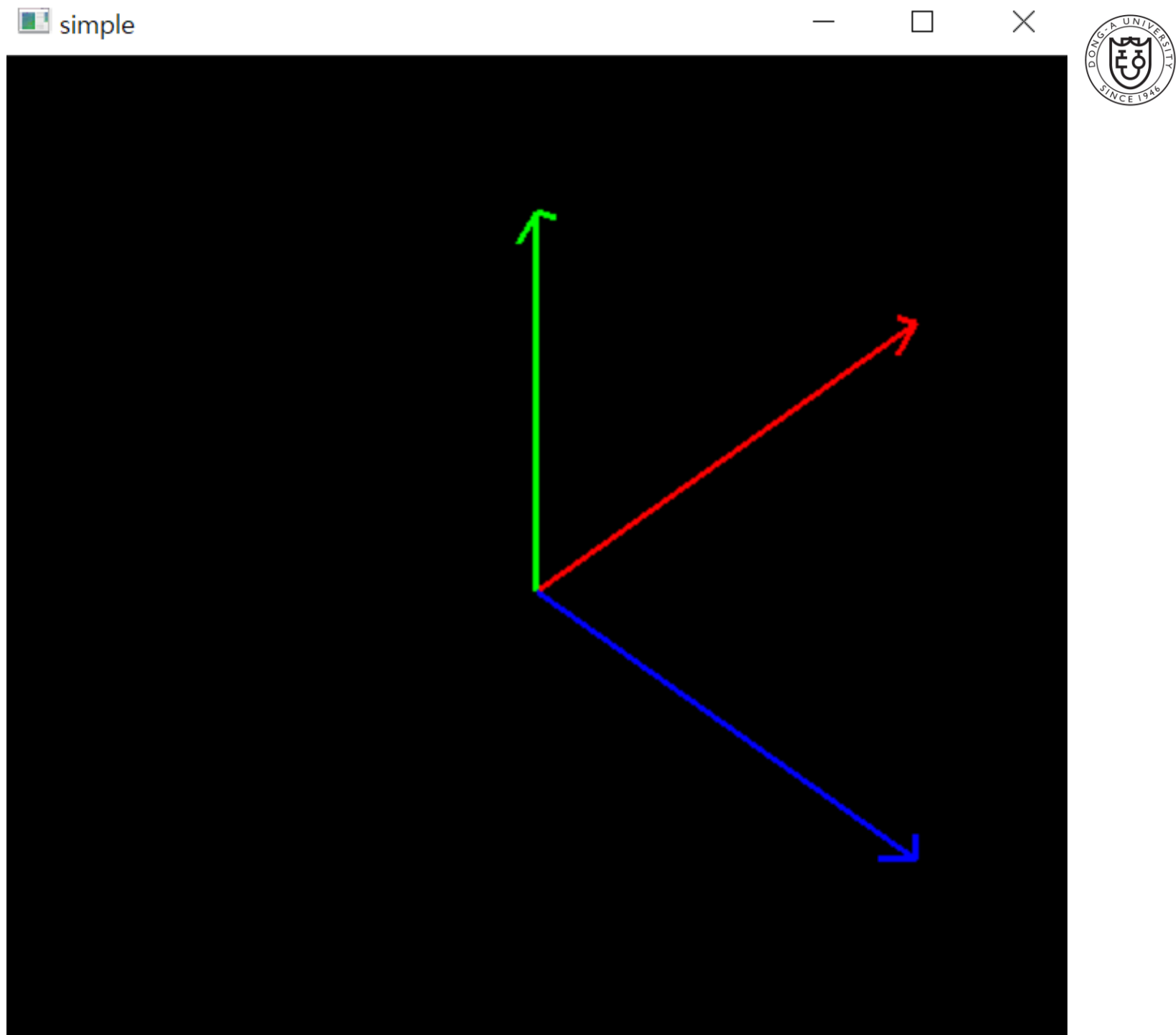


## 점선 그리기(3\_5\_2)

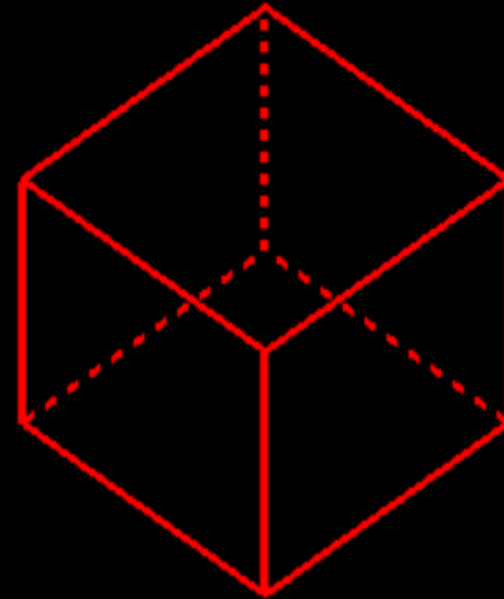
- 화살표 있는 3개의 축 그려보자!



