LEC19: Normalization & Orthogonal Projection

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

Contents

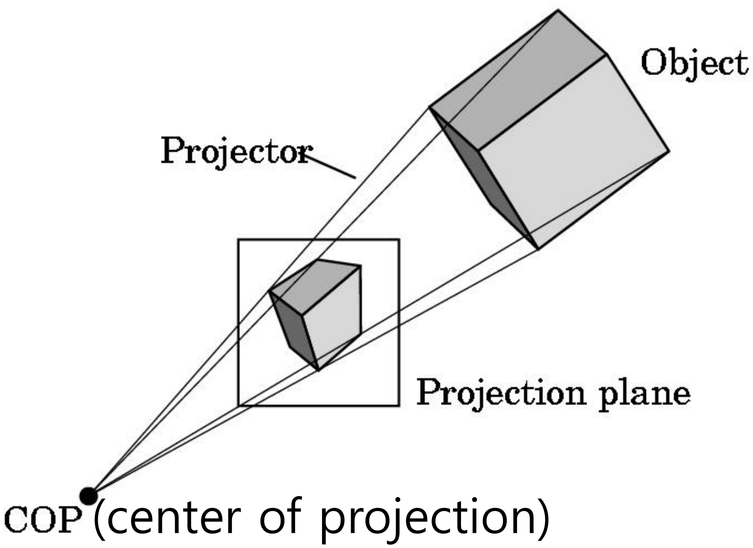
- View volume
- Projection Normalization
- Orthogonal projection

model-view, projection matrices in GLSL

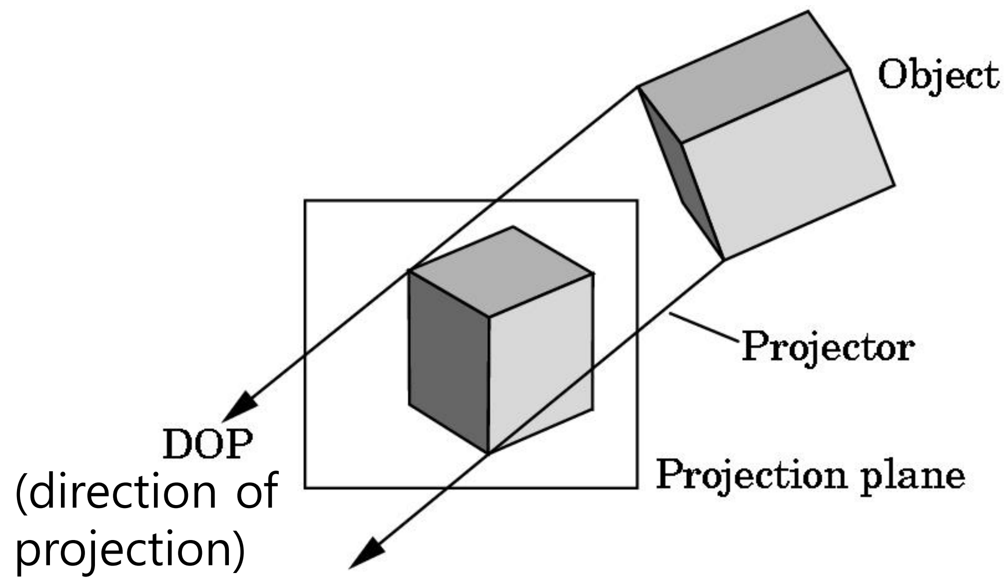
```
uniform mat4 mat_model;          // model matrix
uniform mat4 mat_view;           // view matrix
uniform mat4 mat_projection;   // orthogonal or perspective projection matrix
in vec4 aPosition;
void main() {
    gl_Position = mat_projection * mat_view * mat_model * aPosition;
}
```

Projection Matrix (1)

