OpenGL10

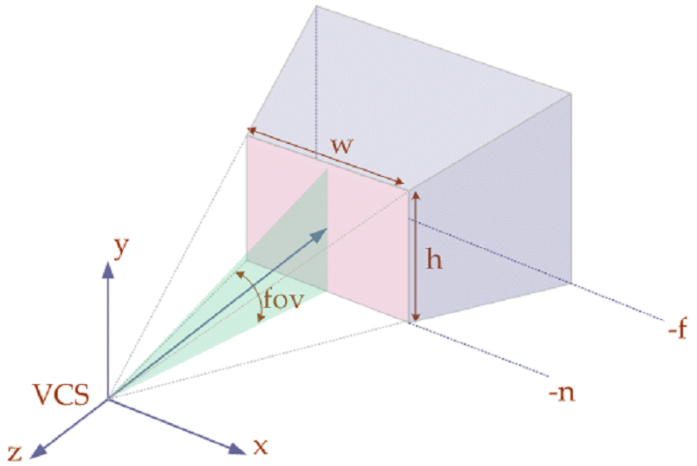
가상현실론

2021/03/24

목차

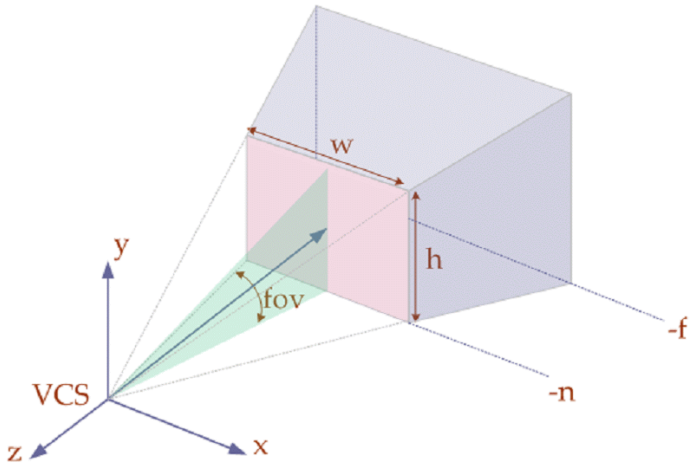
- gluPerspective()
- 시야각과 카메라 렌즈
- 투상 파이프라인
- GL의 뷰포트 변환

gluPerspective()



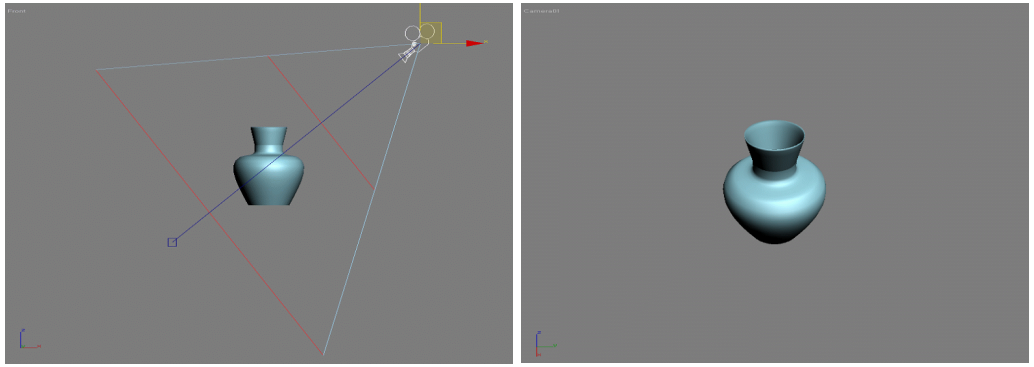
- void gluPerspective(GLdouble fov, GLdouble aspect, GLdouble near, GLdouble far);
  - 원근 투상의 가시 부피를 설정하는 데 있어서 glFrustum() 함수보다 직관적이고 손쉬운 방법.
  - 시선은 가시 부피의 한 가운데 통과 . 즉 시선을 중심으로 가시 부피는 대칭적으로 놓여있음.
  - fov(field of view): 상하 y축 방향의 시야각(0 -180도)
  - aspect: 뷰 윈도우의 종횡비 (Aspect Ratio). 폭(width)를 높이(height)로 나눈 값. 물체 모습의 왜곡을 막으려면 이 종횡비가 뷰 포트의 종횡비와 같아야 함.
  - near, far: glFrustum() 함수의 전방 절단면 및 후방 절단면에 해당. 항상 양수여야 함.

