

## | 2025-04-15\_CG\_01\_사각형 그리기

### | 📁 전체 코드

```
#include <GL/glut.h>
#include <stdio.h>
#include <iostream>

// 📌
void RenderScene(void) {

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-0.5f, -0.5f, 0.5f, 0.5f);

    glFlush();
}

void SetupRC(void) {
    std::cout << "SetupRC" << std::endl;
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

int main(int argc, char** argv) {

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);

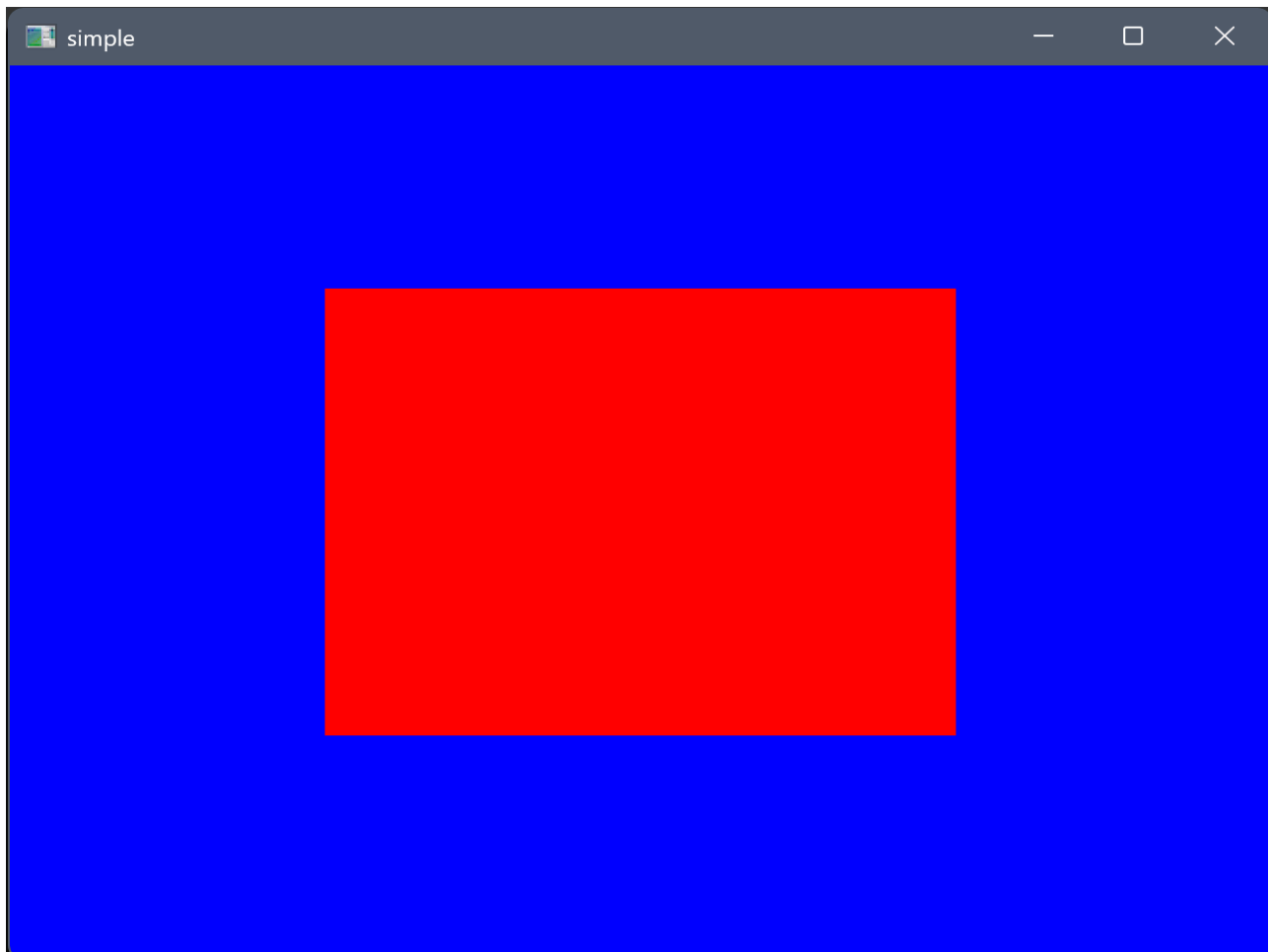
    glutCreateWindow("simple");

    SetupRC();

    glutDisplayFunc(RenderScene);

    glutMainLoop();
}
```