# 점선 그리기(3\_5\_3)

- 안보이는 부분을 점선으로 그려보자!
- 정육면체

simple

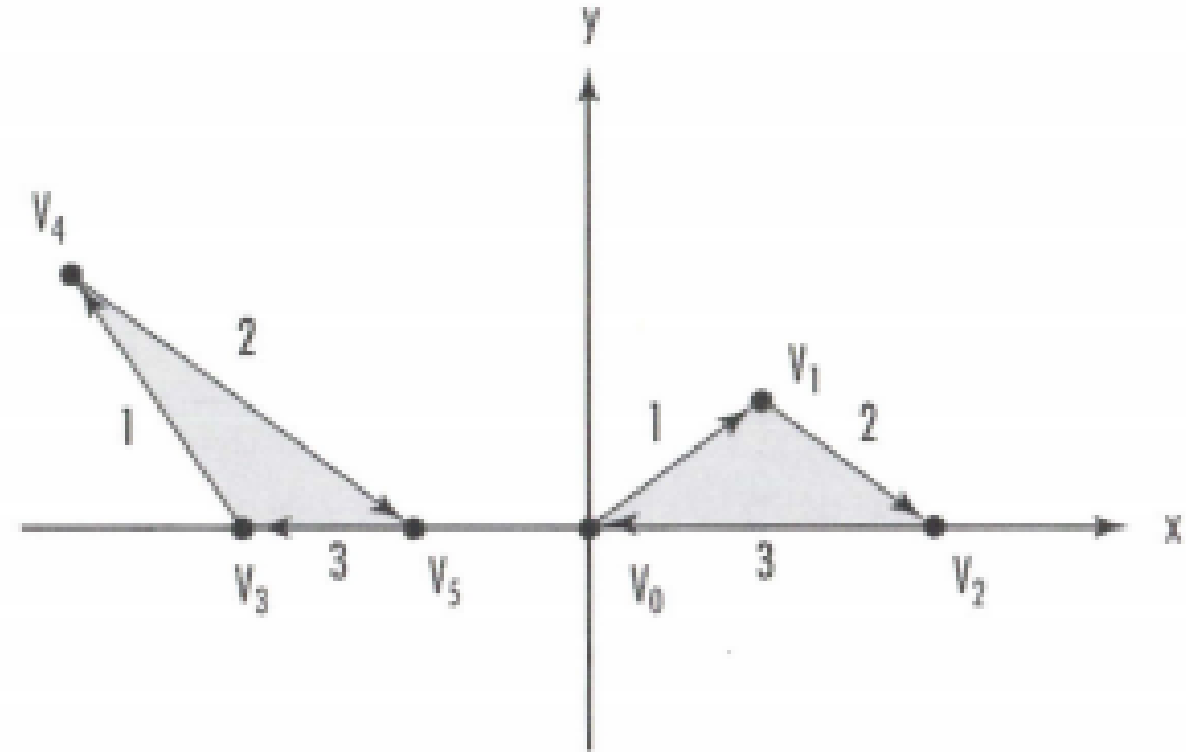


# 폴리곤 그리기

- **내부가 채워진 솔리드 면**을 만들기 위해서는 다각형(폴리곤)이 필요
- 삼각형: 가장 기본적인 폴리곤
  - GL\_TRIANGLES

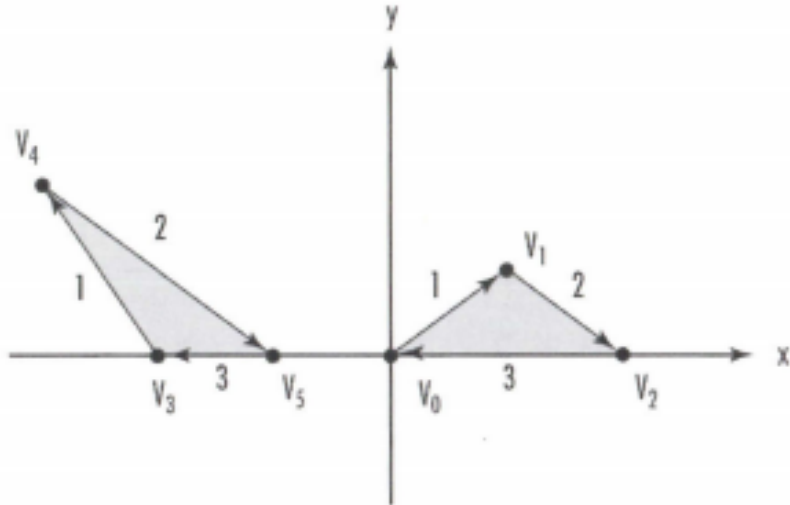
```
glBegin(GL_TRIANGLES);
    glVertex2f(0.0f, 0.0f);           // V0
    glVertex2f(25.0f, 25.0f);         // V1
    glVertex2f(50.0f, 0.0f);          // V2

    glVertex2f(-50.0f, 0.0f);          // V3
    glVertex2f(-75.0f, 50.0f);         // V4
    glVertex2f(-25.0f, 0.0f);          // V5
glEnd();
```

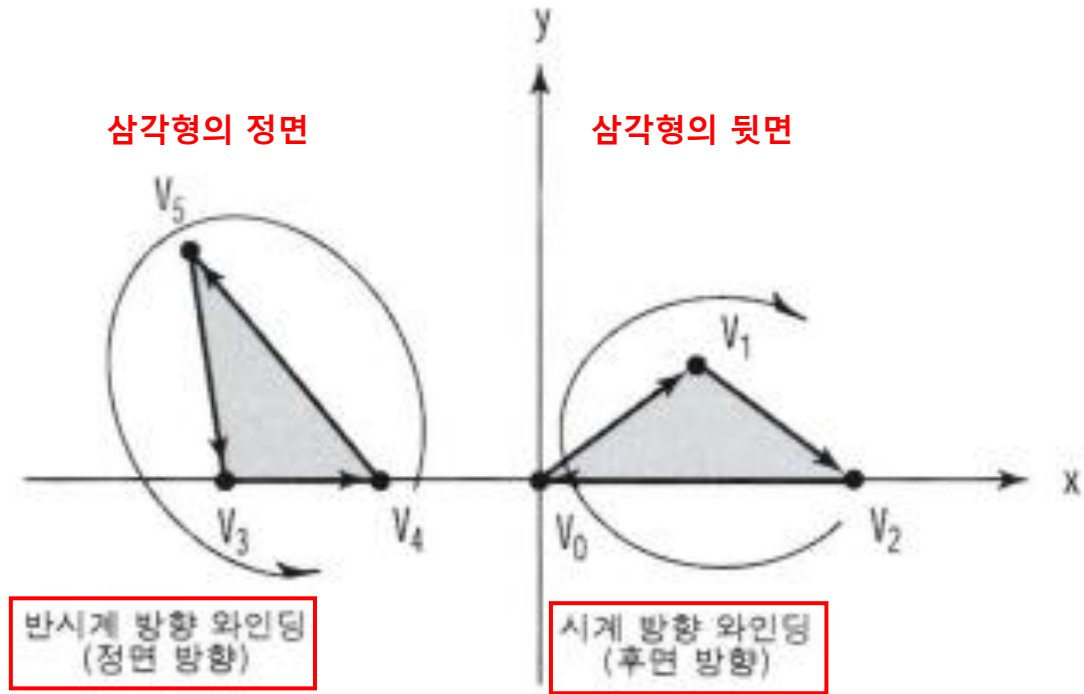


# 폴리곤 그리기

- 와인딩(winding) : 버텍스가 지정된 순서와 방향의 조합
  - OpenGL은 표준적으로 삼각형에 (정면에서 보았을 때) **반 시계 방향**의 와인딩이 적용된 것으로 간주함
- 왜 방향을 따질까?
  - 폴리곤의 앞면과 뒷면에 서로 다른 물리적 속성을 부여하게 되는 경우가 있기 때문
    - ✓ 앞면과 뒷면을 구분해 보이게 하거나 보이지 않게 할 수 있음
- OpenGL의 이러한 특성(반 시계방향이 기준) 변경 가능
  - glFrontFace (GL\_CW)
    - ✓ GL\_CW : 시계 방향의 와인딩을 폴리곤에 적용
    - ✓ GL\_CCW : 반 시계 방향의 와인딩을 폴리곤에 적용