gluPerspective()

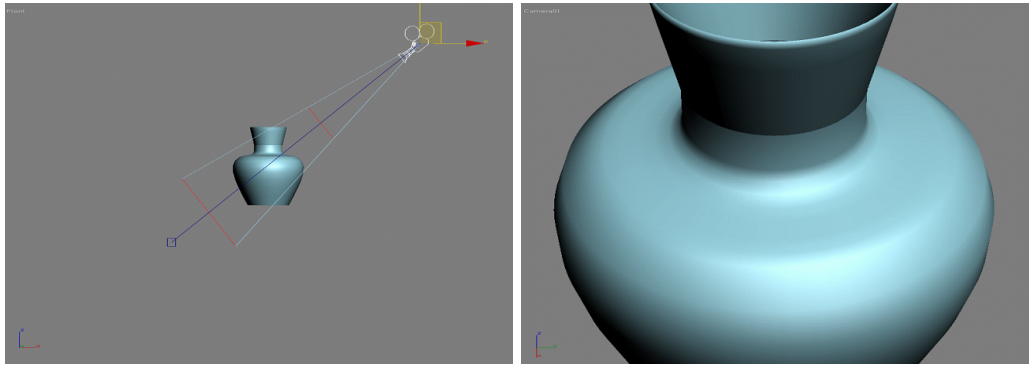


- bottom = -near · tan(fov/2)
- left = bottom · aspect
- top = near · tan(fov/2)
- right = top · aspect

