컴퓨터그래픽스 (CG)

HW01_Drawing_a_Torus

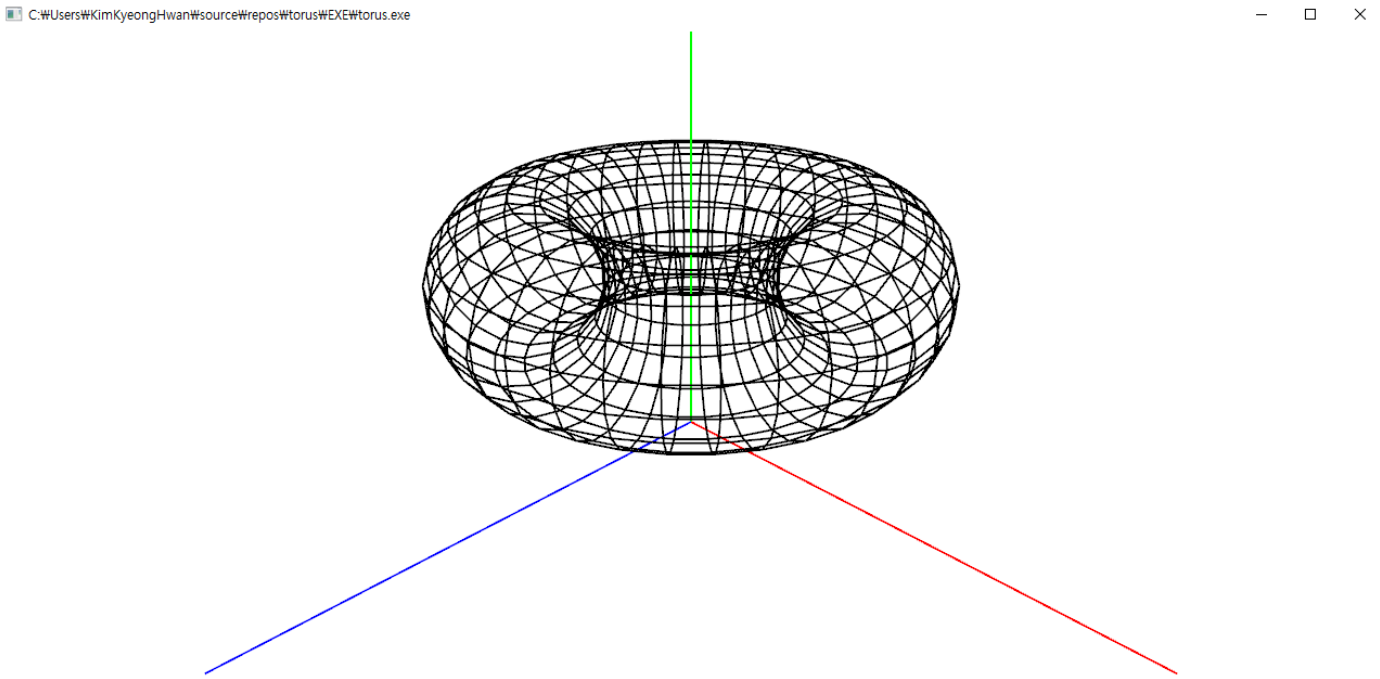
Self-scoring table

	1	2	3	4	Total
Score	**1**	**1**	**1**	**1**	**4**

2018707068 김경환

KwangWoon University

