

Computer Graphics

Prof. Jibum Kim

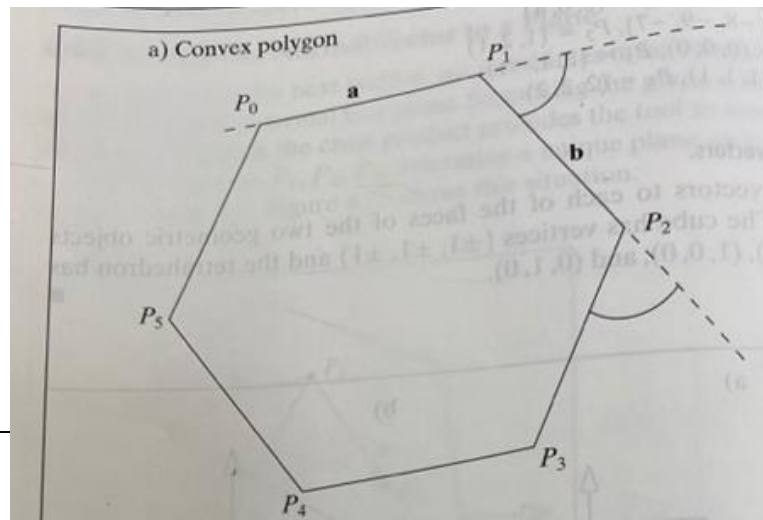
Department of Computer Science & Engineering

Incheon National University

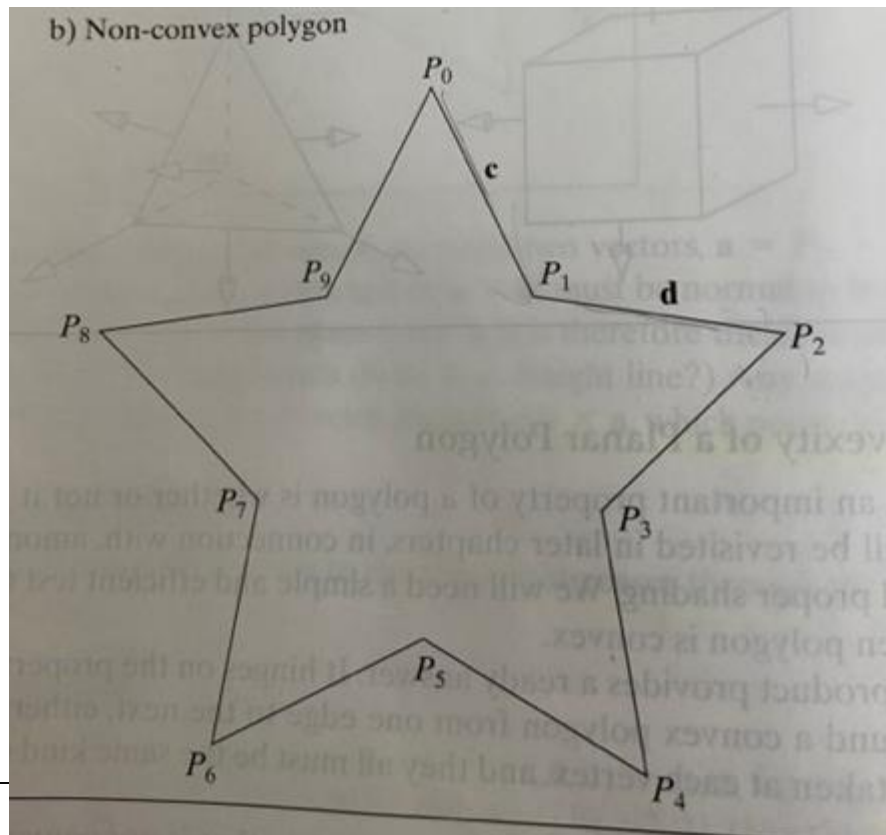
- **Test the convexity of a planar polygon**

-
- An important property of a polygon is whether or not it is convex
 - Cross product를 이용하면 간단한 test가 가능
 - As one traverses around a convex polygon from one edge to the next, either a left turn or a right turn is taken at each vertex, and **they all must be the same kind of turn (all left or all right)**

- 예: Consider a **convex polygon with vertex: P_0, P_1, \dots, P_5**
- Edge from P_0 to P_1 is vector \mathbf{a} . Vector \mathbf{b} points from P_1 to P_2 . There are six **edge vectors**
- There is a right turn from each edge of the polygon to the next edge
- For each right turn, **$\mathbf{a} \times \mathbf{b}$ points into the plane of the polygon**
- For each left turn, **$\mathbf{a} \times \mathbf{b}$ points out of the plane**



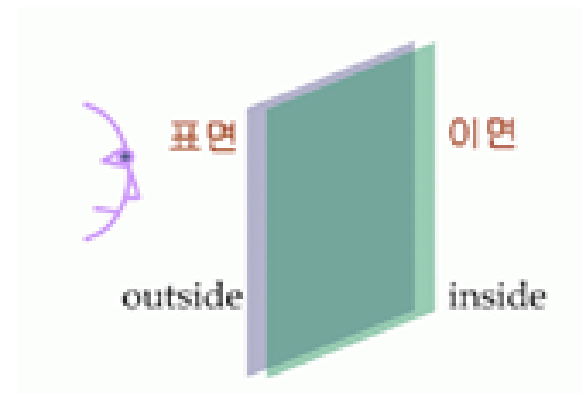
- **Non-convex polygon**
- Not all of the turns are the same: five of them are left turns and five of them are right turns