시야각과 카메라 렌즈



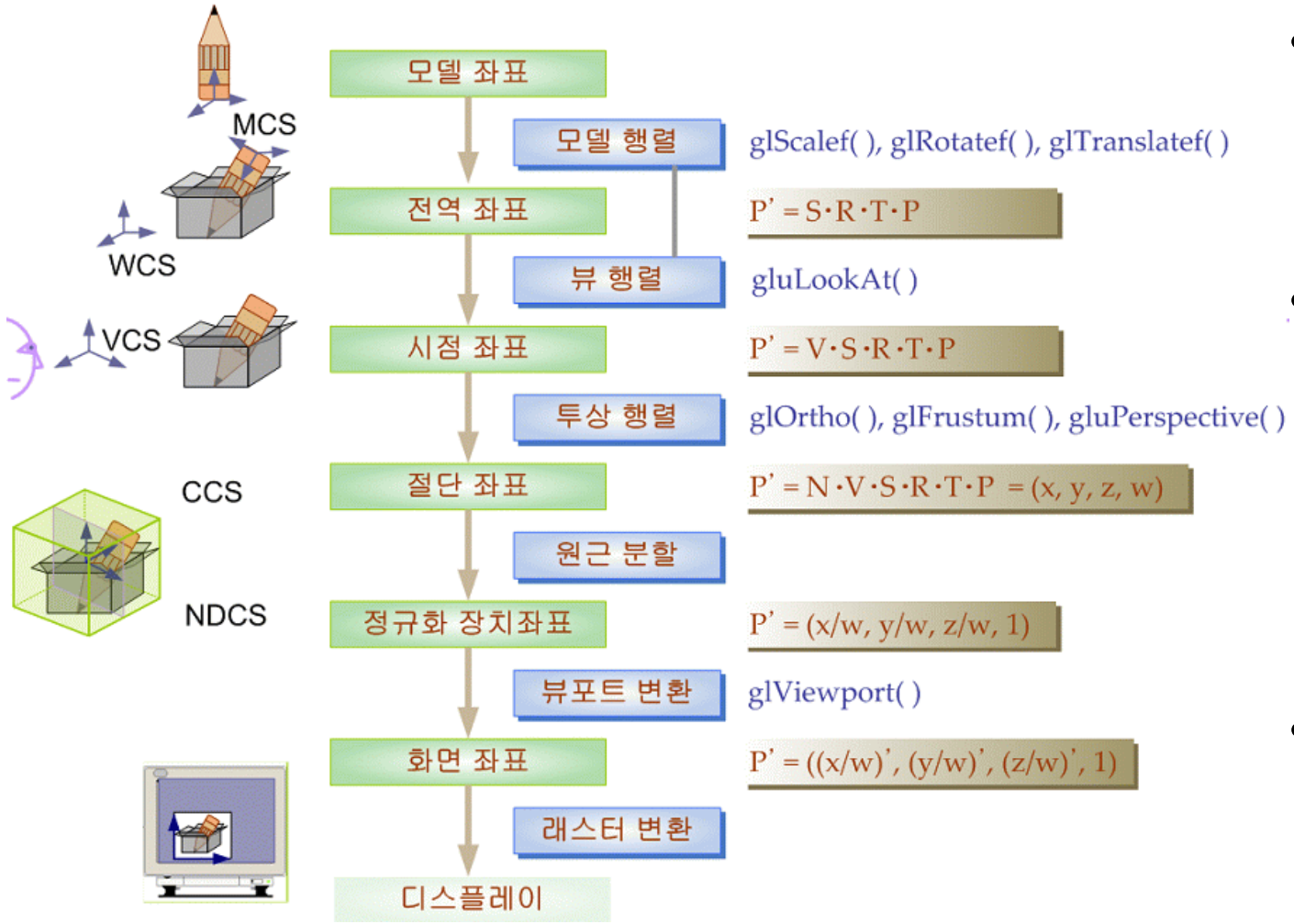
FOV: 85deg. <-> 20mm



FOV:24deg. <-> 85mm lens

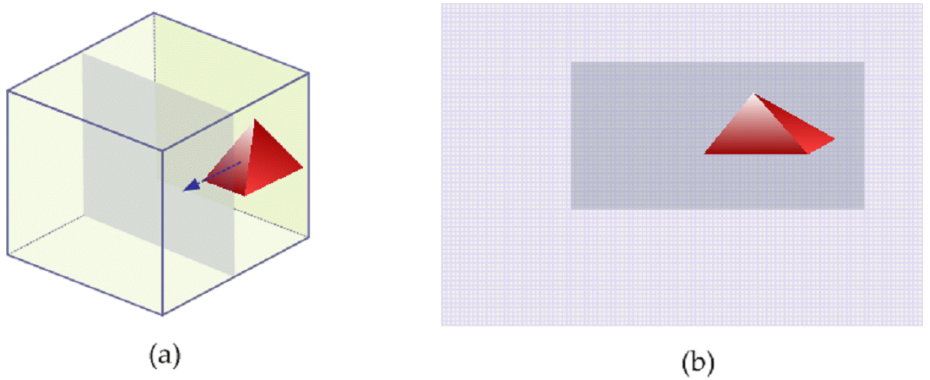
- 시야각을 넓히는 것은 카메라로 말하면 광각 렌즈를 끼우는 것과 같음.
- 시야각을 좁히는 것은 망원 렌즈를 끼우는 것과 같음.
- 카메라 렌즈는 초점 거리(focal length)로 표시되며, 50mm보다 작은 것을 광각 렌즈, 큰 것을 망원 렌즈라고 함.

투상 파이프라인



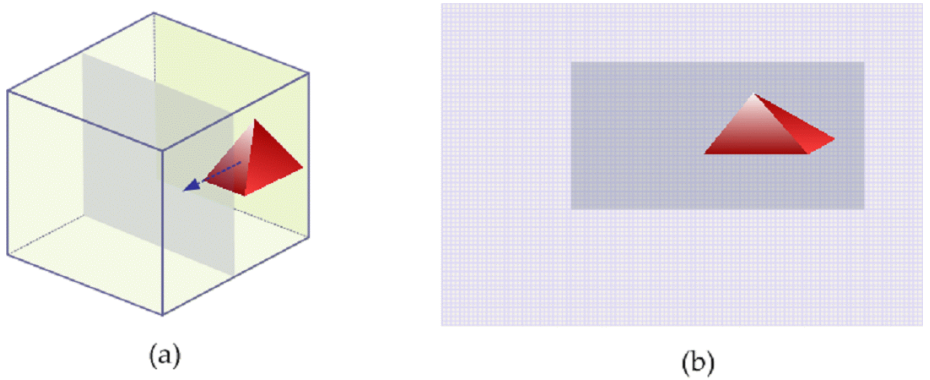
- 1단계 투상
  - 물체의 정점이 절단좌표계로 변환. 가시부피 내 모든 정점은 동차좌표로 표시.
- 2단계 투상
  - 동차좌표를 3차원 좌표로 변환(원근 분할).
  - 원근 분할 결과 물체 좌표는 정규화 장치 좌표계(NDCS: Normalized Device Coordinate System)으로 바뀜
- 3단계 투상
  - 정규화 가시 부피의 z=0평면이 화면의 윈도우 또는 뷰 포트로 투상.