- Perspective projection
- Parallel projection



Projectors pass through COP

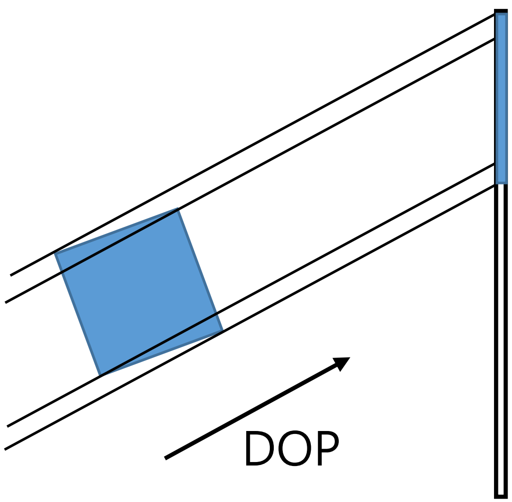


Projectors are parallel with DOP

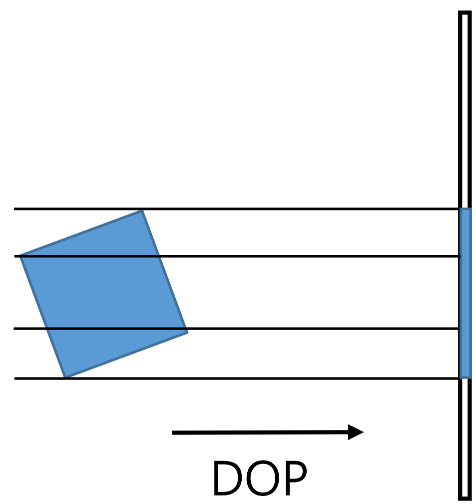
- Projection matrix is computed with the normalization of the view volume

Projection Matrix (2)

- Parallel projection
  - points are projected onto a view plane in a direction that is parallel to DOP (direction of projection)
  - Orthogonal projection
    - a case of parallel projection, where DOP is normal to the view plane

