값을 가지는 i를 사용하였다.

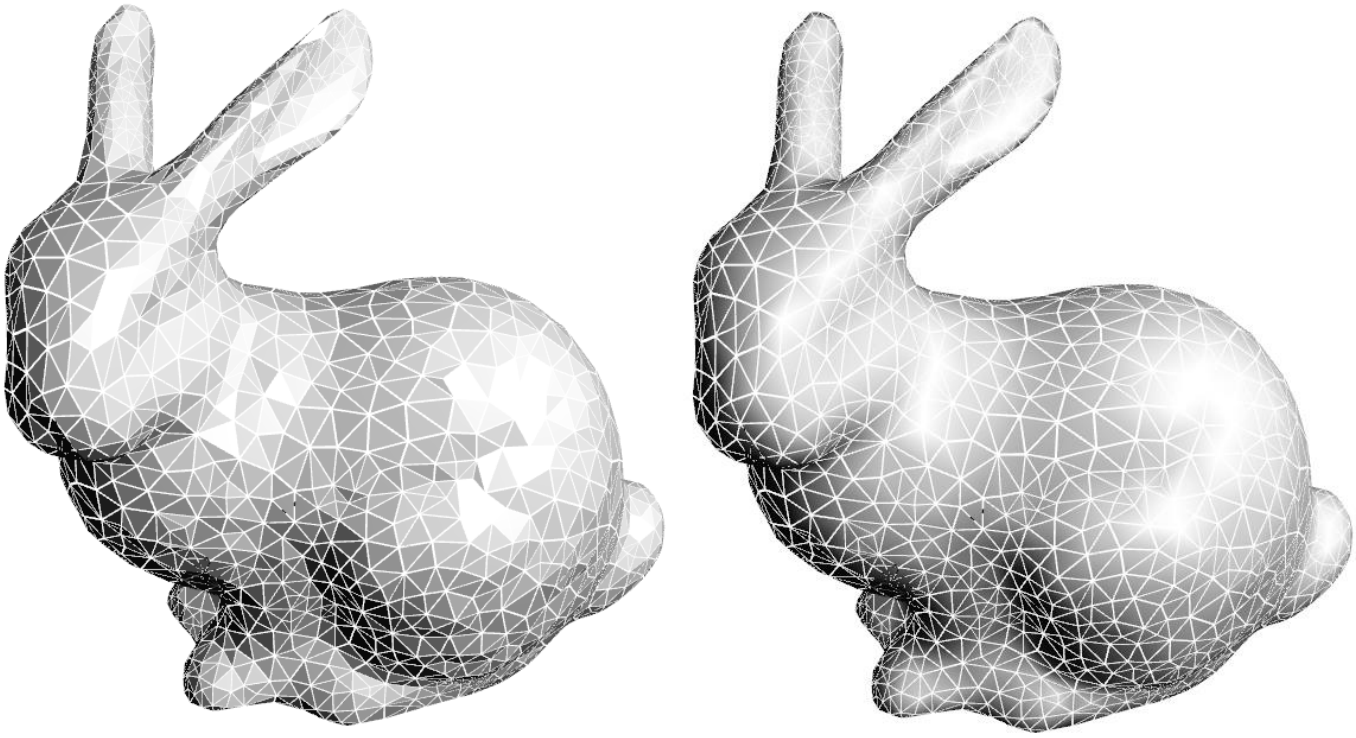
이는 m01_bunny.off 파일에서 한 삼각형을 이루는 세 점을 face의 한 열로 갖게 했으므로 face(j, i)는 i 번째 삼각형의 j번째 점을 의미한다.

따라서 기존 삼각형의 한 점에서 무게중심으로 향하는 벡터를 gap 만큼을 곱해 해당 점에 더함으로써 shrunken face의 한 점을 새로 만든다.