V4와 V5의 위치를 바꾸면

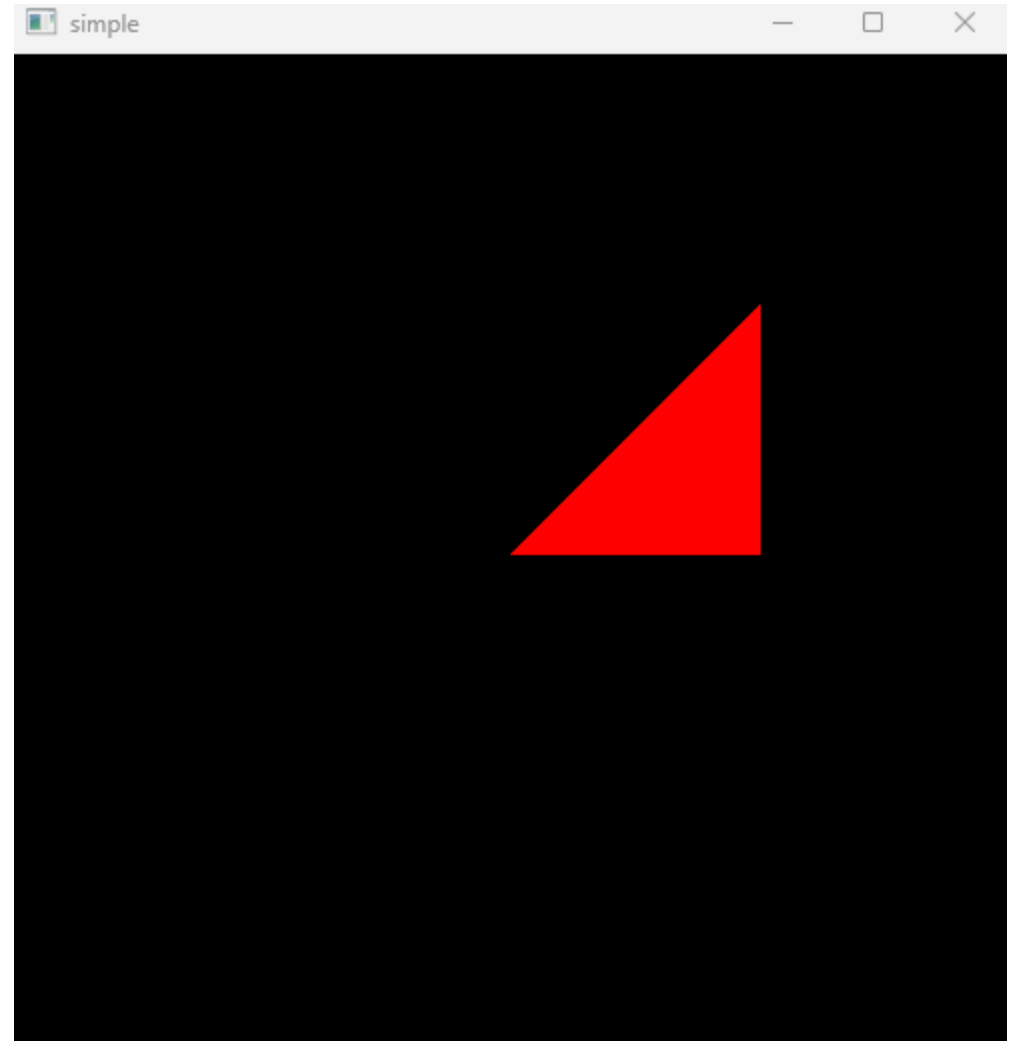


# 삼각형 그리기

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_TRIANGLES);
        glVertex2f(0.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();

    glFlush();
}
```



# 폴리곤 그리기 색상

- `glShadeModel(GL_FLAT);`
- `glShadeModel(GL_SMOOTH);`
- 아래 코드를 추가해서 2개의 모드의 차이를 확인하시오

```
// 폴리곤을 채우도록, (반대로, default = glShadeModel(GL_SMOOTH); )  
glShadeModel(GL_FLAT);
```

# 폴리곤 그리기 색상

```
// 폴리곤을 채우도록, (반대로, default = glShadeModel(GL_SMOOTH); )
glShadeModel(GL_FLAT);
```

- 다음 코드의 결과 차이는?

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        //glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        //glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_FLAT);
}
```

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        //glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        //glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

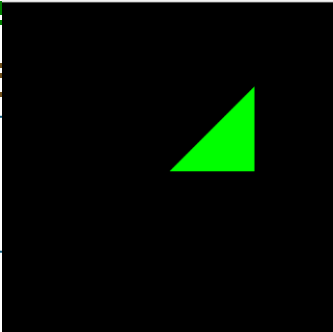
void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_FLAT);
}
```

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        glColor3f(1.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_FLAT);
}
```

# 폴리곤 그리기 색상

// 폴리곤의 마지막 버텍스가 지정되었을 때의 현재 색상으로  
 // 폴리곤을 렌더링하도록, (반대로) `glShadeModel(GL_FLAT)`  
`glShadeModel(GL_FLAT);`

- 다음  이하는?

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        //glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        //glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_FLAT);
}
```

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        //glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        //glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_FLAT);
}
```