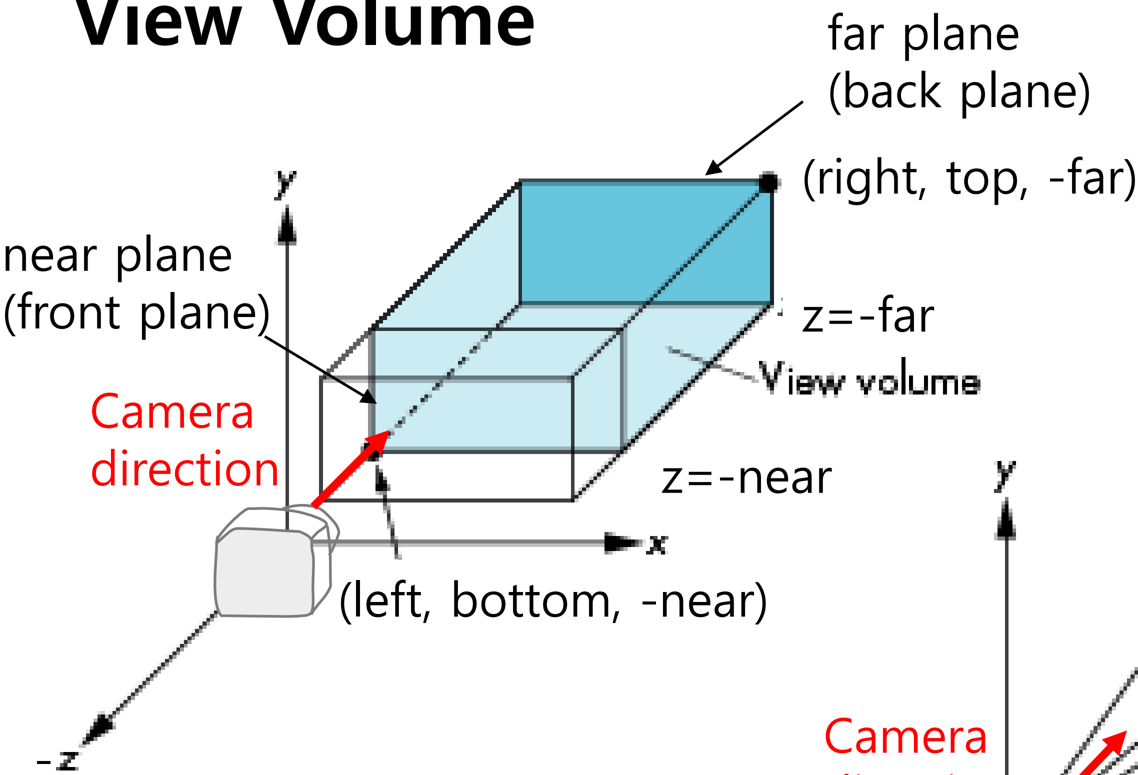


parallel projection

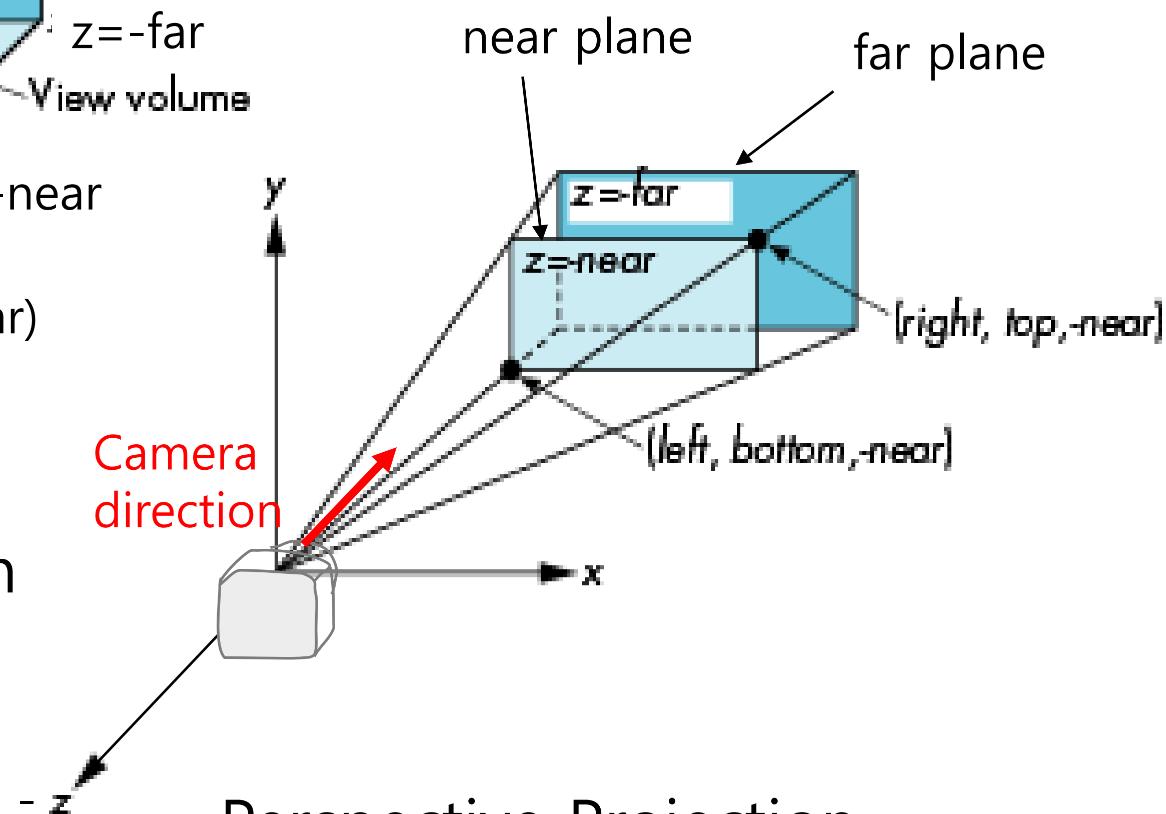


orthogonal projection

View Volume



Orthogonal Projection

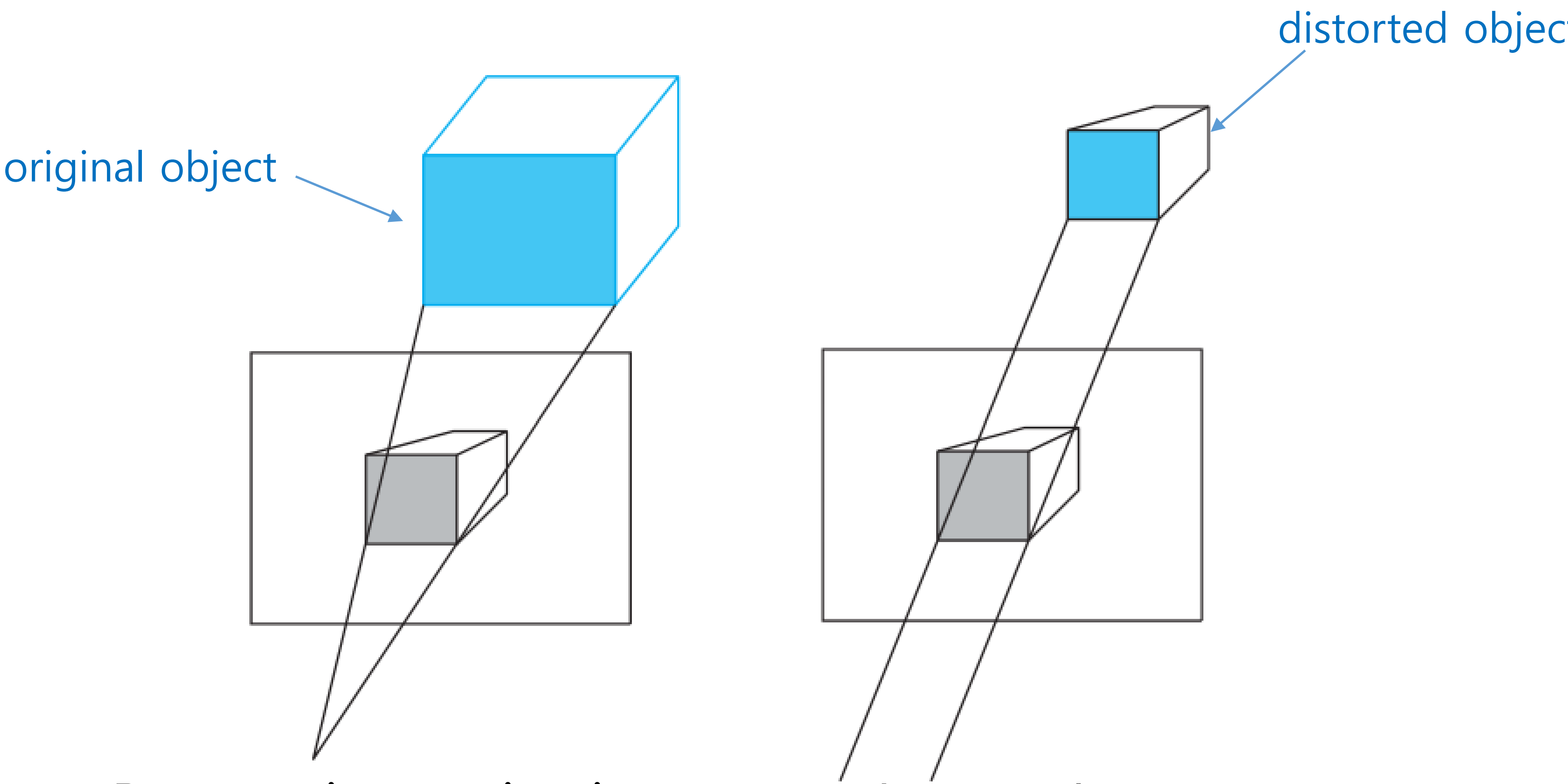


Perspective Projection

Projection Normalization (1)

- transform vertices in camera frame to fit inside the default view volume by using translation and scaling
- Convert all projections into orthogonal projection by distorting the objects
- Implement the orthogonal projection of distorted object (whose rendering result is identical to the desired projection of the original object)

Projection Normalization (2)