### | 📝 출력 결과



---

## 📁 코드 설명 - RenderScene(void)

### 📝 glClear()

```
// 이전 프레임에서 그려졌던 내용을 지우고, 설정된 배경색으로 다시 채움  
glClear(GL_COLOR_BUFFER_BIT);
```

- OpenGL에서는 `glClear()` 함수를 이용해서 화면을 지우는데, 어떤 버퍼를 지울지 지정해줘야 한다.
  - 이때 사용하는 플래그 중 하나가 `GL_COLOR_BUFFER_BIT` 이다.
    - `GL_COLOR_BUFFER_BIT` : 색상 버퍼를 지운다.
- 면에 그리는 모든 픽셀의 색상 정보는 `Color Buffer`에 저장되는데, `glClear(GL_COLOR_BUFFER_BIT);` 라고 쓰면, **현재 설정된 배경색으로 화면 전체를 지우는 동작**을 수행한다.
- 배경색은 보통 `glClearColor()` 로 설정한다.
- 코드에서는 `SetupRC(void)` 에 `glClear()` 를 설정했다.

```
/* 예시 흐름이다. */
```

```
glClearColor(0.5f, 0.7f, 1.0f, 1.0f); // 배경색 설정 (하늘색)
```

```
glClear(GL_COLOR_BUFFER_BIT); // // 화면 초기화: 설정된 색으로 화면을 싹 지움
```

```
// 도형 그리기
```

```
glColor3f(1.0f, 0.0f, 0.0f);
```

```
glRectf(-0.5f, -0.5f, 0.5f, 0.5f);
```

```
glFlush();
```

## I 🖋️ 사각형 그리기

```
// 그릴 도형의 색상 설정
```

```
glColor3f(1.0f, 0.0f, 0.0f);
```

```
// 사각형 그리기
```

```
// 사각형의 좌표는 (-0.5, -0.5)에서 (0.5, 0.5)까지
```

```
glRectf(-0.5f, -0.5f, 0.5f, 0.5f);
```

```
// 드로잉 명령 전달
```

```
glFlush();
```

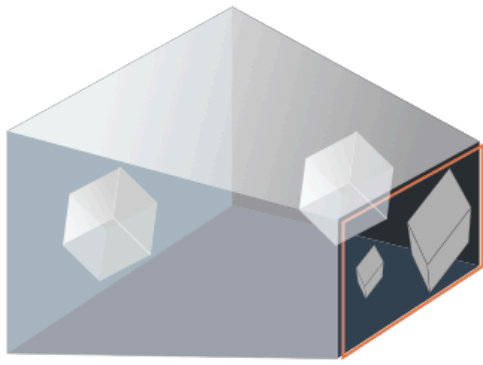
---

## I 📁 추가 설명

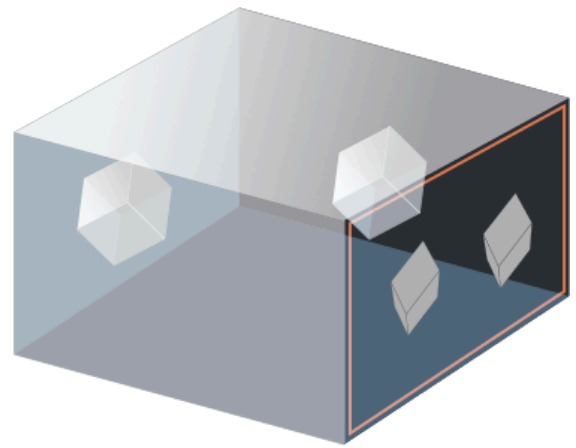
### I Projection

그래픽스에서 카메라의 Projection은 주로 두 가지 타입으로 나뉜다. 하나는 **Perspective**, 하나는 **Orthographic** <sup>[1]</sup>

- **Orthographic**은 주로 2D 어플리케이션에서 사용되고, **Perspective**는 주로 3D 어플리케이션에서 사용된다.
- Perspective는 원근감을 반영한 것, Orthographic은 원근감을 반영하지 않은 것이다.



· Perspective projection



Orthographic projection

## 📝 기본 OpenGL Orthographic

- 기본적으로 OpenGL의 **기본 설정 좌표계(카메라 시야)** 는 아래와 같다.
  - 화면 가운데:  $(0,0)$
  - 왼쪽 끝:  $x = -1.0$
  - 오른쪽 끝:  $x = 1.0$
  - 아래  $y = -1.0$
  - 위  $y = 1.0$
- 위 코드의 사각형을 요약하면 아래와 같다.