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        glColor3f(1.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_FLAT);
}
```



# 폴리곤 그리기 색상

- GL\_SMOOTH 의 결과는?

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(50.0f, 50.0f);
    glEnd();
    glFlush();
}

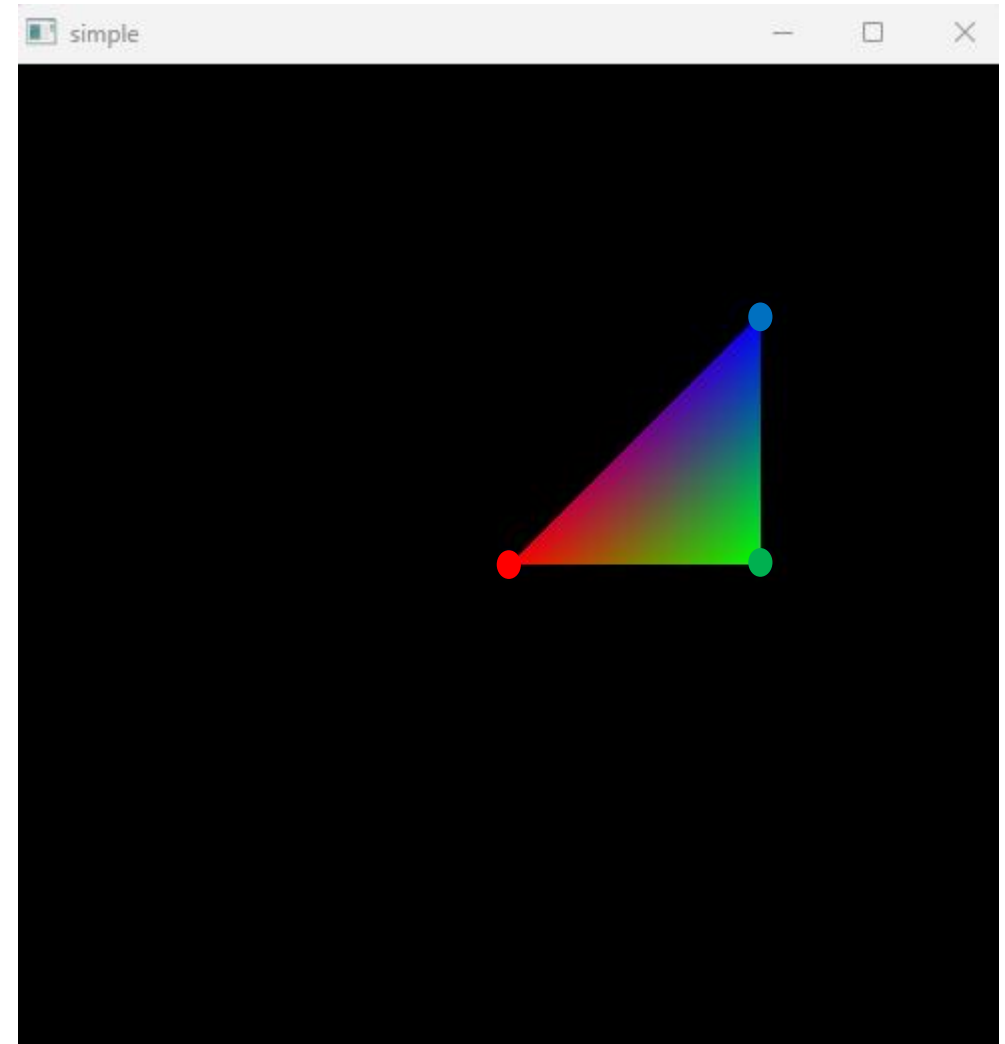
void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_SMOOTH);
}
```