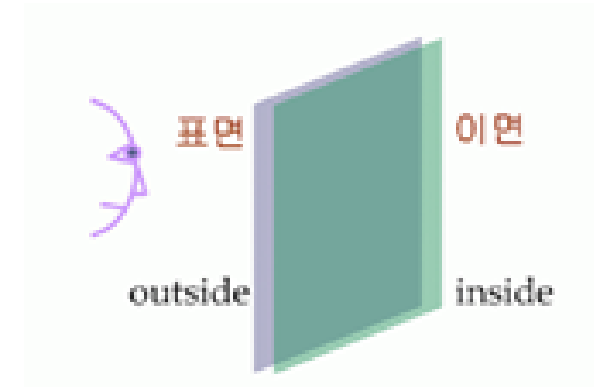
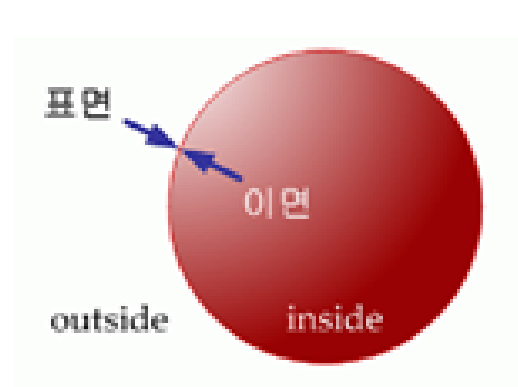
-
- Observation
 - A polygon is convex if the cross products between each edge vector and the **next all point** into the plane or out of the plane

■ Backface culling (후면 제거)

- 정의
- Given a polygon (e.g., 2D triangle, 2D quad)
- Front face (앞면): viewer (카메라)에게 보이는 면
- Back face (후면): viewer에게 보이지 않는 면



- 아래 왼쪽과 같이 구와 같은, **닫힌 다각형 (closed polygon)**의 경우 안쪽 면은 viewer의 위치와 상관 없이 (viewer가 구 내부에 있지 않은 한) 안쪽 면은 항상 viewer가 볼 수 없는 back face가 된다. 이런 경우 front face와 back face가 명확하다
- 하지만, 아래 오른쪽과 같은 **열린 다각형 (open polygon)**은 viewer의 위치에 따라서 front face와 back face가 결정 된다



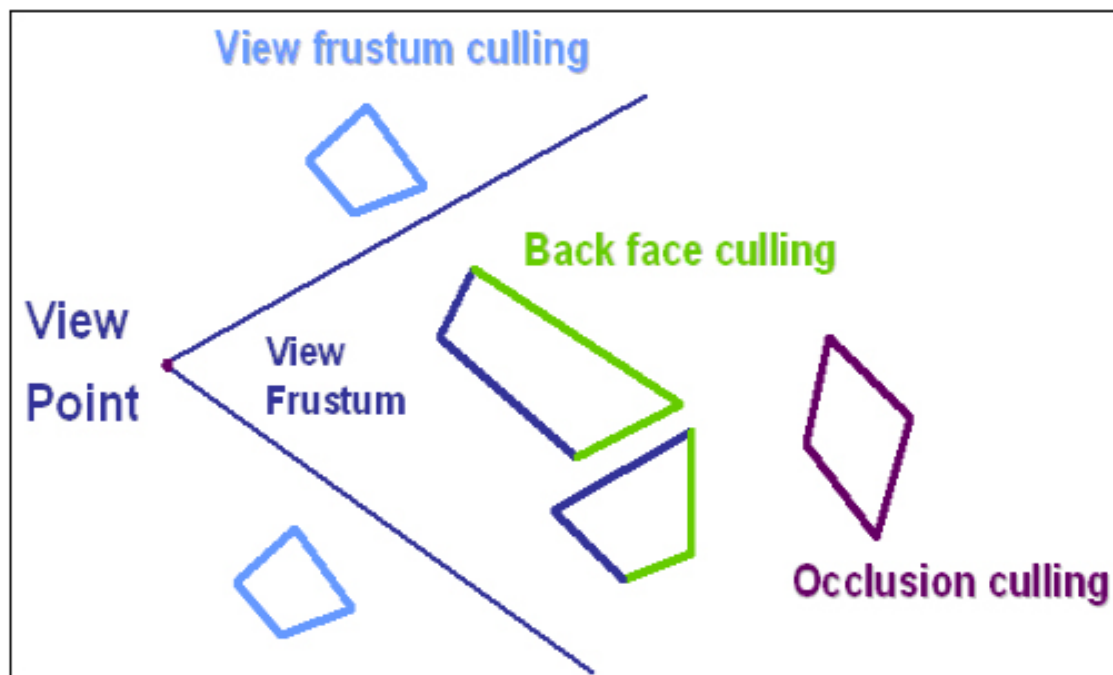
- Back face는 어차피 viewer에게 보이지 않는 면이므로 제거하는 것이 효과적일 수 있다. 이와 같이 후면을 제거하는 것을, 이를 **back face culling (후면 제거)**이라고 한다. 혹은, back face removal 이라고도 한다
- 수만/수십만 개의 polygon으로 이루어진 복잡한 object의 경우 후면 제거를 사용하면 큰 시간 절약 가능