GL의 뷰포트 변환



- 절단 작업 후에 투상 변환의 마지막 단계에서 3차원 정점이 2차원 정점으로 사상되어야 함.
- (a)의 정규화 가시 부피에 있는 물체가 (b)처럼 2차원 화면에 맺히는 것을 의미.
- 우선 원근 분할에 의해 물체 정점의 좌표를 (x, y, z, w)에서 (x/w, y/w, z/w, 1) 형식으로 변환.
    - (x', y', z', 1) = (x/w, y/w, z/w, 1)

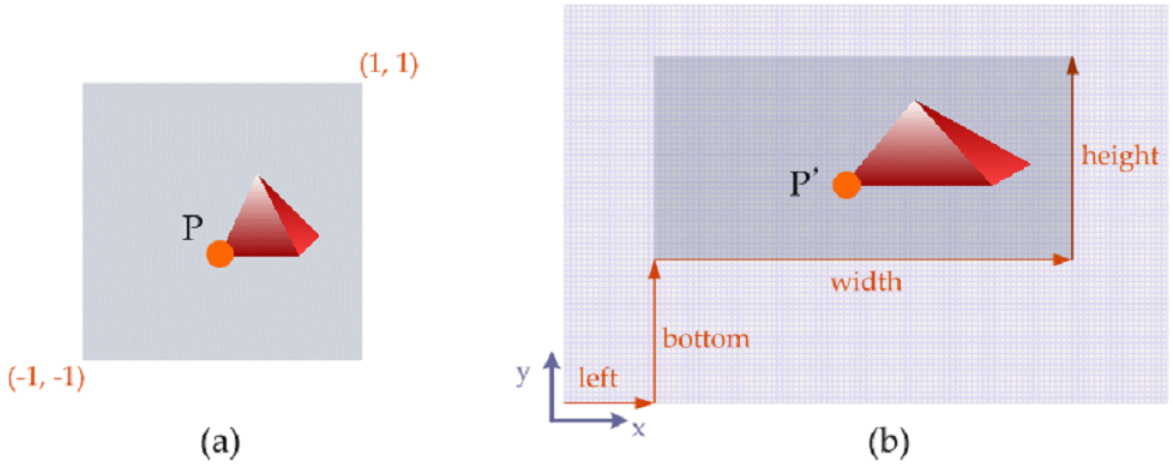
GL의 뷰포트 변환



(a) (b)

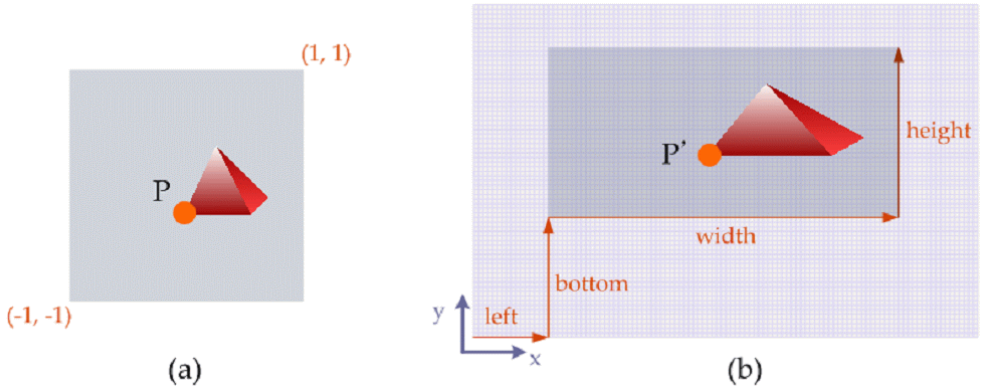
- 정규화 장치 좌표계 (NDCS: Normalized Device Coordinate System): 원근 분할의 결과 좌표계
- 뷰포트 변환 (Viewport Transformation): (a)-> (b)처럼 정규화 장치 좌표계에서 화면 좌표계로 변환하는 작업.
- 화면 좌표계(SCS: Screen Coordinate System): 뷰 포트 좌표계 (Viewport Coordinate System), 혹은 윈도우 좌표계 (Window Coordinate System)이라고도 함.