좌표계 기준 (기본 OpenGL Orthographic)

```
(-1,1)      (0,1)      (1,1)
+-----+-----+
|               |
|               |
| 빨간 사각형   |
| (-0.5, -0.5) ~ |
| (0.5, 0.5)    |
|               |
+-----+-----+
(-1,-1)     (0,-1)     (1,-1)
```

## 📝 glOrtho()

- OpenGL에서 사용하는 **직교 투영(Orthographic Projection)**을 수동으로 설정할 수 있는 함수.

```
glOrtho(좌, 우, 아래, 위, 앞, 뒤);
```

## 📁 정사각형을 그렸는데, 왜 직사각형?

## I 📝 원인:

- `glutInitWindowSize(640, 480);` 으로 윈도우를 설정했다.
- `glOrtho()` 도 설정하지 않았다.
- OpenGL 좌표계는  $-1.0 \sim 1.0$ 의 정사각형 기준이다. 그런데 실제 **윈도우가 직사각형이라서 투영된 결과가 왜곡**되는 것이다.
- 해결을 위해서 **윈도우 크기 고정** 뿐만 아니라, **좌표계** 및 **viewport** 조정까지 같이 해야 한다.

## I 📝 과정 1. `glutInitWindowSize(500, 500)`으로 변경

```
glutInitWindowSize(500, 500);
```

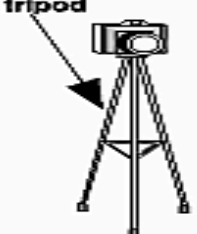
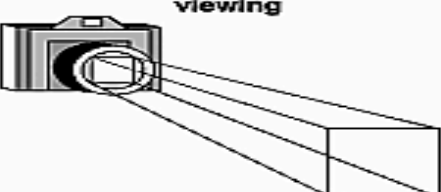
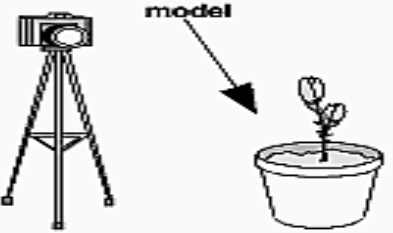
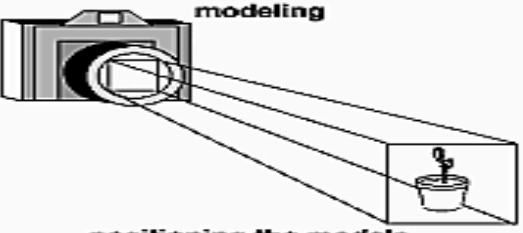

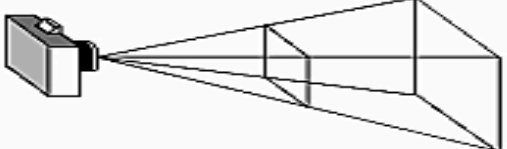


- 처음에 표시되는 사각형이 정사각형으로 보인다.
- 하지만 윈도우 크기를 변경한다면, 직사각형으로 보일 것이다.

## I 📝 과정 2. Viewport 조정 (`glViewport()`)

- viewport는 OpenGL이 실제로 렌더링할 영역(픽셀 범위)이다.
- `glViewport()` 는 기본적으로 전체 **윈도우 사이즈와 동일**하게 설정된다.
- Viewport 영역이 비대칭하거나 크기가 다르면, OpenGL 좌표가 왜곡되어 렌더링된다.
  - 즉, `glRectf()` 로  $-0.5 \sim 0.5$  좌표를 줘도 정사각형이 사각형처럼 찌그러질 수 있음.
  - **Viewport 비율과 ortho 비율이 일치해야 왜곡이 없다.**

```
void RenderScene(void) {  
  
    // ✨ 뷰포트 설정 (윈도우 크기와 동일하게 설정)  
    // 시작좌표 + width-height  
    glViewport(0, 0, 500, 500);  
  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1.0f, 0.0f, 0.0f);  
    glRectf(-0.5f, -0.5f, 0.5f, 0.5f);  
    glFlush();  
}
```

- 창의 크기를 늘리거나 줄여도 viewport는 고정되어 있으므로
  - **viewport 밖의 공간은 아무것도 그려지지 않아 빈 영역으로 늘어난다.**