1. Draw the wireframe only:

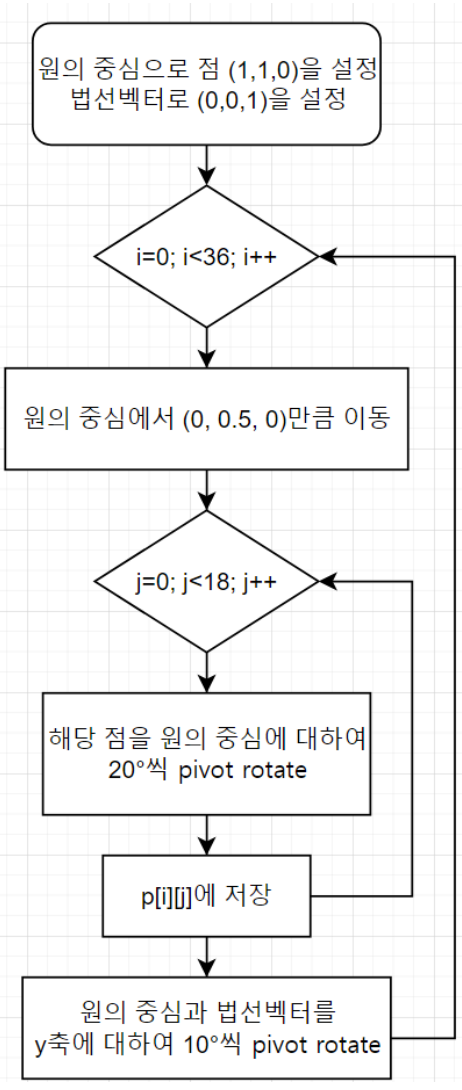


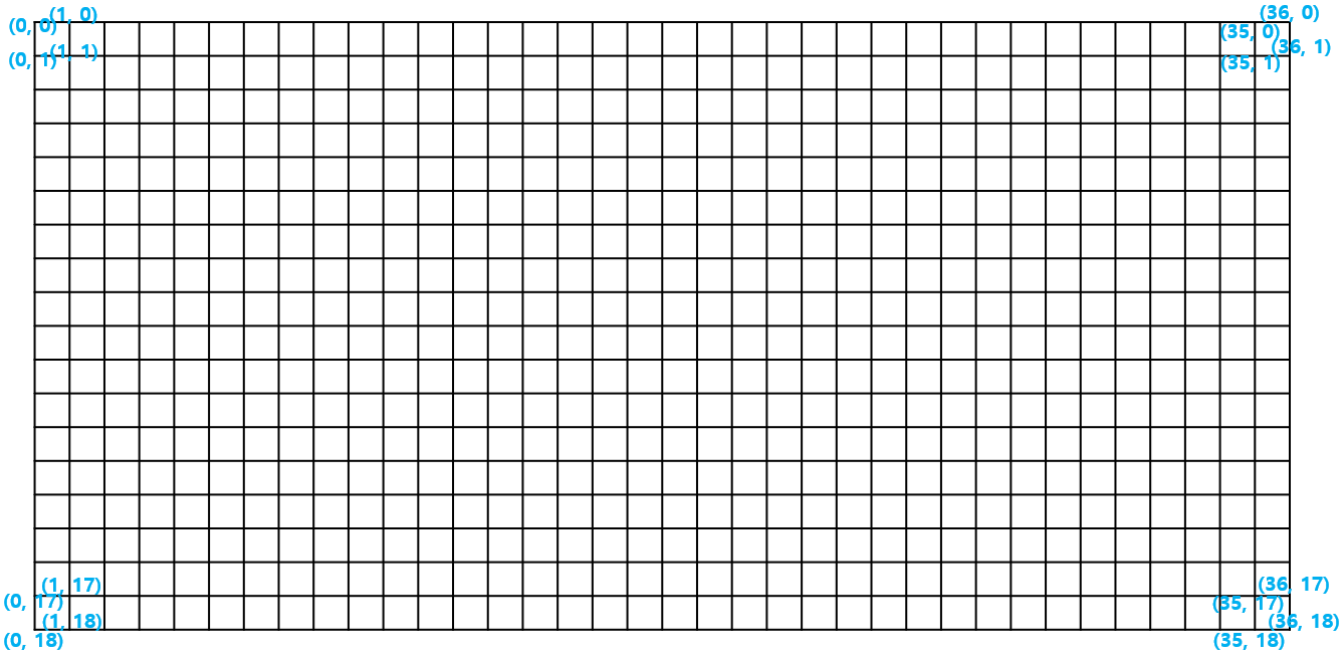
위와 같은 Torus의 wireframe을 그리기 위해서 우선 (1, 1, 0)을 원의 중심으로 갖고 반지름이 0.5이며, 법선벡터가 (0, 0, 1)인 평면원 하나를 생각하였다.