뷰포트 설정



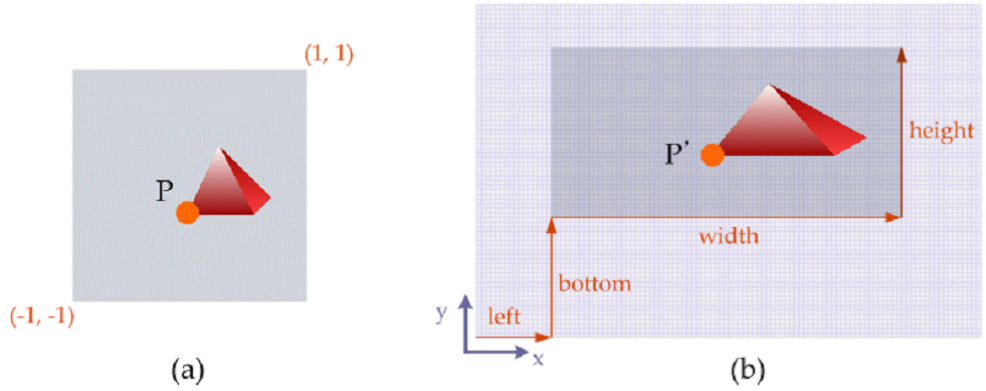
- void glViewport(GLint left, GLint bottom, GLsizei, width, GLsizei, height);
  - 정규화 장치 좌표계에서 화면 좌표계로 변환하기 위한 함수
  - 뷰 포트의 위치 및 크기를 지정하는 함수.
  - left, bottom은 뷰 포트 좌하단의 좌표. 이를 중심으로 폭(width), 높이(height)에 의해 뷰 포트 사각형이 정의.
  - 좌표 값은 화면의 좌하단을 원점으로 하고, 오른쪽으로 x, 위쪽으로 y 방향으로 진행하는 GL의 화면 좌표계를 기준으로 함.

뷰포트 변환



- 정점 P의 좌표를 (x, y), 이를 viewport로 사상시킨 (b)의 정점 P'의 좌표를 (x', y')이라고 하면,
  - $x' = \frac{x-(-1.0)}{(1.0)-(-1.0)} \times width + left$
  - $y' = \frac{y-(-1.0)}{(1.0)-(-1.0)} \times height + bottom$
- 위의 변환은 1) 정규화 장치 좌표값을 (0, 2.0) x (0, 2.0)의 좌표계로 normalize하고 2) viewport의 위치 (원점이 (left, botoom)이고 크기가 width x height)로 매핑하는 것을 알 수 있음.

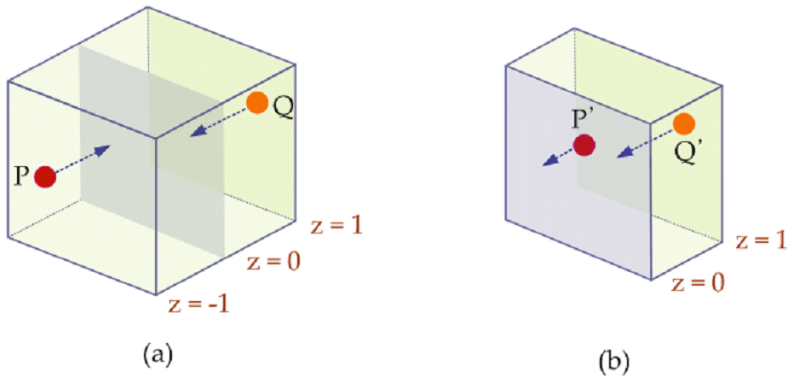
뷰포트 변환



$$\mathrm{P'} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & 0 & left + \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & bottom + \frac{height}{2} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- 부동 소수 정밀도의 좌표로부터 정수 정밀도 화면 좌표로 바꾸는 과정을 래스터 변환(Rasterization) 또는 스캔 변환(Scan Conversion)이라고 함.

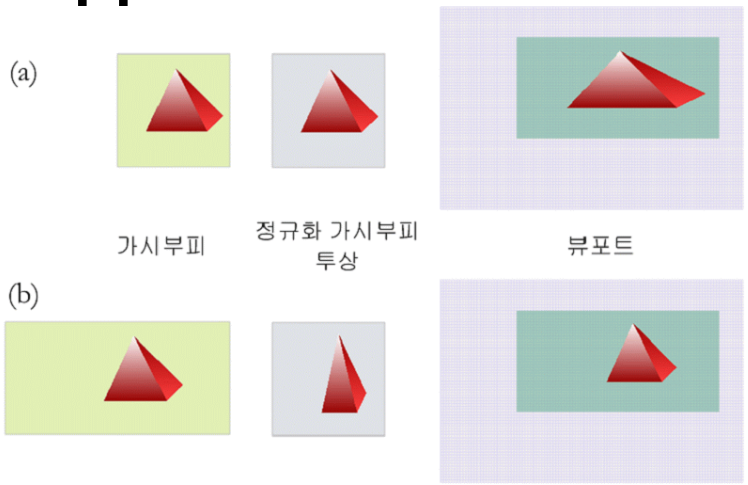
z축 재 정규화



- GL을 포함한 대부분의 소프트웨어는 z값을 (-1, 1)에서 (0, 1)로 재 정규화(Renormalization)함.
  - 정점 사이의 상대적인 깊이는 유지됨
  - 정규화 가시부피를 원점을 중심으로 해서 z축 방향으로 1/2 만큼 크기조절 변환을 가한 후, z 방향으로 1/2 만큼 이동.