With a Camera	With a Computer
 <p>tripod</p>	 <p>viewing</p> <p>positioning the viewing volume in the world</p>
 <p>model</p>	 <p>modeling</p> <p>positioning the models in the world</p>
 <p>lens</p>	 <p>projection</p> <p>determining shape of viewing volume</p>
 <p>photograph</p>	 <p>viewport</p>

### 📌 🖌️ 과정 3. 좌표계 설정 ( `glOrtho()` )

- `glOrtho(left, right, bottom, top, near, far)`
- OpenGL의 2D 좌표계(투영 행렬)를 설정한다.
- 예를 들어 `left=-1.0, right=1.0` 과 `bottom=-1.0, top=1.0` 은 정사각형 좌표계를 만든다.
- Viewport와 이 설정이 **비율상 일치**해야 화면에 정사각형이 정확히 출력된다.

```
void RenderScene(void) {

    glClear(GL_COLOR_BUFFER_BIT);

    // 🚩
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glViewport(0, 0, 500, 500);
```

```
// ✦ 직교 투영 설정 -> 카메라가 어떻게 세상을 바라보는지(투영공간)을 정의한다.
// `(-1,-1)` ~ `(1,1)` 안에 있는 물체만 보이게 된다.
glOrtho(-1, 1, -1, 1, 1, -1);

// ✦
glMatrixMode(GL_MODELVIEW); // object 좌표계로 다시 이동
glLoadIdentity(); // 좌표계 초기화

glColor3f(1.0f, 0.0f, 0.0f);
glRectf(-0.25f, -0.25f, 0.25f, 0.25f);

// 드로잉 명령 전달, 명령을 강제로 실행하여 즉시 렌더링
glFlush();
}
```

## 📎 코드 설명

일반적으로 투영 행렬 (GL\_PROJECTION)을 먼저 설정한 후, 모델뷰 행렬 (GL\_MODELVIEW)을 설정한다.

```
glMatrixMode(GL_PROJECTION);
```

- 현재 행렬 모드를 투영 행렬(projection matrix)로 설정한다.
- 시야, 카메라 위치, 어떤 공간을 보여줄지 결정한다.
- 이후에 사용하는 행렬 관련 함수( `glLoadIdentity` , `glOrtho` , `gluPerspective` 등)는 **투영 행렬**에 적용
- 투영행렬은 **카메라의 시야각, 원근감, 절두체(Frustum)** 등을 정의하는 데 쓰인다.

```
glMatrixMode(GL_MODELVIEW);
```

- 이후 행렬 연산을 **모델뷰 행렬(modelview matrix)** 에 적용한다.
- 모델뷰 행렬은 **모델 변환 + 뷰(카메라) 변환**을 동시에 처리한다.
  - `glTranslatef` , `glRotatef` , `glScalef` 등
- 물체의 위치, 회전, 확대/축소 등을 정의한다.

```
glLoadIdentity();
```

- 모든 변환을 초기화하고 "처음 상태"로 만든다.
- 선택된 행렬(GL\_PROJECTION 또는 GL\_MODELVIEW)에 **단위 행렬(identity matrix)** 을 설정한다.

🔗 OpenGL에서 변환은 계속 누적된다.

- 어떤 객체를 회전시키고 또 이동시키고, 또 스케일링도 하면... 그게 전부 누적돼서 한꺼번에 적용된다.
- 그런데 만약, **새로운 도형**을 그리고 싶다? 또는 **카메라를 새로 초기화**하고 싶다?
  - 이럴 땐 이전에 누적된 변환을 다 **지우고** 처음 상태로 돌아가야 한다.
  - 그래서 `glLoadIdentity()`를 써서 행렬을 다시 **단위행렬로 초기화**하는 것이다.

## I 최종 코드

```
#include <GL/glut.h>
#include <stdio.h>
#include <iostream>

void RenderScene(void) {
    // 1. 이전 프레임 지우기 (배경색으로 클리어)
    glClear(GL_COLOR_BUFFER_BIT);

    // 2. 출력 영역 설정 (화면 좌표계 설정)
    glViewport(0, 0, 500, 500);

    // 3. 투영 행렬 설정 (카메라 시야 설정)
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, 1, -1); // 직교 투영 (시야범위 지정)

    // 4. 모델뷰 행렬 설정 (객체 위치/회전 등 변환) -> 아직 `glTranslatef()`, `glRotatef()`,
    `glScalef()` 같은 변환 함수가 전혀 없는 상태.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // 5. 도형 그리기
    glColor3f(1.0f, 0.0f, 0.0f);           // 빨간색 설정
    glRectf(-0.25f, -0.25f, 0.25f, 0.25f); // 사각형 그리기

    // 6. 명령 즉시 실행
    glFlush();
}

void SetupRC(void) {
    std::cout << "SetupRC" << std::endl;
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

int main(int argc, char** argv) {

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```