그리고 이러한 원을 18개의 curve point로 그리기 위해서 임의의 점을 원의 중심인 (1, 1, 0)에서 y축 방향으로 0.5만큼 이동하고 원의 중심에 대하여 20°씩 pivot rotate를 18번 실행하여 해당 결과를 36 by 18 2차원 배열에 저장하였다.

또한, 이렇게 그린 원 하나를 사용하여 Torus를 그리기 위해서 원의 중심이었던 점(1, 1, 0)과 법선벡터였던 (0, 0, 1)을 y축에 대하여 10°씩 rotate를 실행하며 위 18개의 curve point로 이루어진 원을 구하는 과정을 총 36번 반복하였다.

그리하여 Torus를 이루는 모든 점들의 좌표가 36 by 18의 2차원 배열에 저장되었다. 그리고 해당 배열의 각 index에서 시계방향에 따라 4개의 vertex를 부여하였고, 이를 사용하여 glBegin(GL_QUADS)와 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)로 내부가 빈 polygon 만들어 wireframe을 그릴 수 있었다.

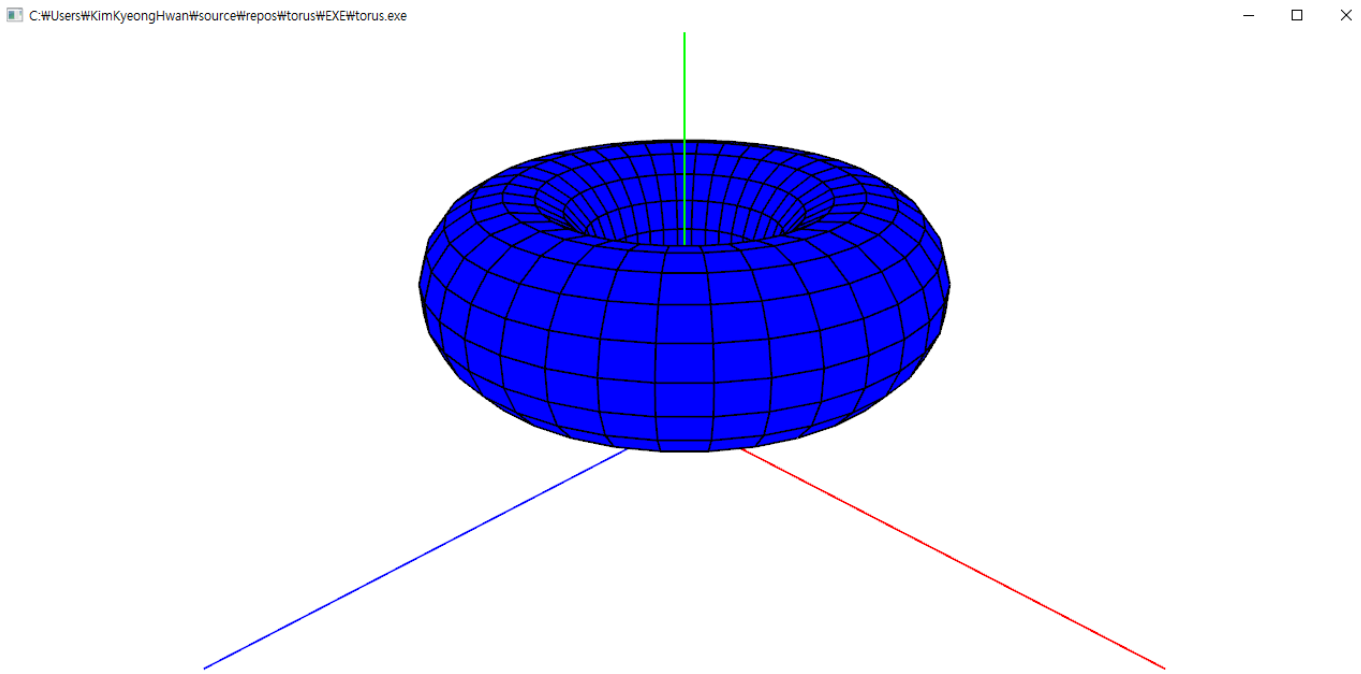