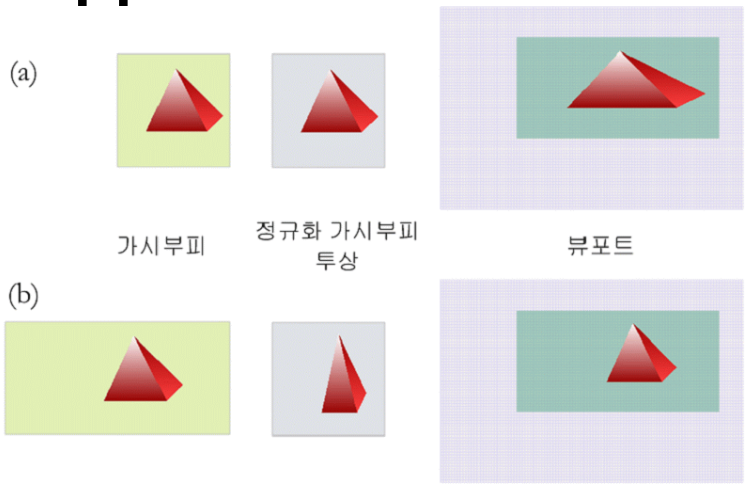
- $$P' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{width}{2} & 0 & 0 & left + \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & bottom + \frac{height}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

가시 부피와 뷰 포트



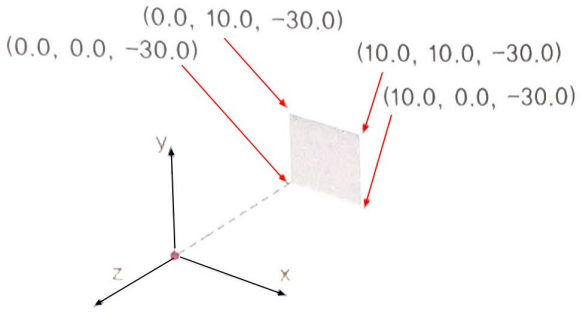
- 가시 부피의 종횡비와 뷰 포트의 종호이비가 일치하지 않으면 왜곡이 야기.
- 이를 방지하기 위해서는 가시 부피의 종횡비와 뷰 포트의 종횡비를 일치시켜야 함.
- 평행 투상일 경우 가시 부피의 종횡비가 glOrtho() 함수, 원근 투상이라면 glFrustum()이나 gluPerspective()함수가 가시 부피의 종횡비를 결정. 이에 맞춰 top-bottom과 left-right의 비율을 조정하면 됨

가시 부피와 뷰 포트

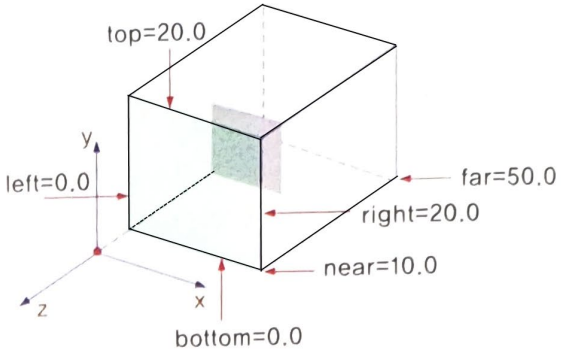


- (a)의 경우 가시 부피 내부 뷰 윈도우 종횡비가 1:1인 경우로 정규화 가시 부피 투상면의 종횡비 1:1과 동일하게 됨.
- (a)에서 뷰포트 종횡비를 2:1로 설정하면 물체는 왜곡됨.
- 반면 (b)에서 가시 부피의 종횡비를 2:1로 잡으면 정규화 가시 부피에서는 왜곡되어 보이나 이를 뷰 포트로 사상시키면 원래 모습으로 회복.
- 즉, **가시 부피의 종횡비와 뷰 포트의 종횡비만 일치**시키면 물체는 왜곡되지 않음.