# 폴리곤 그리기 색상

## ■ GL\_SMOOTH 의 결과는?

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f); ●
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(50.0f, 0.0f); ●
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(50.0f, 50.0f); ●
    glEnd();
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glShadeModel(GL_SMOOTH);
}
```



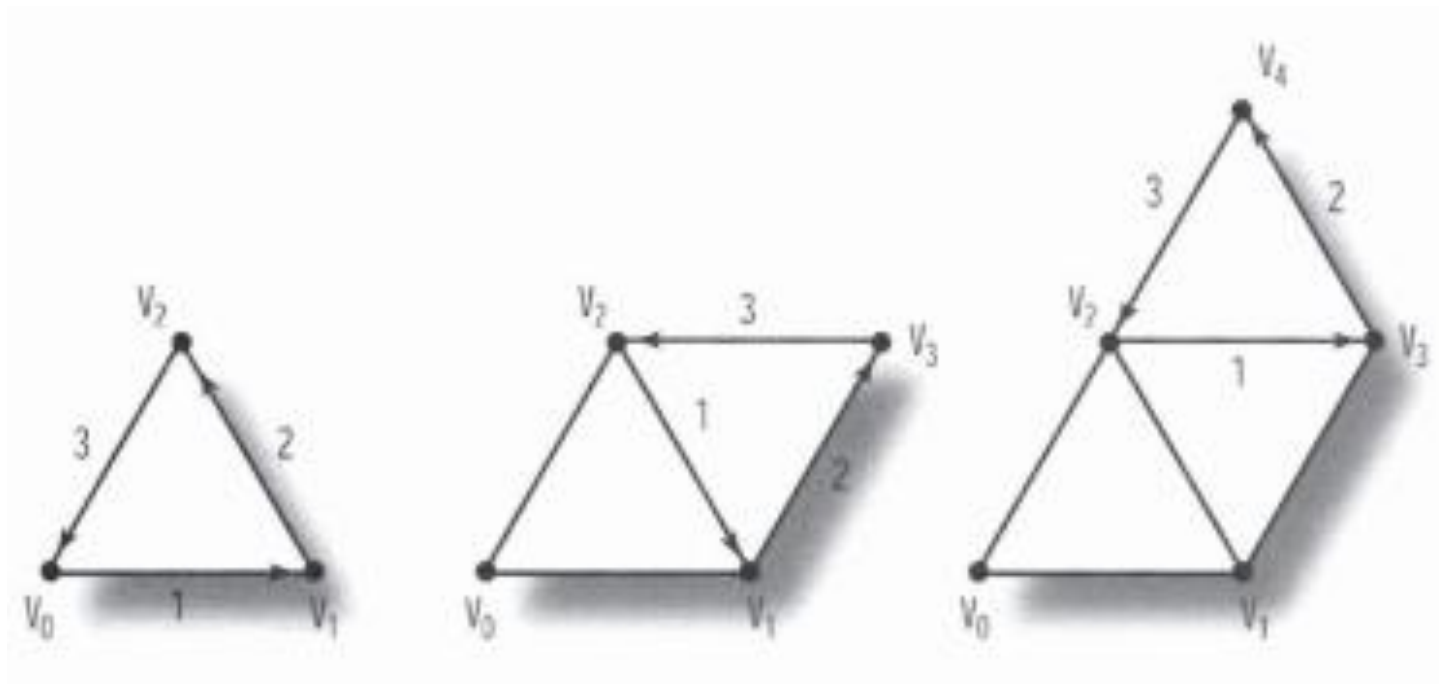
# 폴리곤 그리기

## ■ 삼각형 스트립

### • 여러 개의 삼각형 연결

#### ✓ GL\_TRIANGLE\_STRIP

- 첫 번째 삼각형은 첫 번째( $V_0$ ), 두 번째( $V_1$ ), 세 번째( $V_2$ ) 정점으로 구성
  - 정점이 전달된 순서대로 그려 짐
- 이후 새로운 정점( $V_n$ )이 추가될 때마다 **마지막 두 개의 정점과 함께 새로운 삼각형을 형성**



# 폴리곤 그리기(3\_6)

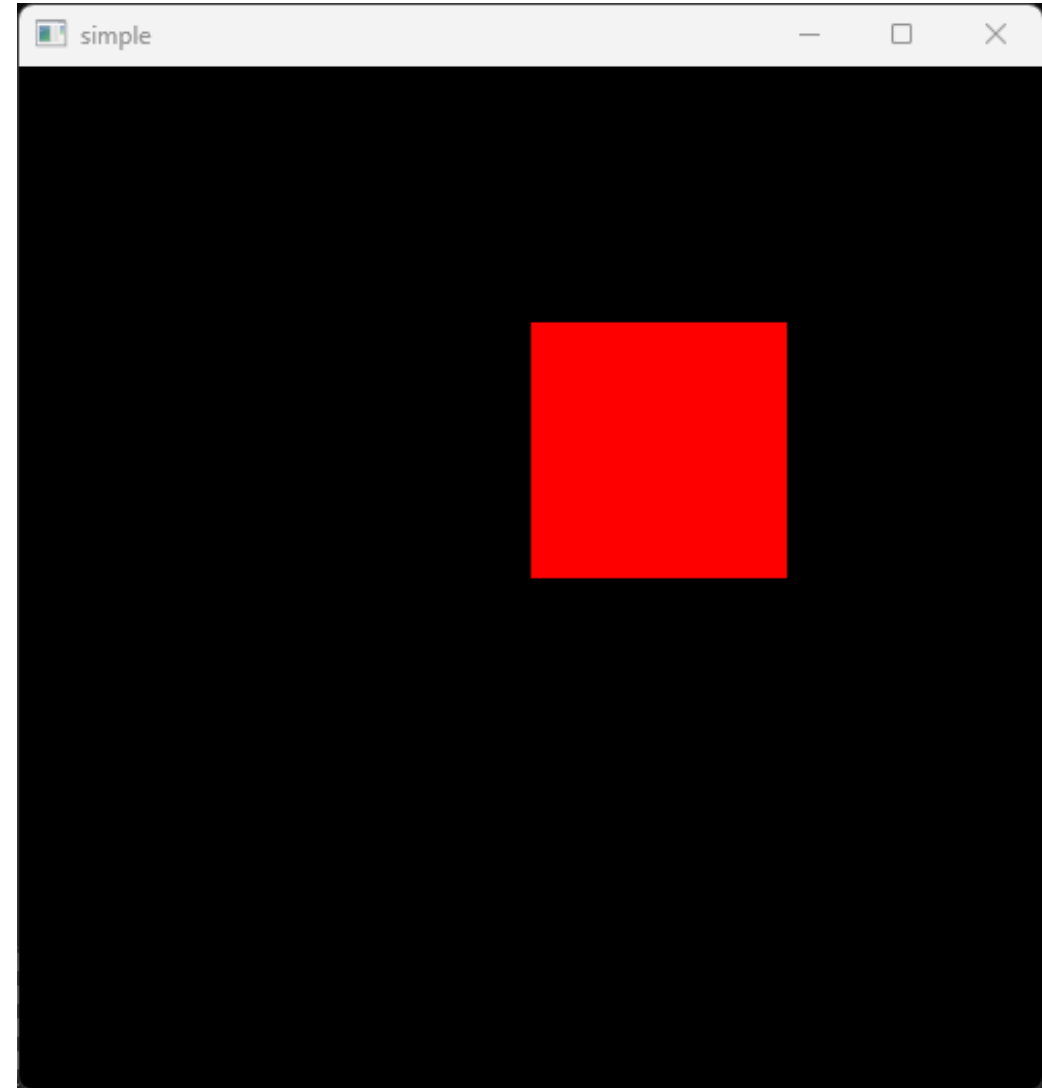
Try



## ■ GL\_TRIANGLE\_STRIP 예

- glBegin(GL\_TRIANGLE\_STRIP);  
    glVertex2f();  