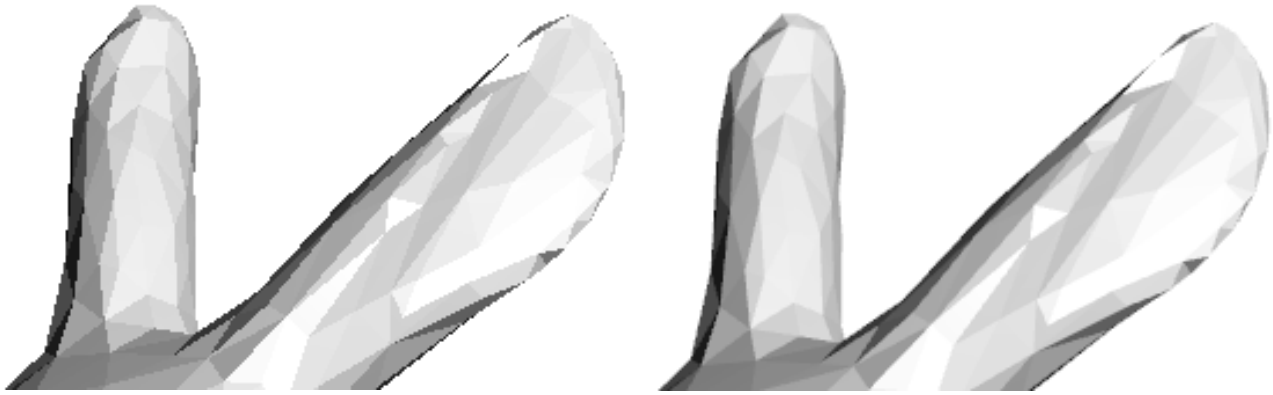
그리고 shrunken face는 각 face마다 3개의 점을 다른 face와 중복되지 않게 가져야 함으로 faceVertex의 col을 3 * face.cols() 만큼 지정하고 3 * i + j을 사용하여 shrunken face를 이루는 세 점이 연속되도록 저장한다.

왼쪽의 경우는 gap = min(gap + 0.05f, 0.5f) 최소값으로 설정해놓은 0.5에 의해 범위내에서 그릴 수 있는 가장 작은 shrunken faces이다.

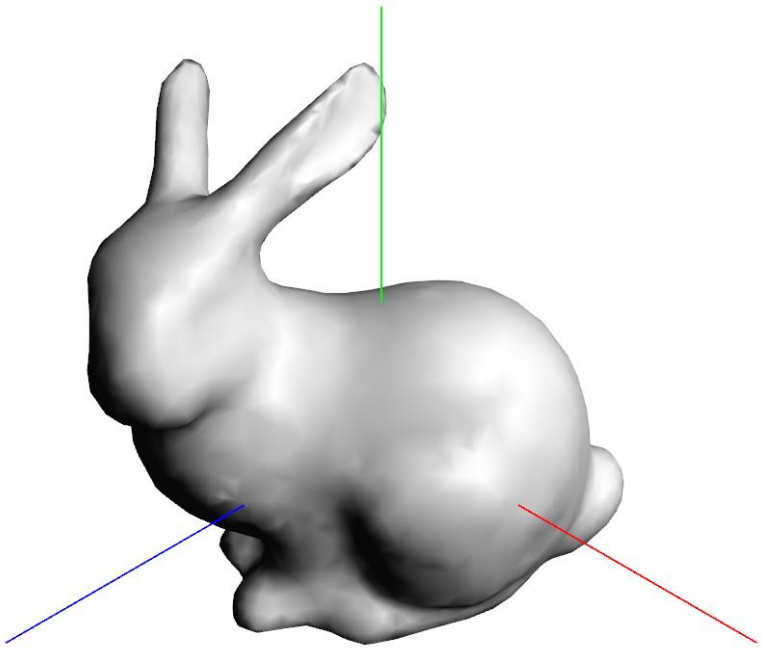
오른쪽의 경우는 최소값의 한계가 없으므로 삼각형의 세 점이 무게중심에서부터 뻣어나가게 그려진다.