가시 부피와 뷰 포트



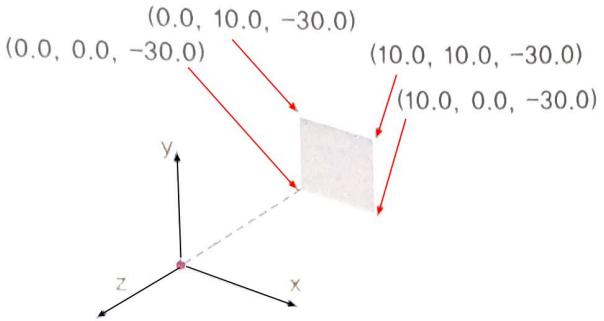
[그림 7-44] 물체 모델링



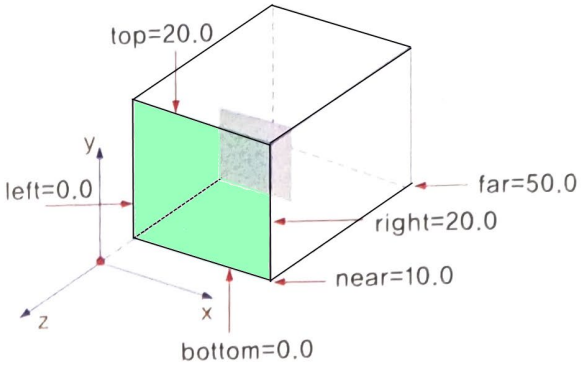
[그림 7-45] 가시 부피 설정

- 예) 사각형
  - glBegin(GL_POLYGON);
  - glVertex3f(0.0, 0.0, -30.0);
  - glVertex3f(10.0, 0.0, -30.0);
  - glVertex3f(10.0, 10.0, -30.0);
  - glVertex3f(0.0, 10.0, -30.0);
  - glEnd();
- 프로그램 시작시에는 전역 좌표계와 모델 좌표계가 일치하므로 전역좌표계로 생각

가시 부피와 뷰 포트



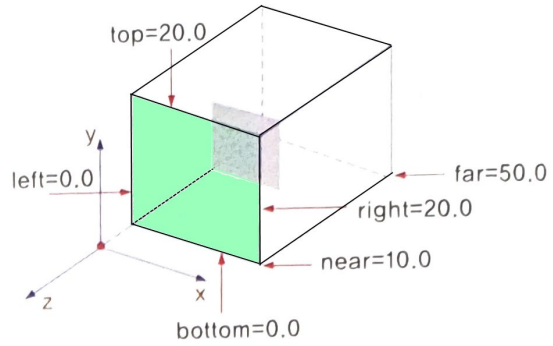
[그림 7-44] 물체 모델링



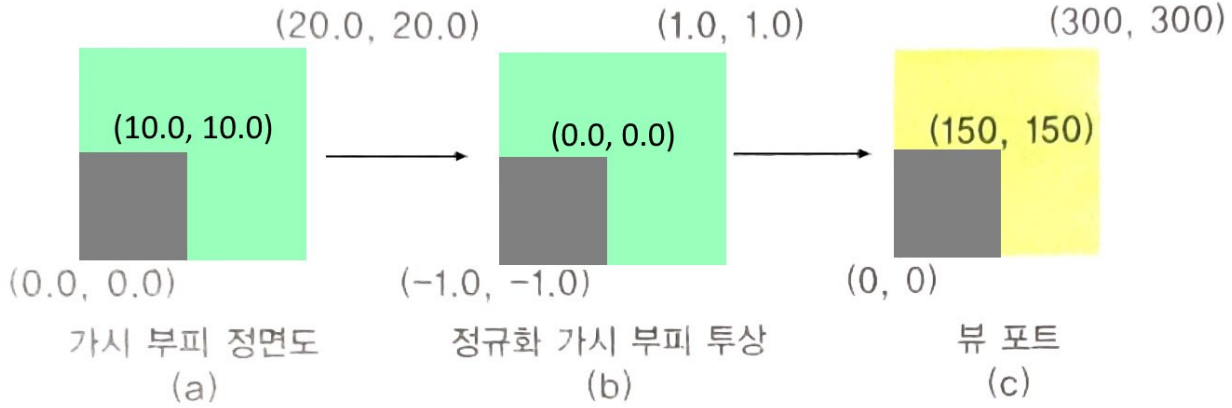
[그림 7-45] 가시 부피 설정

- 시점변환이 없기 때문에 전역 좌표계가 시점좌표계이기도 함
- 카메라는 –z축을 바라고고 있음.
- 시점 좌표계를 기준으로 glOrtho(0.0, 20.0, 0.0, 20.0, 10.0, 50.0)의 투상 함수를 적용
- 7-45의 가시 부피가 형성