```

glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);

glutCreateWindow("simple");

SetupRC();

glutDisplayFunc(RenderScene);

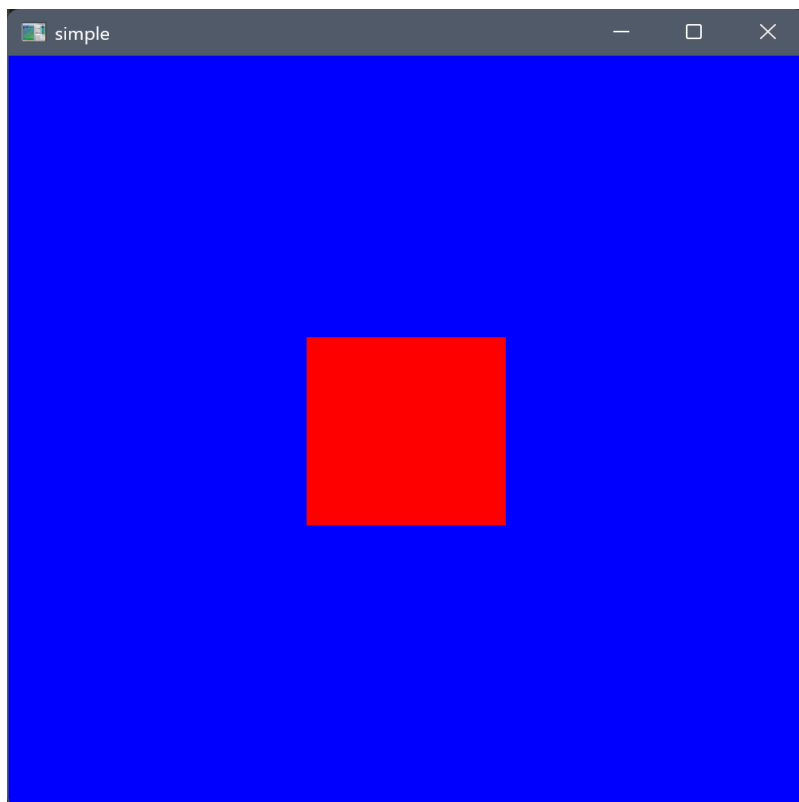
glutMainLoop();
}

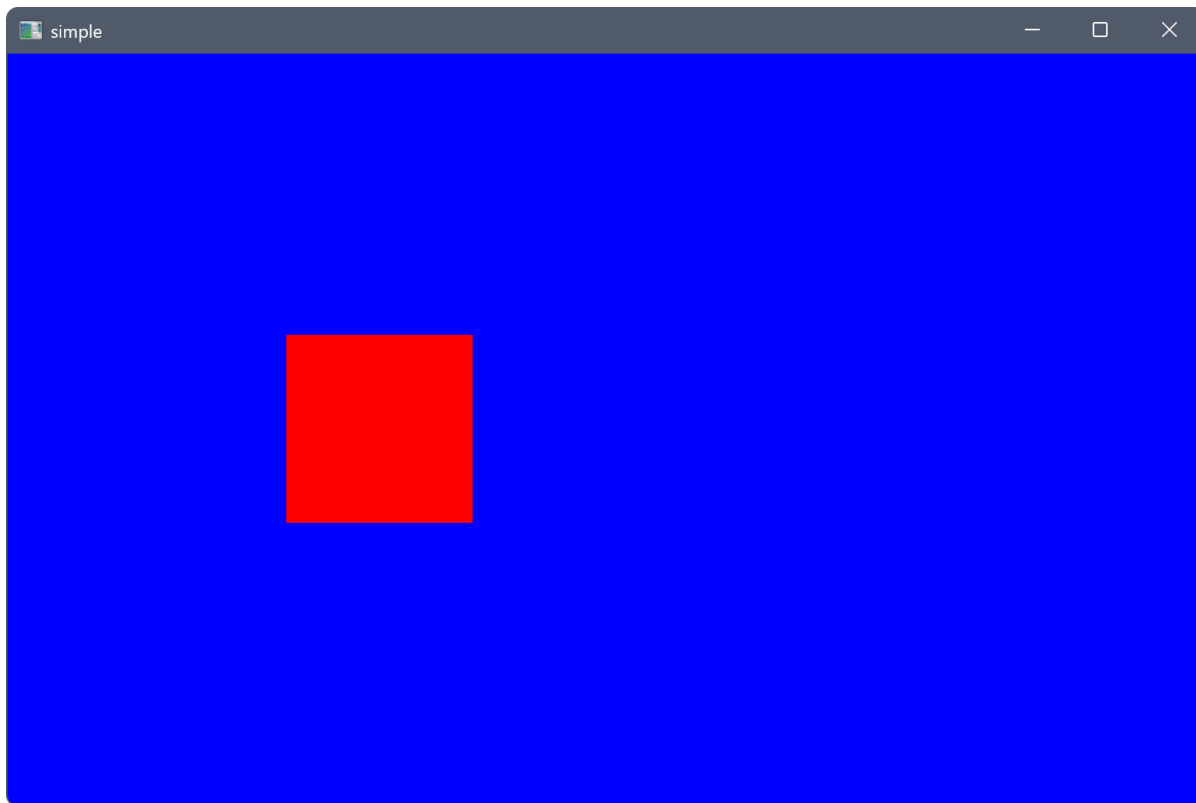
```

## 📎 흐름

- |   |                      |
|---|----------------------|
| 1. <code>glClear(...)</code>                | → 이전 프레임 제거 (배경 초기화) |
| 2. <code>glViewport(...)</code>             | → 화면에 보여질 영역 지정      |
| 3. <code>glMatrixMode(GL_PROJECTION)</code> | → "카메라 렌즈" 설정 시작     |
| 4. <code>glLoadIdentity()</code>            | → 투영행렬 초기화           |
| 5. <code>glOrtho(...)</code>                | → 투영 범위(시야 영역) 설정    |
| 6. <code>glMatrixMode(GL_MODELVIEW)</code>  | → "물체 위치/회전" 설정 시작   |
| 7. <code>glLoadIdentity()</code>            | → 모델뷰행렬 초기화          |
| 8. <code>glColor3f(...)</code>              | → 색상 설정              |
| 9. <code>glRectf(...)</code>                | → 도형 그리기             |