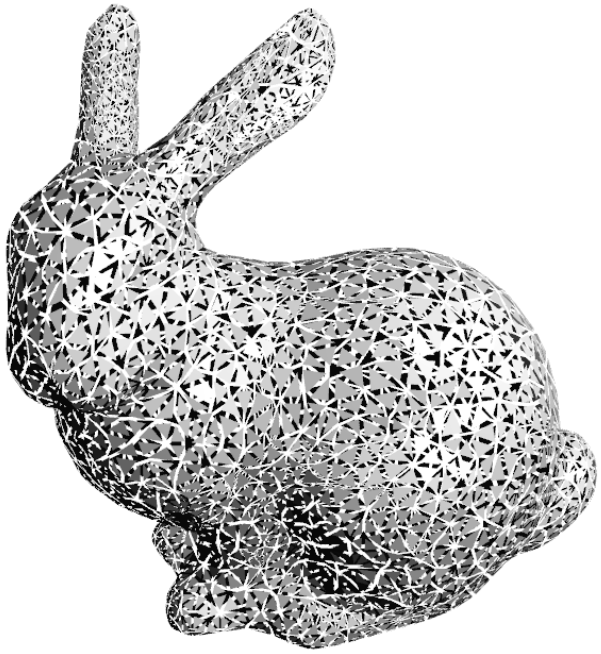


useFaceNormal의 이름을 가진 bool type 변수를 통해 rendering할 때 face normal vector를 사용할 것인지 vertex normal vector를 사용할 것인지 설정하여 왼쪽은 face normal vector를 사용한 flat shading이 그려졌고, 오른쪽은 vertex normal vector를 사용한 smooth shading이 그려졌다.



aaEnabled의 이름을 가진 bool type 변수를 통해 glEnable(GL_MULTISAMPLE)을 할 것인지 glDisable(GL_MULTISAMPLE)을 할 것인지 설정하여 왼쪽은 glDisable(GL_MULTISAMPLE)하여 antialiasing이 되지 않아 윤곽선이 계단처럼 보이는 aliasing이 일어났고 오른쪽은 glEnable(GL_MULTISAMPLE)하여 antialiasing이 되어 윤곽선이 매끄럽게 그려진 것을 볼 수 있다.