■ 후면제거, 은면 제거의 차이점

-
- **Backface culling (후면 제거):** viewer에게 보이지 않는 뒷면에 위치하는 back face를 제거
 - **Hidden surface removal (은면 제거):** 다른 물체에 가려서 보이지 않는 hidden surface 은면 제거
OpenGL에서는 Depth buffer (Z-buffer) 사용하여 은면을 제거함

View frustum culling: clipping

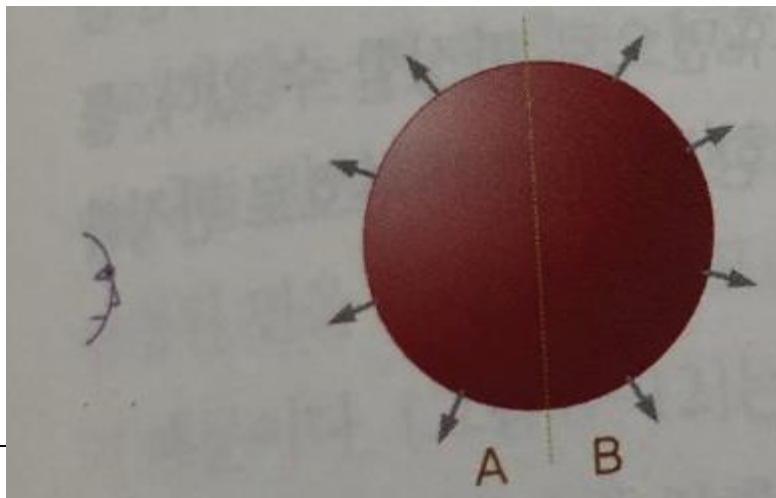
Occlusion culling: hidden surface removal (은면 제거)



-
- Unity에서의 frustum culling과 occlusion culling 설명 (field of view, FOV 설명도)
 - Unity - Manual: Understanding the View Frustum (unity3d.com)
 - Unity - Manual: Occlusion culling (unity3d.com)

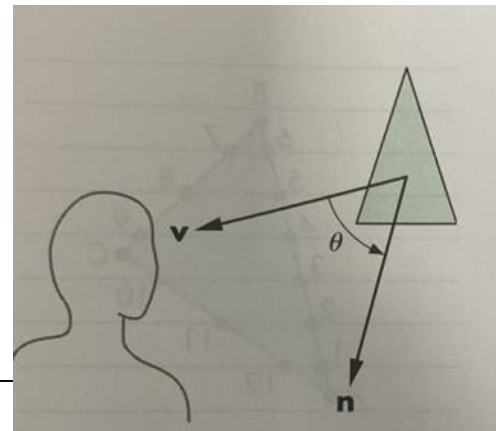
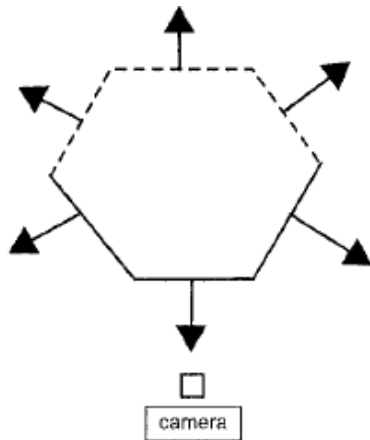
■ Normal vector와 backface culling

- 구의 화살표는 평면 다각형의 법선 벡터를 의미한다
- Viewer (camera)가 보았을 때 B부분의 polygon들은 보이지 않는다. 그 이유는 이 면들의 법선 벡터가 시점 반대쪽 (viewer로 부터 멀어지는 방향)을 바라보기 때문이다. 즉, B부분의 면들은 backface이다
- Viewer가 보았을 때 A부분의 polygon들은 보인다. 그 이유는 이 면들의 법선 벡터가 viewer 쪽을 향하기 때문이다. 즉, A 부분의 면들은 frontface이다

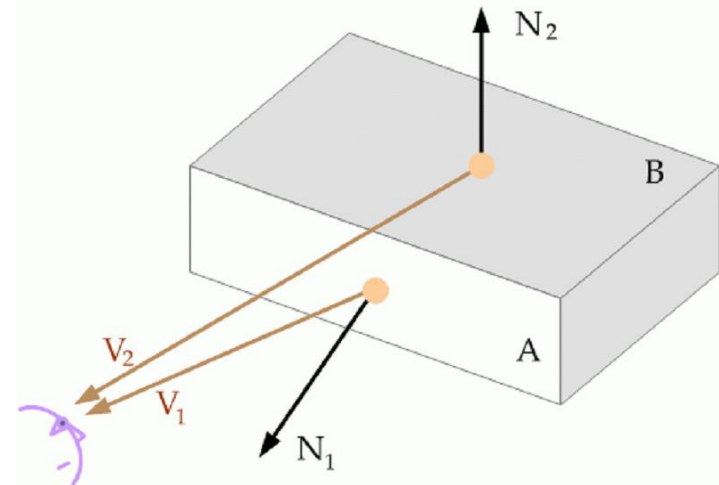
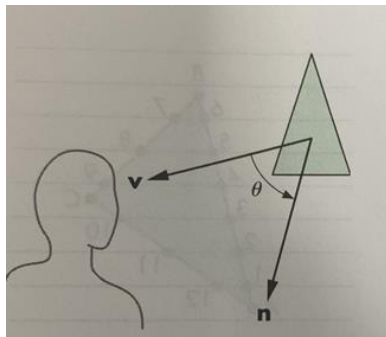