| 10. <code>glFlush()</code>                  | → 화면에 출력             |

## 📎 결과





- 윈도우창을 늘려도 사각형은 그대로.

## 📝 정리

순서	설명	효과
1. <code>glutInitWindowSize()</code> 만 설정	창 크기만 정사각형	내부 좌표계가 정사각형이 아니면 왜곡 발생
2. + <code>glViewport()</code>	렌더링 영역도 정사각형으로 지정	창이 늘어나거나 줄어들어도 비율 유지 가능
3. + <code>glOrtho()</code>	좌표계 자체도 정사각형	최종적으로 정사각형 객체가 왜곡 없이 출력됨

## 📁 추가 설명

### 📝 현재 설정

```
glViewport(0, 0, 500, 500)
```

→ 스크린 영역: 500x500 픽셀의 윈도우 전체 영역을 사용

```
glOrtho(-1, 1, -1, 1, 1, -1)
```

→ 정규화 공간(NDC): (-1, -1) ~ (1, 1) 영역이 “카메라에 보이는 범위”

```
glRectf(-0.25, -0.25, 0.25, 0.25)
```

→ 도형 좌표: 정사각형의 너비/높이 =  $0.5 + 0.5 = 1.0$  (좌표계 단위)

## ! ? 사각형의 크기는 좌표상으로 1도 안되는데, 왜 화면에서 꽤 크게 보이는가?

- 정규화된 좌표가 화면 전체에 매핑되기 때문이다.
- `glOrtho(-1, 1, -1, 1, ...)` 은 가시 영역 전체가  $2 \times 2$  단위로 되어 있다.
  - 이 전체 영역이 `glViewport` 에 따라  $500 \times 500$  픽셀로 매핑되는 것.

설정 예시	효과
<code>glOrtho(-10, 10, -10, 10, 1, -1)</code>	도형이 작게 보임
<code>glOrtho(-0.5, 0.5, -0.5, 0.5, 1, -1)</code>	도형이 매우 크게 보임 (사각형이 꽉 찰 수도 있음)

---

### 1. [블로그-Projection](#)