마지막으로 각 index에서 시계방향으로 vertex를 부여할 때 범위를 벗어나는 것을 주의해야한다.

직사각형의 우변과 아랫변은 index를 초과하게 됨으로 p[(i+1)%36][(j+1)/18]을 사용하여 quads가 이어지도록 설계한다.

2. Draw the quads and the wireframe:



우선 위 사진은 quads와 wireframe을 한 번에 그리면서 서로 구분되도록 색깔을 달리하여 그렸다.

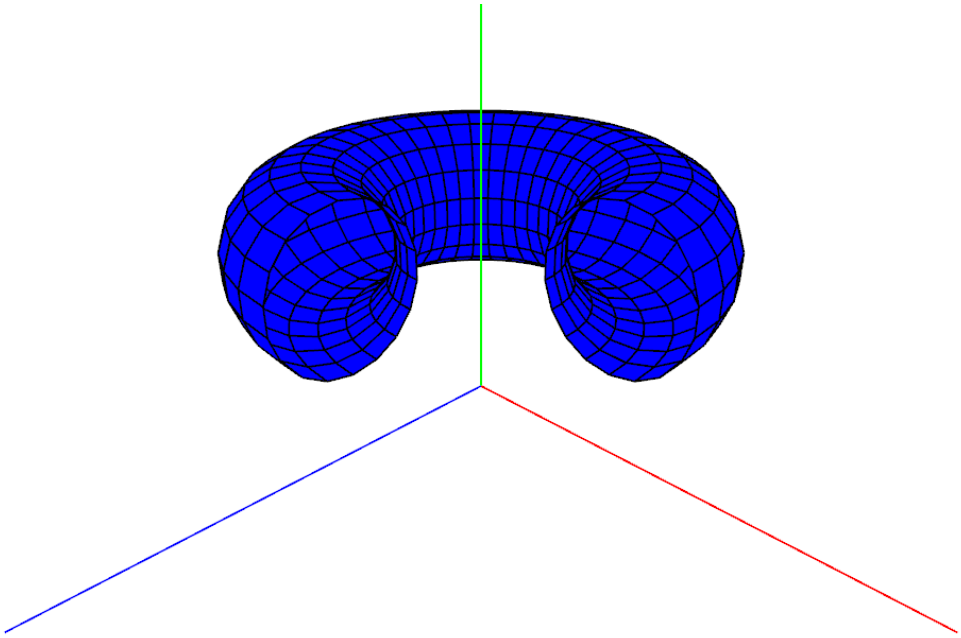
그리고 1번 과정을 통해 얻은 p[36][18]의 2차원 배열을 통해서 glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)으로 wireframe을 그리고, 그 위에 quads가 올 수 있도록 glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)과 glPolygonOffset(1.0, 1.0)을 사용하였다.

또한, 여기서 glEnable(GL_POLYGON_OFFSET_FILL)을 실행하여 GL_FILL 모드에서 렌더링되는 경우 깊이

비교가 수행되기 전에 다각형 조각의 깊이 값에 오프셋이 추가되도록 하였다.