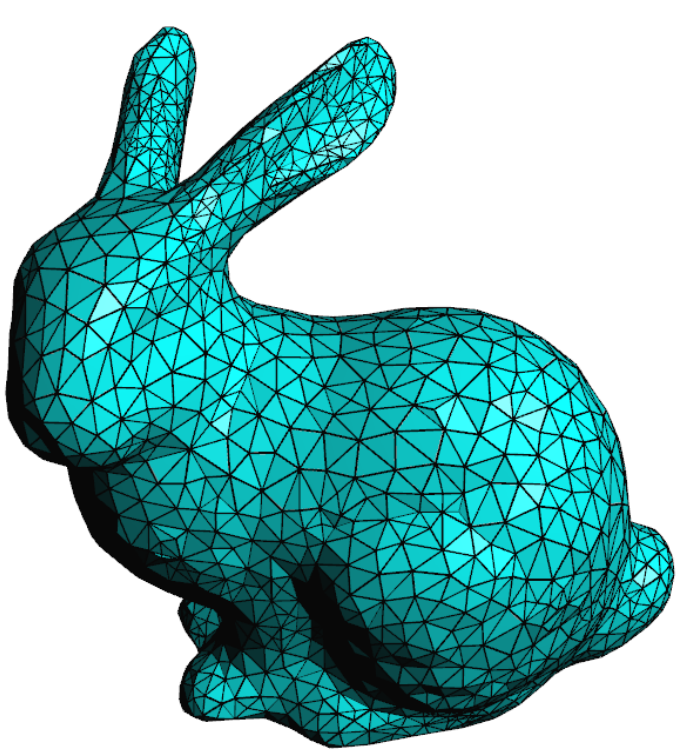


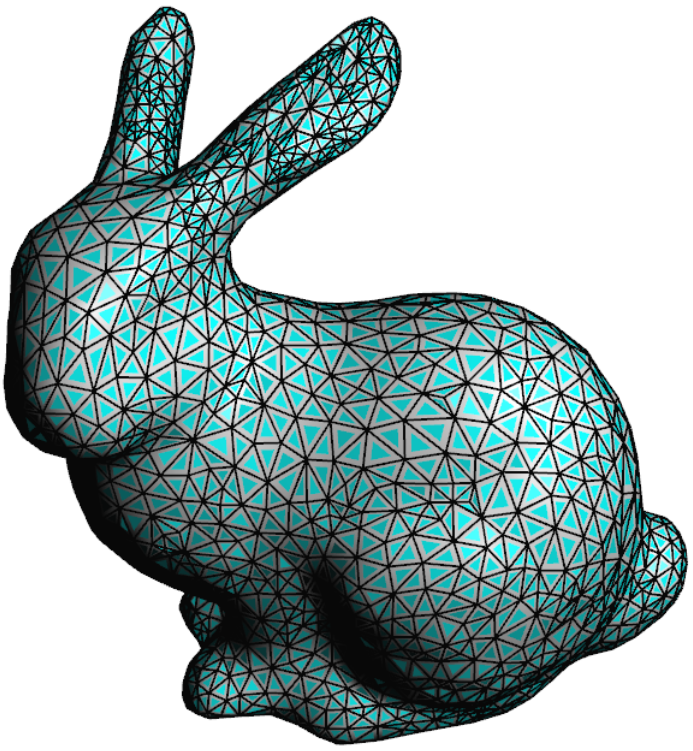


왼쪽은 Axes의 이름을 가진 bool type 변수를 통해 drawAxes()를 실행하여 축을 그린 모습이다.

오른쪽은 bfcEnabled의 이름을 가진 bool type 변수르 통해 back face culling을 꺼서 buuny의 뒤에 그려지는 shrunken faces도 같이 그려진 것을 볼 수 있다.

**Exercise 01.** Overlay the shrunken faces on the original wireframed faces with offsetting:





우선 위 그림은 GL_TRIANGLES로 bunny model을 회색으로 그리고 그 위에 GL_LINES로 wireframe을 검은색으로 그렸다. 또 그 위에 GL_TRIANGLES로 shrunken face를 청록색으로 그린 모습이다.

그리고 wireframe를 overlay할 때 오류없이 올바르게 그려지도록 glEnable(GL_POLYGON_OFFSET_FILL)과 glPolygonOffset(1.0, 1.0)를 사용하였다.

Wireframe을 그리는 과정의 경우는 practice에서 주어지지 않았는데 이는 readMesh()를 수정하여 매개변수로 <Vector<Vector<int>> edge를 추가하였다.

이러한 edge는 각 vertex에 대하여 인접한 vertex를 저장할 것이므로 nVertices만큼 vertex의 좌표를 입력받는 반복문에서 빈 vector를 추가하도록 하였다. 그리고 각 vertex에 인접한 vertex를 찾는 과정으로는 삼각형을 이루는 세 vertex의 번호가 주어질 때 큰 vertex의 번호를 기준으로 작은 vertex의 번호가 인접해있는 것으로 저장하였다. 이는 먼저 삼각형의 세 변을 모두 표현하기 위해 0-2 사이값인 j와 k = (j + 1) % 3에 해당하는 k를 사용하였다. 그리고 face(j, i) > face(k, i) 즉, i번째 삼각형의 j번째 vertex의 번호가 i번째 삼각형의 k번째 vertex의 번호보다 크고 !count(edge[face(j, i)].begin(), edge[face(j, i)].end(), face(k, i)) 즉, i번째 삼각형의 j번째 vertex와 인접한 vertex들 중에 i번째 삼각형의 k번째 vertex가 없다면 edge[face(j, i)].push_back(face(k, i)); 즉, i번째 삼각형의 k번째 vertex를 추가하도록 하였다.

반대의 경우인 face(k, i) > face(j, i)도 똑같이 수행해주면 된다.

마지막으로 original face, wireframe, shrunken face가 서로 다른 색을 갖고 있는 것을 볼 수 있는데 이는 setupColoredMaterial()를 통해 RGB값을 저장하고 있는 Vector3f의 값을 다르게 부여하여 구현하였다. Wireframe은 검은색으로 Vector3f(0.0f, 0.0f, 0.0f), original face는 회색으로 Vector3f(0.9f, 0.9f, 0.9f), shrunken face는 청록색으로 Vector3f(0.033f, 1.0f, 1.0f)를 부여하였다.