■ Observation

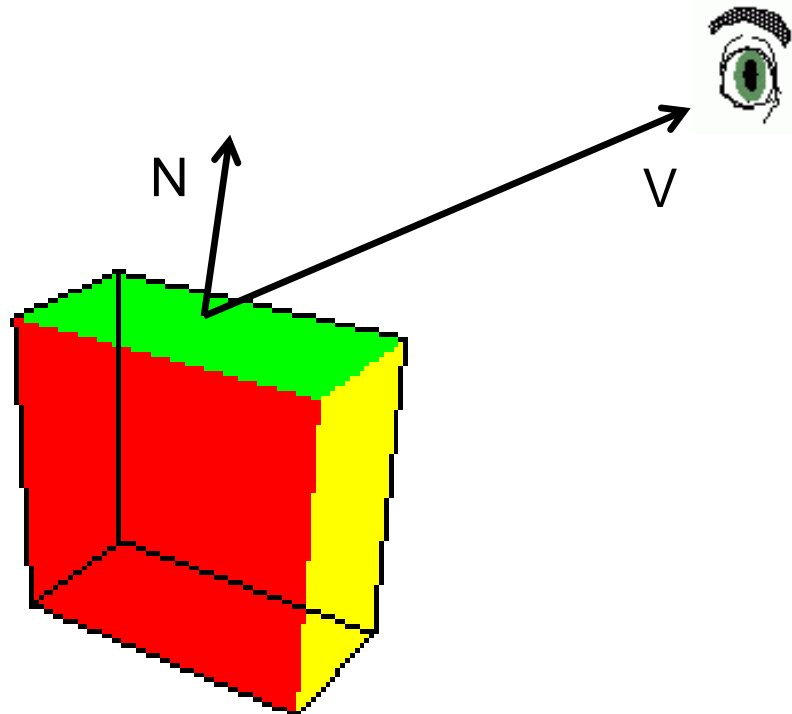
- 1. viewer로 부터 멀어지는 방향의 normal vector를 갖는 polygon의 면은 **back face** 이다
- 2. viewer쪽을 향하는 방향의 normal vector를 갖는 polygon의 면은 **front face**이다
- If θ is the angle between the normal vector (n) and the view vector (v), then the polygon is facing forward if and only if
- $-90 \leq \theta \leq 90$ or $\cos(\theta) \geq 0$



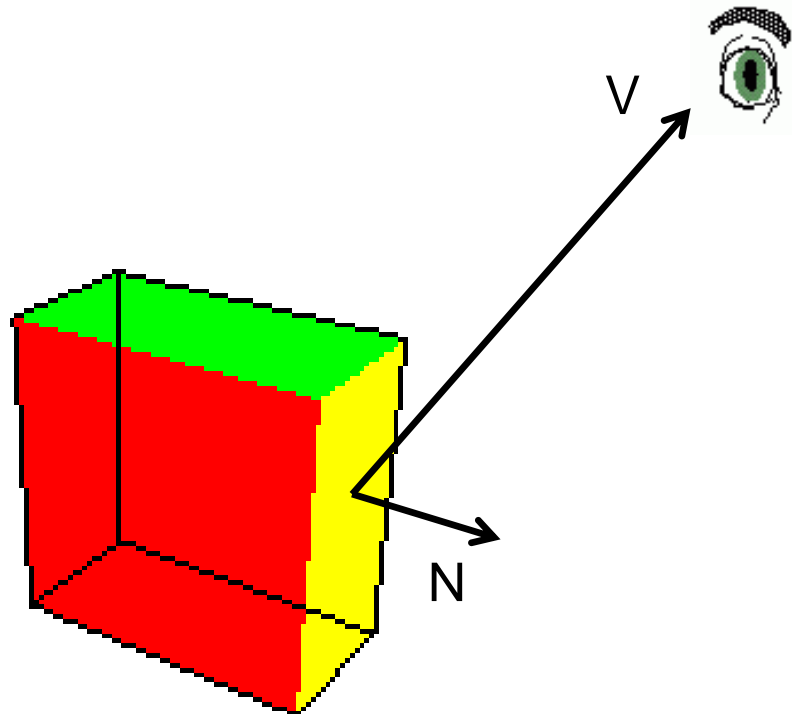
- 정의: view vector (V): polygon 면으로부터 viewer를 향한 벡터
- 정리: polygon 면의 법선 벡터 (N)과 view vector가 이루는 각이 90도가 넘으면 back face 이다
- 앞에서의 벡터 정리를 이용하면
- 즉, $V \cdot N < 0$ 인 polygon 면은 viewer로부터 back face 이다
- Face A: $V_1 \cdot N_1 \geq 0$, front face
- Face B: $V_2 \cdot N_2 < 0$, back face