마지막으로 wireframe과 quads 모두 glBegin(GL_QUADS)를 사용하여 그렸기 때문에 quads의 경우에도 1번 과정에서 wireframe을 그린 방법처럼 glVertex3f()을 CW에 따라 4개 부여하여 그렸다.

3. Sweep angle control around the y-axis:

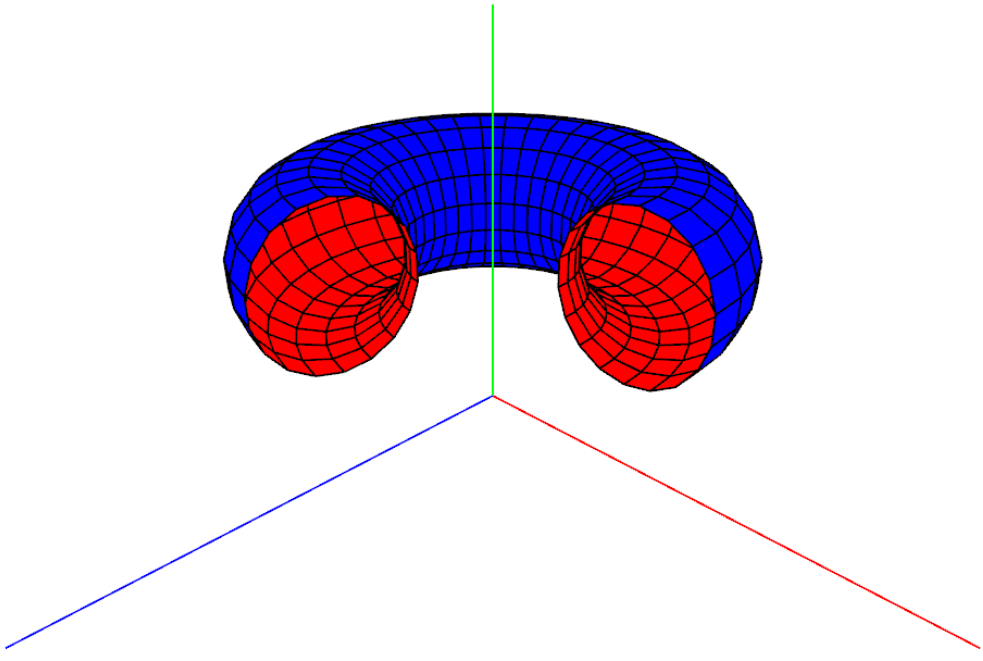


위 사진은 방향키로 y축에 대한 sweep angle control을 하는 모습이다.