    glVertex2f();  
    glVertex2f();  
    glVertex2f();  
    glEnd();

의 **점 4개만** 이용해서 정사각형 그리기



# 폴리곤 그리기(3\_6\_1)

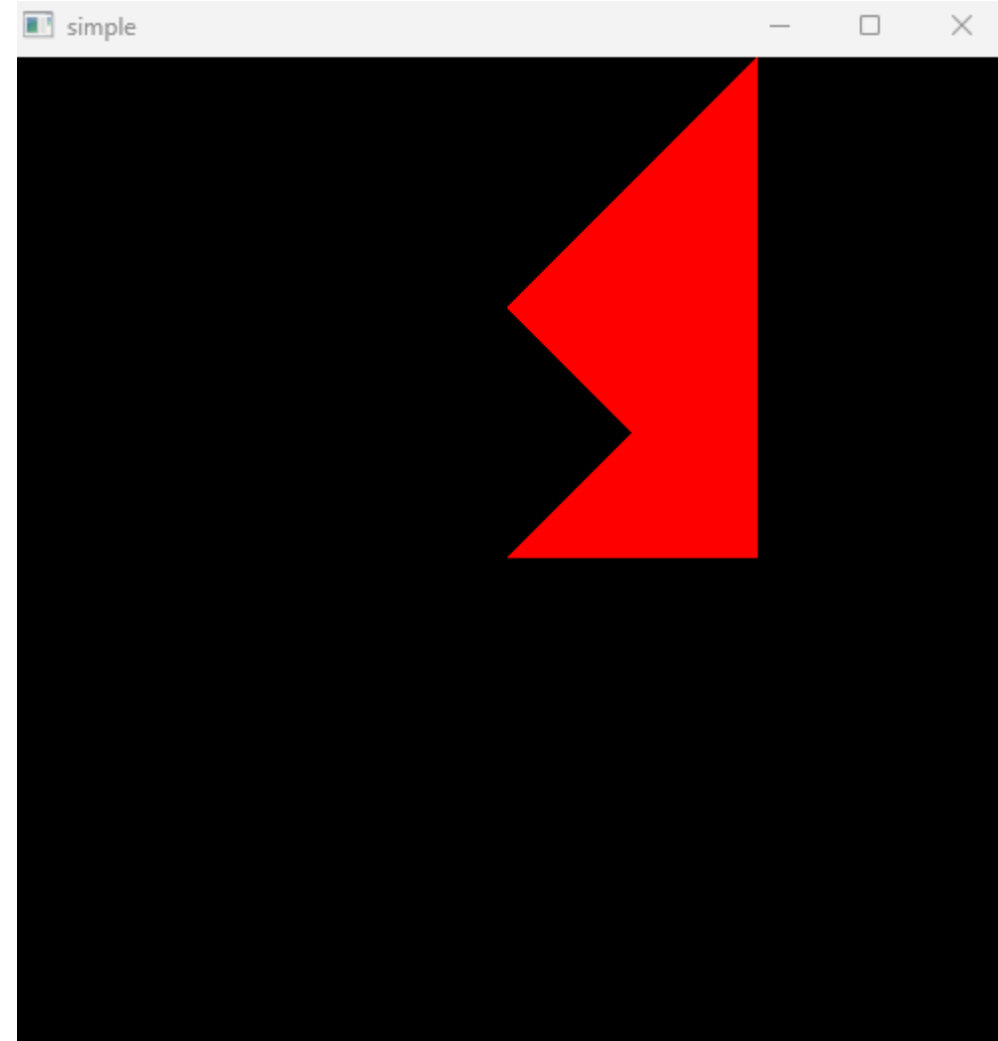
Try



## ■ GL\_TRIANGLE\_STRIP 예

- glBegin(GL\_TRIANGLE\_STRIP);  
glVertex2f();  
glVertex2f();  
glVertex2f();  
glVertex2f();  
glVertex2f();  
glEnd();

의 점 5개만 이용해서 그리기

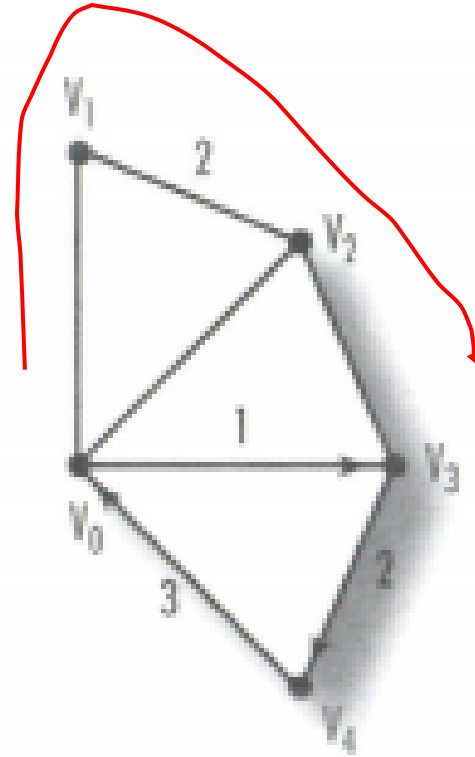
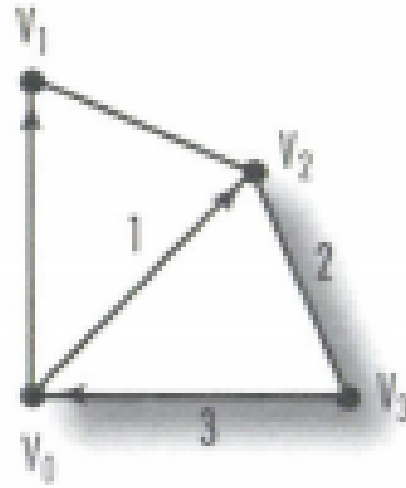
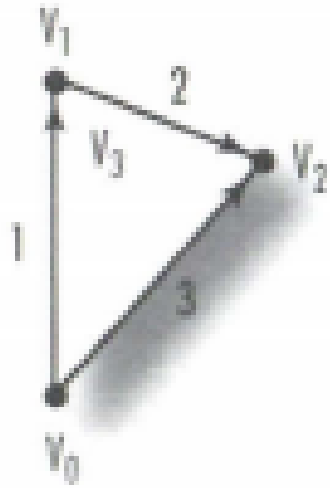


# 폴리곤 그리기(3\_7)

## ■ 삼각형 팬

- 고정된 버텍스를 중심으로 부채꼴 모양(fan)의 연결된 삼각형들을 만드는 함수

`glBegin(GL_TRIANGLE_FAN)`



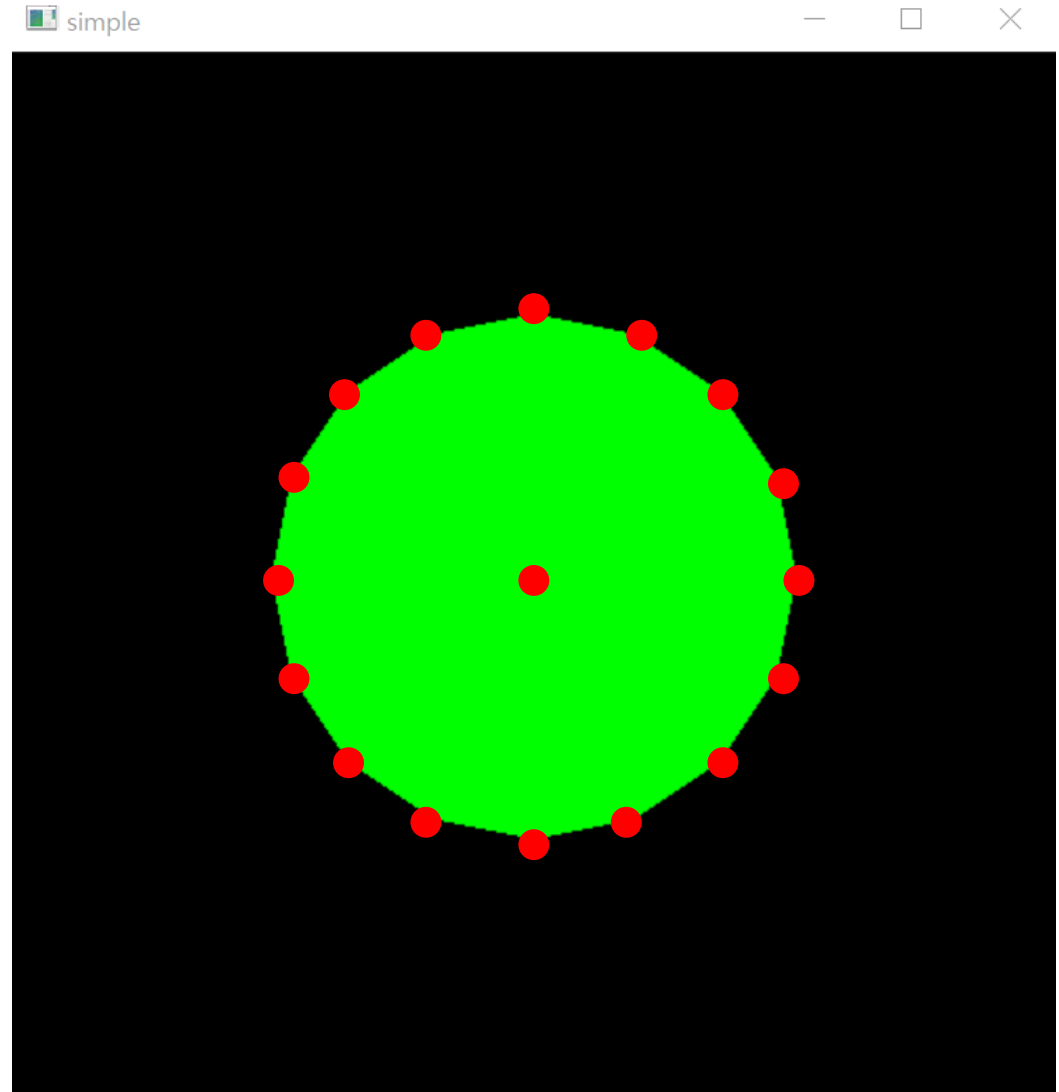
# 폴리곤 그리기(3\_7)

Try



- “GL\_TRIANGLE\_FAN”을 이용하여 다음을 그려보시오
  - 빨간점은 이해를 위한 것으로 그리는 것은 아님!
  - 빨간점의 개수 = 17개(원점 포함) = 16조각