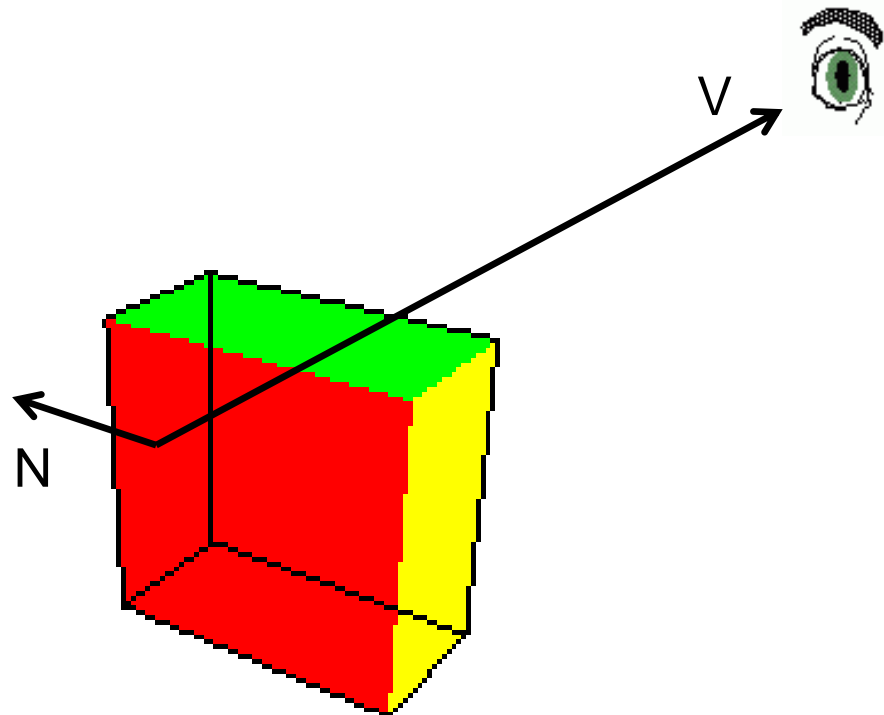
- Face (green) : front face
- Why?



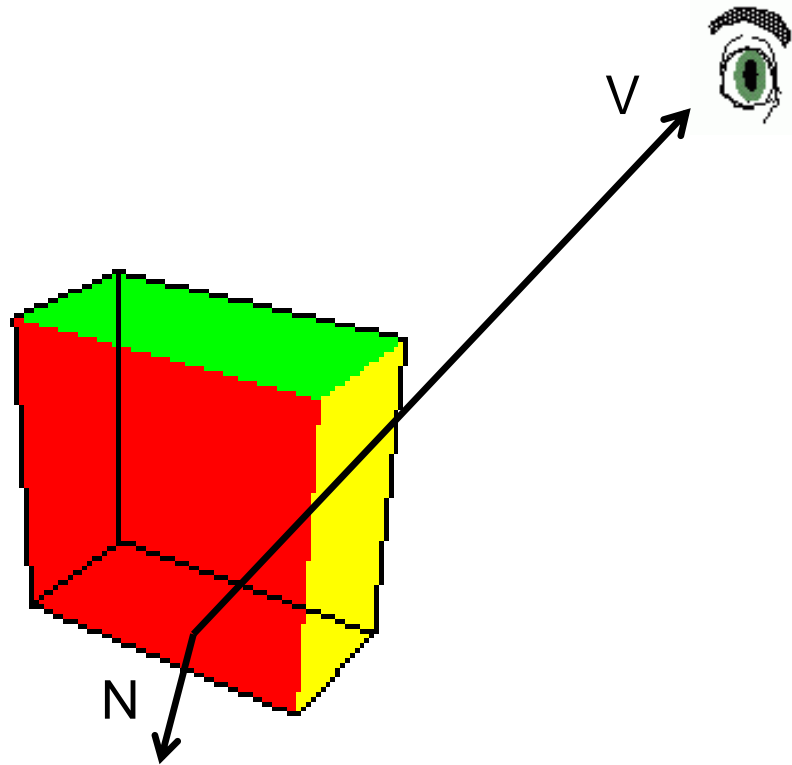
- Face (Yellow) : front face
- Why?



- Left face : back face
- Why?



- Bottom face : back face
- Why?



- **OpenGL에서의 backface culling**

- OpenGL에서는 back-face인 polygon (혹은 triangle)을 culling하는 방법이 있다
- To enable face culling, we only have to enable OpenGL'S "GL_CULL_FACE" option

```
glEnable(GL_CULL_FACE);
```

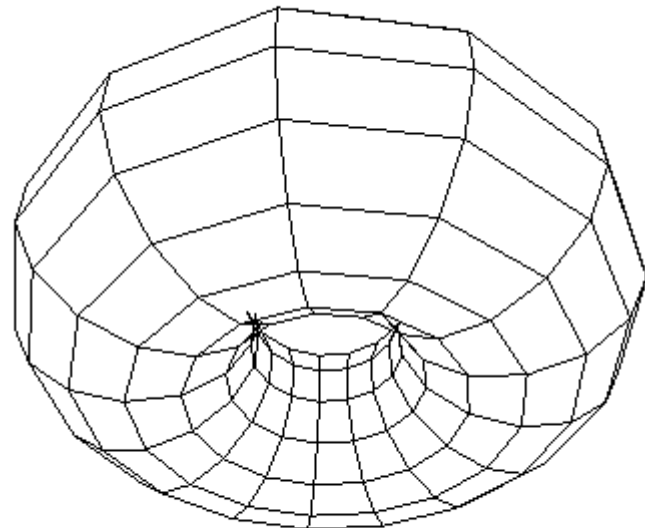
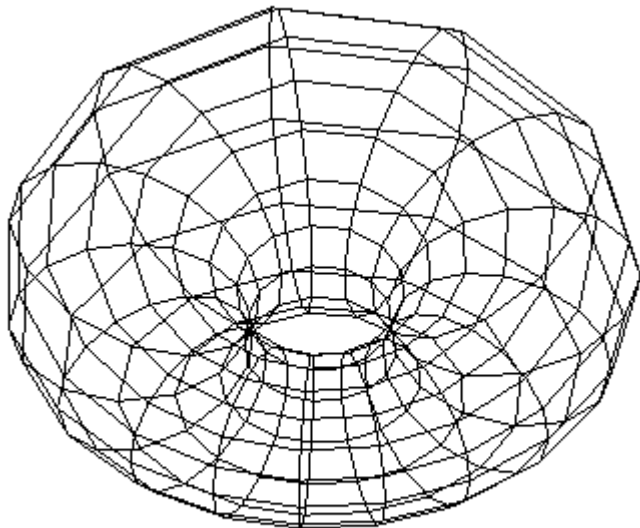
- Default로는 front-face가 아니면 제거