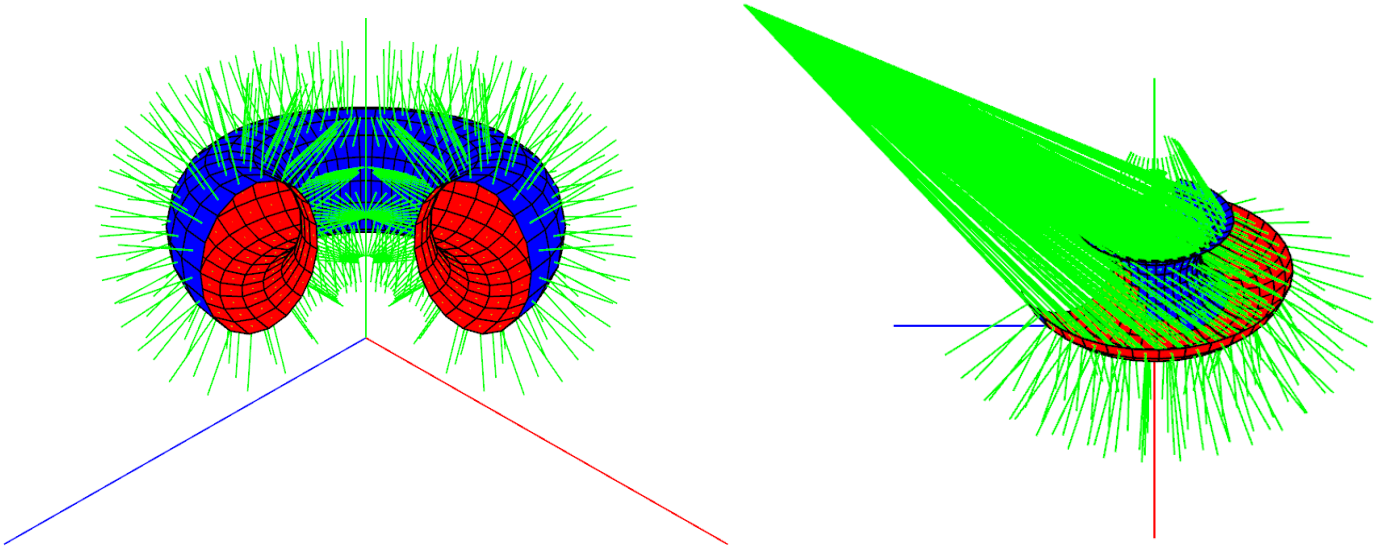
이는 Torus를 렌더링할 때 36 by 18의 2차원 배열에서 36에 해당하는 값을 sweep_angle이라는 변수로 두어 초기의 36값에서 방향키 입력에 따라 1씩 증감하도록 하였다. (증감하는 1의 값은 10°에 해당)

또한, 여기서 오류가 발생하지 않도록 0보다 작거나 36보다 커지지 않도록 조건문을 추가하였다.

4. Two-sided constant shading:



위 사진은 Torus의 안쪽면을 빨강, 바깥면을 파랑으로 서로 다른 색상을 갖도록 shading한 모습이다.
이러한 안과 밖을 구분 짓기 위해서는 각 polygon의 normal vector **n**와 polygon의 중심에서 COP로 향하는 vector **v**을 사용한다.
여기서 **n**과 **v** 사이의 이루는 각이 90°이하일 경우에는 바깥면을 보고 있는 것이고, 90°보다 클 때는 안쪽면을 보고 있는 것이다.
또 이렇게 깔끔하게 구분되는 이유는 해당 각을 구하는 연산을 각 polygon에서 진행하기 때문에 perspective projection이 되기 때문이다.



위 왼쪽 사진은 각 polygon에서의 normal vector를 시각화한 것이다. 여기서 COP는 현재 사진에서 카메라의 위치이므로 normal vector가 시작되는 각 polygon의 중심에서 viewer의 눈을 잇는 가상의 직선이 있다고 생각하고, 가상의 직선과 각 polygon의 normal vector의 각도가 90°가 넘는 것과 넘지 않는 것을 보면 안과 밖을 구분짓기 편할 것이다.
오른쪽 사진은 polygon의 중심에서 시작되는 normal vector와 COP로 향하는 vector를 시각화한 것이다. 이를 보면 파란면의 경우에는 두 vector가 이루는 각이 90° 이하이고, 빨간면의 경우에는 90°가 넘는 것을 확인할 수 있다.

Extra(번외):

Torus를 회전시켜도 4번과 같이 두 면에 대한 다른 색상으로 shading하기 위해서는 Torus가 회전됨에 따라 COP도 회전되야한다.
여기서 말하는 COP는 카메라의 위치가 아닌 vector **v**를 정의하기 위한 각 polygon의 중심에서 "COP"로 향하는 vector를 말할 때의 vector **v**를 계산하기 위한 COP이다.
그리하여 Torus가 회전됨에 따라 vector **v**의 종점도 회전되면 4번과 같이 두 가지 색상으로 shading이 가능하다.
여기서 주의할 점은 vector **v**의 종점은 Torus가 회전하는 각과 반대로 회전해야 한다는 것이다.