```
glBegin(GL_TRIANGLE_STRIP);  
    glVertex3f();  
    ...  
glEnd();
```

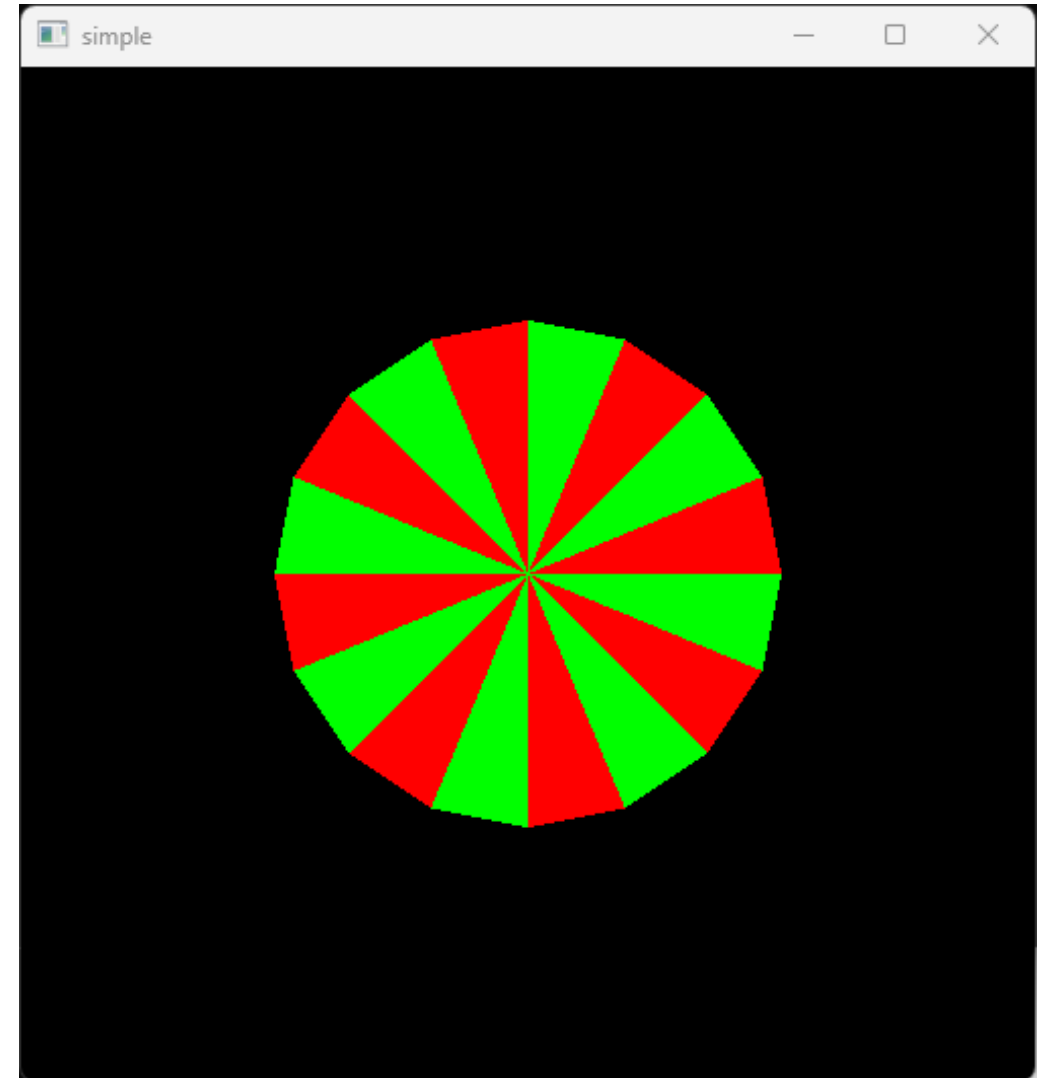


# 폴리곤 그리기(3\_7\_a)

Try



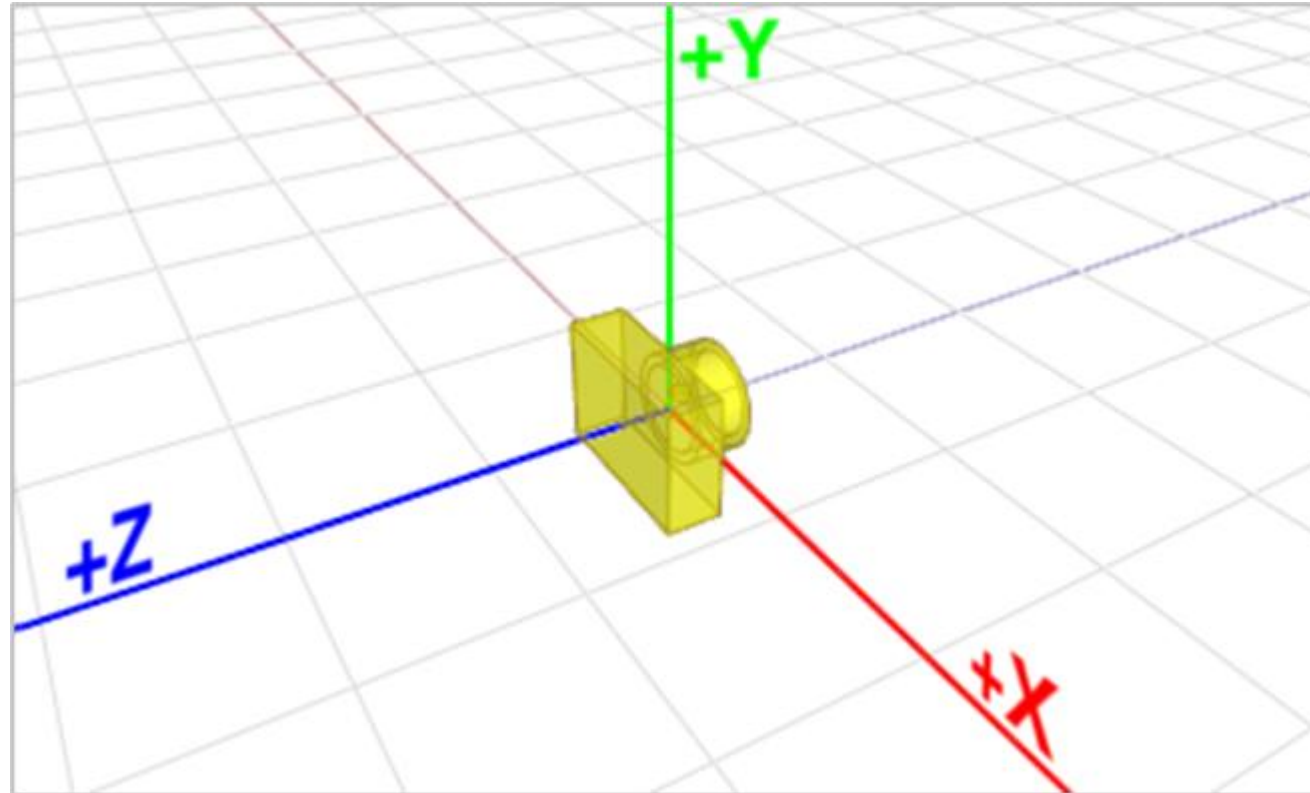
- “GL\_TRIANGLE\_FAN”을 이용하여 다음을 그려보시오
- 3\_7 예제를 응용함
- 지금까지 배운 것 활용





## ■ 카메라 기본 방향

- 원점에서 카메라가 바라보는 방향은 -z 이고, 카메라 위 방향은 +y