The `glCullFace` function has three possible options:

- `GL_BACK`: Culls only the back faces.
- `GL_FRONT`: Culls only the front faces.
- `GL_FRONT_AND_BACK`: Culls both the front and back faces.

-
- 더 자세한 내용은
 - LearnOpenGL - Face culling

- `// glEnable(GL_CULL_FACE);`
- `// glCullFace(GL_BACK);`
- `// back face culling`
- 주석 제거 전후를 비교해 보자



-
- https://www.dropbox.com/s/eb3u1edeamsv805/backface_2.txt?dl=0

■ OpenGL에서의 backface culling 예 2

- OpenGL에서 viewing frustum을 사용시 어떠한 polygon이 다음과 같은 vertex 순서로 정의되었다. Synthetic camera의 위치는 세계좌표로= (0, 7, 10)
- Q) 이 polygon 면은 viewer에게 front-face일까? Back-face일까?

```
glVertex3f(-5.0, 1.0, 5.0); //P1  
glVertex3f(-5.0, -1.0, 5.0); // P2  
glVertex3f(5.0, 1.0, 5.0); // P3
```

N: normal vector, V: view vector
 $N \cdot V > 0 \Rightarrow$ front face
 $N \cdot V < 0 \Rightarrow$ back face

1. 이 polygon (P1, P2, P3)의 normal vector, N 을 계산해 보자

2. View vector (polygon의 점으로부터 viewer로의 벡터) 계산

3. 이 polygon 면은 viewer에게 front-face? Back-face?

-
- OpenGL에서는 front face와 back face를 쉽게 구별하기 위하여 다음과 같은 mode를 제공한다
 - `glPolygonMode(GL_FRONT, GL_FILL);`
 - `// front face: filled`
 - `glPolygonMode(GL_BACK, GL_LINE);`
 - `// back face: outlined`
 - OpenGL에서 viewer (카메라)의 위치는 어떻게 조정할 수 있을까?

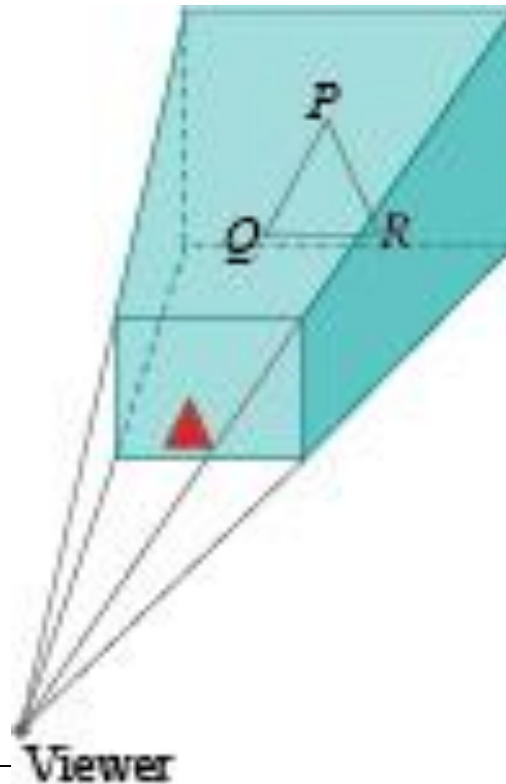
-
- https://www.dropbox.com/s/suns1g5z8r4vutl/backface_1.txt?dl=0

- Polygon을 명시할 때 vertex의 순서는 그 polygon 면의 normal vector의 방향을 결정 짓고 카메라가 위치한 view vector와의 연산을 통해 back-face인지 front-face인지 결정된다. 이를 **orientation**이라 한다
- 앞에서 세 vertex의 orientation을 바꿔 보자. 어떻게 보이는가?