가시 부피와 뷰 포트

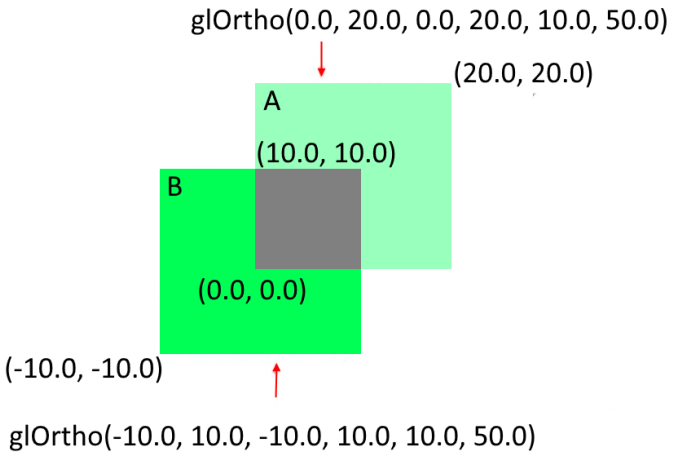


[그림 7-45] 가시 부피 설정



- (a): [7-45]의 가시 부피를 정면에서 바라본 것
- (b): (a)를 정규화 변환
- (c): 정규화 변환 후 glViewport(0, 0, 300, 300) 실행

가시 부피 설정에 따른 차이점



- 가시 부피 설정에 따라서 뷰 포트에 표시되는 물체의 위치가 변함.
- 가시 부피를 설정하는 것은 물체를 고정한 상태에서 물체를 관찰하기 위해 창을 움직이는 것과 같음.

예제

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, -1.0, 1.0, 1.0, 0.0);
    glutSolidTeapot(0.5);
    glFlush();
}
void myInit()
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 20.0, 0.0, 20.0, 10.0, 50.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

```
void keyboard(unsigned char key, int x, int y)
{
   switch (key) {
   case 27:
      exit(0);
      break;
   case '1' :
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(0.0, 20.0, 0.0, 20.0, 10.0, 50.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glutPostRedisplay();
      break;
   case '2' :
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-10.0, 10.0,-10.0, 10.0, 10.0, 50.0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      glutPostRedisplay();
      break;
   }
}
```

예제

```
int main(int argc, char** argv)
{
   m_width = 500; m_height = 500;
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
   glutInitWindowSize(m_width, m_height);
   glutInitWindowPosition(0, 0);
   glutCreateWindow("OpenGL viewport division example");
   myInit();
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

```
void myInit()
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-2.0, 2.0, -2.0, 2.0, 0.5, 5.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

예제

```
void DrawScene()
{
   glColor3f(0.7, 0.7, 0.7);
   glPushMatrix();
   glTranslatef(0.0, -1.0, 0.0);
   glBegin(GL_QUADS);
   glVertex3f(2.0, 0.0, 2.0);
   glVertex3f(2.0, 0.0, -2.0);
   glVertex3f(-2.0, 0.0, -2.0);
   glVertex3f(-2.0, 0.0, 2.0);
   glEnd();
   glPopMatrix();
   glColor3f(0.3, 0.3, 0.7);
   glPushMatrix();
   glTranslatef(0.0, 0.0, -0.5);
   glutWireTeapot(1.0);
   glPopMatrix();
}
```

예제

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glViewport(0.0, 0.0, m_width/2, m_height/2);
   glPushMatrix();
   gluLookAt(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   DrawScene();
   glPopMatrix();
   glViewport(m_width/2, 0.0, m_width / 2, m_height / 2);
   glPushMatrix();
   gluLookAt(1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   DrawScene();
   glPopMatrix();
   glViewport(0.0, m_height/2, m_width / 2, m_height / 2);
   glPushMatrix();
   gluLookAt(0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0);
   DrawScene();
   glPopMatrix();
   glViewport(m_width / 2, m_height / 2, m_width / 2, m_height / 2);
   glMatrixMode(GL_PROJECTION);
   glPushMatrix();
   glLoadIdentity();
   gluPerspective(30, 1.0, 3.0, 50.0);
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
   gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   DrawScene();
   glPopMatrix();
   glMatrixMode(GL_PROJECTION);
   glPopMatrix();
   glFlush();
}
```