```
gluLookAt(0.0, 0.0, 0.0,    // 카메라 위치 (Eye) - 원점
          0.0, 0.0, -1.0,    // 바라볼 목표 지점 (Center) - Z축 음의 방향의 한 점
          0.0, 1.0, 0.0);    // 상향 벡터 (Up) - Y축 양의 방향
```

# 후면 제거(3\_7\_0)

- OpenGL 이 물체의 앞면만 그리도록 하거나 뒷면만 그리도록 하면 이미지 렌더링에 필요한 **처리 시간을 줄일 수 있음**
  - 깊이 테스트를 적용한 경우와 결과는 같다고 해도, 그랬지만 보이지 않는 것과 아예 그리지 않는 것의 성능 차이는 있음
- 정점들이 3차원 공간에서 2차원 화면 공간(Screen Space)으로 투영된 후, 해당 정점들이 정의된 순서대로 연결될 때 화면상에서 그려지는 방향을 기준으로, 그려지는 방향이 반시계(CCW) 방향이 전면!
  - +방향이 가까운 쪽
    - ✓ 이것을 기준으로 CCW방향이 +회전
- `glEnable(GL_CULL_FACE);` // 후면 제거기능 활성화(기본값 = 비활성)
- 3\_7\_a code에 다음 추가

```
#define GL_PI 3.1415f
bool bCull = true;

void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    if (bCull)
        glEnable(GL_CULL_FACE);
    else
        glDisable(GL_CULL_FACE);
}
```

# 후면 제거(3\_7\_0)

Try



- OpenGL 이 물체의 앞면만 그리도록 하거나 뒷면만 그리도록 하면 이미지 렌더링에 필요한 **처리 시간을 줄일 수 있음**
  - 깊이 테스트를 적용한 경우와 결과는 같다고 해도, 그랬지만 보이지 않는 것과 아예 그리지 않는 것의 성능 차이는 있음
- 정점들이 3차원 공간에서 2차원 화면 공간(Screen Space)으로 투영된 후, 해당 정점들이 정의된 순서대로 연결될 때 화면상에서 그려지는 방향을 기준으로, 그려지는 방향이 반시계(CCW) 방향이 전면!
  - +방향이 가까운 쪽
    - ✓ 이것을 기준으로 CCW방향이 +회전
- glEnable(GL\_CULL\_FACE); // 후면 제거기능 활성화(기본값 = 비활성)
- 3\_7\_a code에 다음 추가
- **결과가 제대로 나오나? 왜? 고쳐보자...**

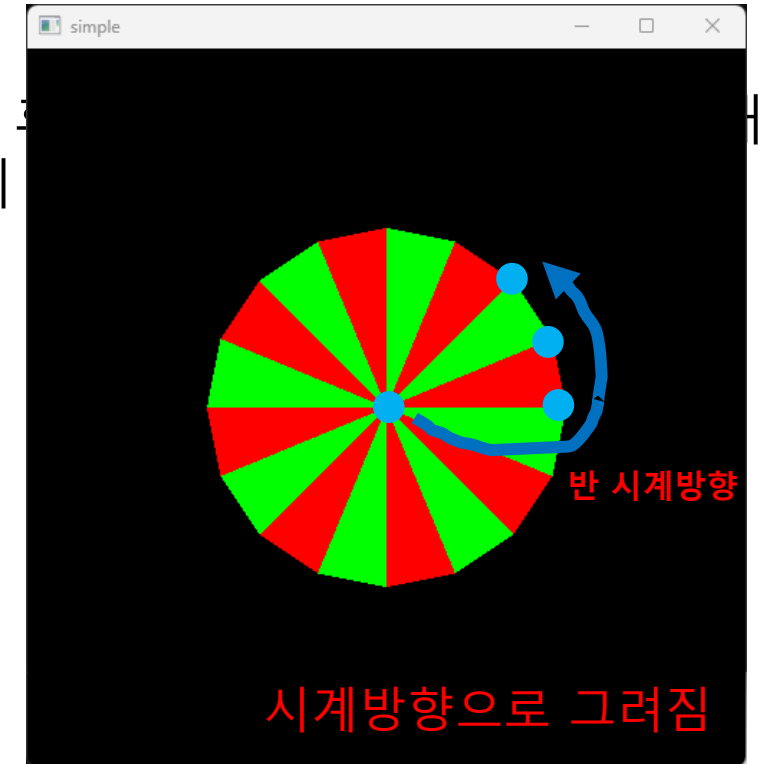
```
#define GL_PI 3.1415f
bool bCull = true;

void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    if (bCull)
        glEnable(GL_CULL_FACE);
    else
        glDisable(GL_CULL_FACE);
}
```

# 후면 제거(3\_7\_0)

- OpenGL 이 물체의 앞면만 그리도록 하거나 뒷면만 그리 도록 하면 이미지 렌더링에 필요한 **처리 시간을 줄일 수 있음**
  - 깊이 테스트를 적용한 경우와 결과는 같다고 해도, 그렸지만 보이지 않는 것과 아예 그리지 않는 것의 성능 차이는 있음
- 정점들이 3차원 공간에서 2차원 화면 공간(Screen Space)으로 투영된 후, 화면 상에서 그려지는 방향으로 연결될 때 화면상에서 그려지는 방향을 기준으로, 그려지는 방향이
  - +방향이 가까운 쪽
    - ✓ 이것을 기준으로 CCW방향이 +회전
- `glEnable(GL_CULL_FACE);` // 후면 제거기능 활성화 (기본값 = 비활성)
- 3\_7\_a code에 다음 추가
- **결과가 제대로 나오나? 왜?**
  - **점이 생성되는 순서 확인! sin / cos 순서에 따른 방향 반대...**



```
//어떤 면을 제거할 것인가?
glCullFace(GL_BACK); //후면

//앞면 설정
glFrontFace(GL_CCW); //반시계방향을 앞면
```