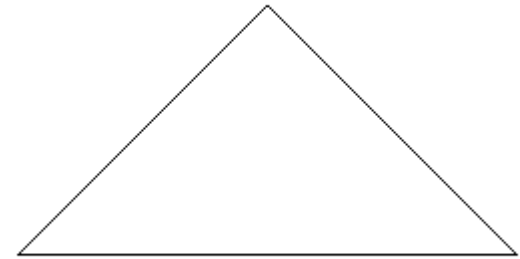
```
glBegin(GL_POLYGON);  
glVertex3f(-5.0, 1.0, 5.0); //P1  
glVertex3f(5.0, 1.0, 5.0); //  
glVertex3f(-5.0, -1.0, 5.0); // P2  
glEnd();
```


- Orientation: how are vertices ordered?
- Orientation은 viewer가 보았을 때 시계 방향 (Clockwise)이나 반 시계 방향 (counter-clockwise)로 정의될 수 있다
- 일반적으로, 반 시계 방향 (CCW) 의 orientation을 viewer는 앞면 (front face)라고 판단한다
- Viewer가 보았을 때 CW이면 후면 (backface)이라 판단한다

- 예: viewer (0,0,0)
- Q) Triangle의 vertex들의 orientation이 QPR일 때 이 triangle은 viewer로 부터 CCW로 보일까? CW로 보일까?



-
- https://www.dropbox.com/s/lnuwyc33d1wt7io/backface_3.txt?dl=0



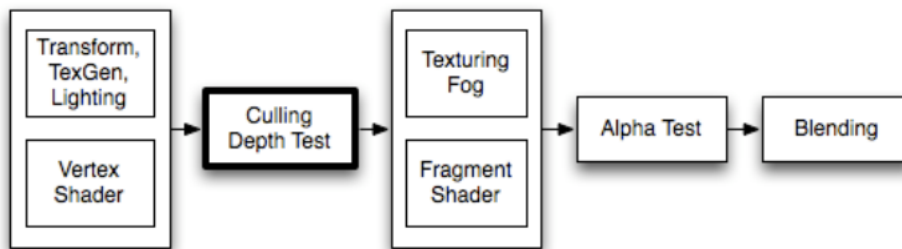
- 1. 다음의 코드는 viewer가 보았을 때 orientation이 CW인가? CCW인가? 그렇다면 front-face라고 판단하는가? Back-face라고 판단하는가?
- 2. Polygon 모드에서 front-face인 경우에는 fill이고 back-face이면 outline만 그린다. 어떻게 그려지는지 확인해 보자
- 3. 이 polygon의 normal vector를 계산해 보고 $V \cdot N$ 의 부호를 판단해 보자 1의 결과와 일치하는지 확인해보자
- $N =$
- $V =$
- $V \cdot N$

```
glBegin(GL_POLYGON);  
glVertex3f(-5.0, 0.0, -10.0); // Q  
glVertex3f(0.0, 5.0, -10.0); // P  
glVertex3f(5.0, 0.0, -10.0); // R  
glEnd();
```

■ Unity에서의 culling & depth testing

ShaderLab: Culling & Depth Testing

[Other Versions](#)

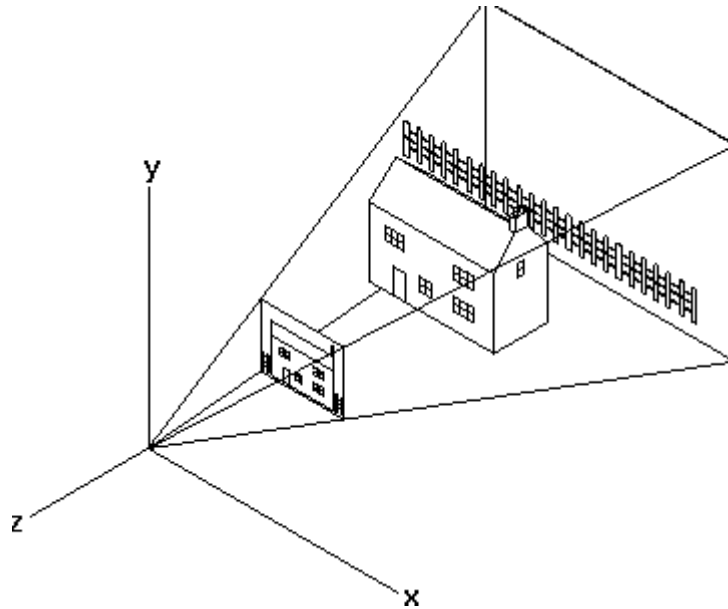


Culling is an optimization that does not render polygons facing away from the viewer. All polygons have a front and a back side. Culling makes use of the fact that most objects are closed; if you have a cube, you will never see the sides facing away from you (there is always a side facing you in front of it) so we don't need to draw the sides facing away. Hence the term: Backface culling.

The other feature that makes rendering looks correct is Depth testing. Depth testing makes sure that only the closest surfaces objects are drawn in a scene.

■ Hidden surface removal

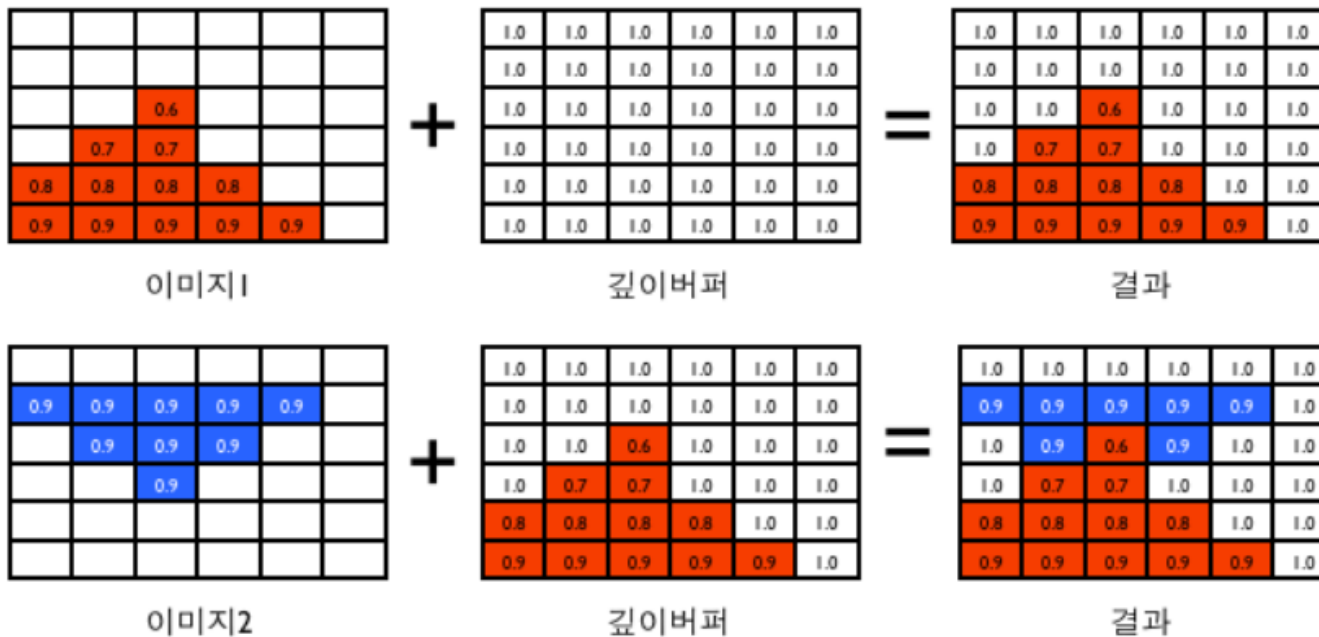
- 앞 물체에 가려서 Viewer에게 보이지 않은 물체 (면)를 제거하는 작업을 **hidden surface removal (은면 제거)**라고 한다



- OpenGL에서 frame buffer내의 color buffer를 사용하여 각 pixel의 색깔 정보를 2차원 배열로 저장한다. 추가적으로 **hidden surface removal (은면 제거)**을 위하여 depth buffer (깊이 버퍼)를 사용할 수 있다
- Depth buffer는 color buffer와 마찬가지로 memory이다
- OpenGL에서는 depth-buffer 를 사용하여 viewer가 볼 때 다른 물체에 가려서 뒤에 있는 물체 (즉, 은면)을 제거할 수 있다
- **Depth buffer를 z-buffer**라고도 한다. 기본적으로 z값은 viewer와 물체와의 거리를 의미한다

- Z-buffer 알고리즘은 **각 픽셀 별로 동작**한다 (Frame buffer의 하나)
그래서 memory를 많이 사용하는 단점이 있다
- Z-buffer 알고리즘은 **screen coordinate 단계**에서 동작한다
- Z-buffer 알고리즘
 - 1. The depth buffer is initialized to a value that corresponds to the **farthest distance from the viewer**
 - 2. When each polygon is rasterized, the depth of each pixel (how far the corresponding point on the polygon is from the viewer) is calculated
 - 3. If the depth is **less (closer) than what is already in the z-buffer**, we use the color of the polygon to replace the color of the pixel and update the depth in the z-buffer.

- 최초 depth buffer의 각 픽셀값 초기화 (여기서는 1.0)
- 여기서는 각 픽셀의 depth 값이 작을수록 camera와의 거리가 가깝다고 가정



■ OpenGL에서의 depth-buffer 사용

1. Display 콜백 함수 (Default로는 disable되어있음)

glEnable(GL_DEPTH_TEST);

2. Display 콜백 함수 (buffer 지움)

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// bitwise OR 연산 사용

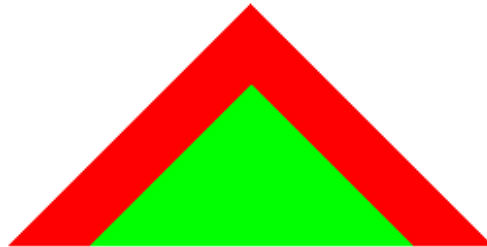
3. Main 함수 (예전 버전)

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)

// 디스플레이 모드 설정 , color buffer= single, RGB, DEPTH buffer 사용

// bitwise OR 연산 사용

- Frustum을 사용하고 camera의 위치 및 방향을 변경하지 않고 같은 크기의 두 개의 polygon에 대하여 z값을 다르게 주고 Viewer에게 z-buffer를 사용하지 않고 실행해 보았다. 나중에 그린 삼각형 (초록색)이 먼저 그린 삼각형 (빨간색) 위에 그려진다



-
- https://www.dropbox.com/s/yspp5m02jvon7bl/depth_1.txt?dl=0

-
- 초록색 삼각형이 빨간색 삼각형 보다 camera에서 더 멀리 있다
 - **hidden-surface removal (은면 제거)**을 적용해보자
 - Depth buffer 적용후의 실행 화면 왜 빨간색 삼각형만 보이는가?



-
- https://www.dropbox.com/s/nbxu3lche9r99s0/depth_2.txt?dl=0

- LearnOpenGL - Depth testing