Perspective projection of original object

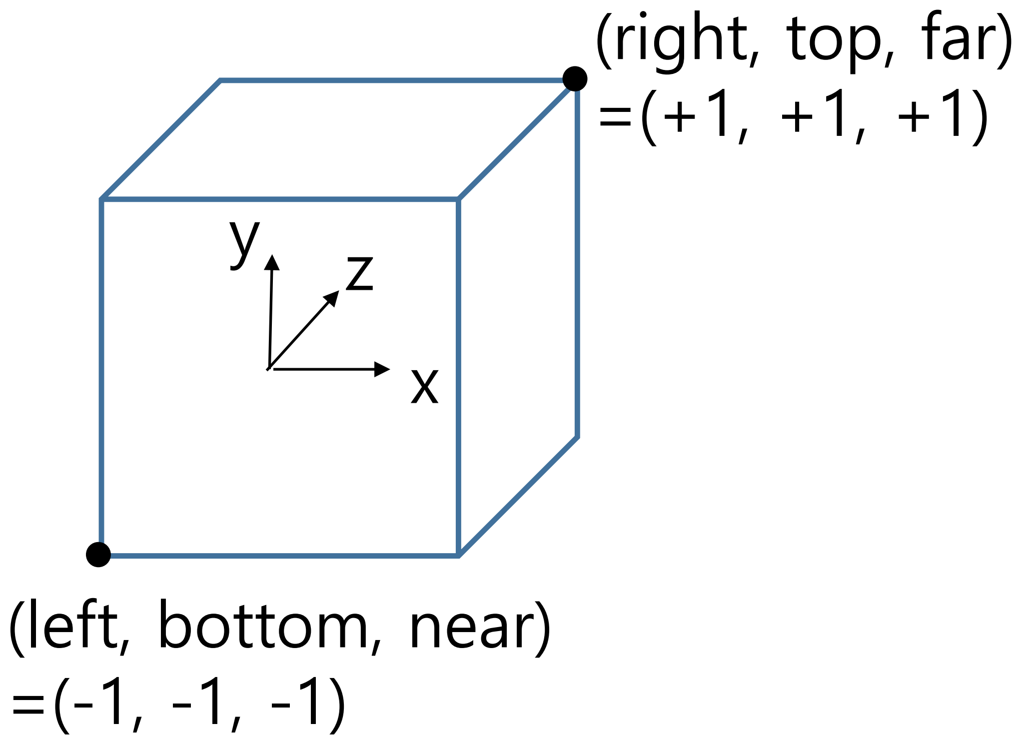
Orthogonal projection of distorted object

Projection Normalization (3)

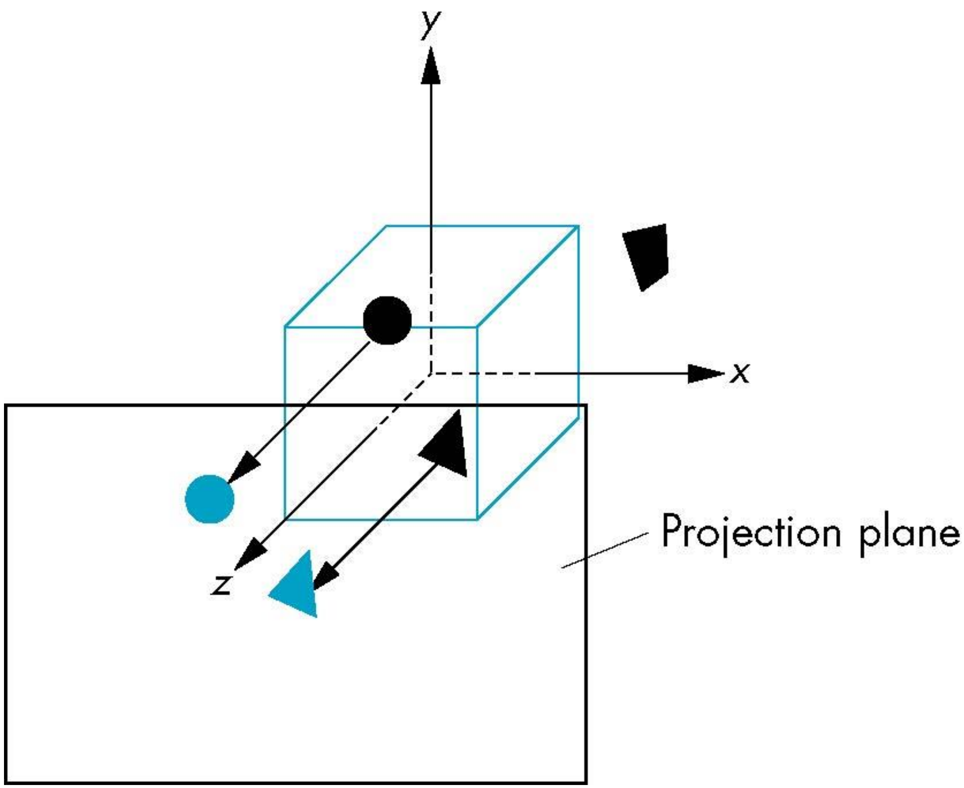
- Most graphics systems use normalization
- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping
- Delay final projection until end
  - Important for hidden-surface removal to retain depth information as long as possible
- Projection matrix: normalization + projection

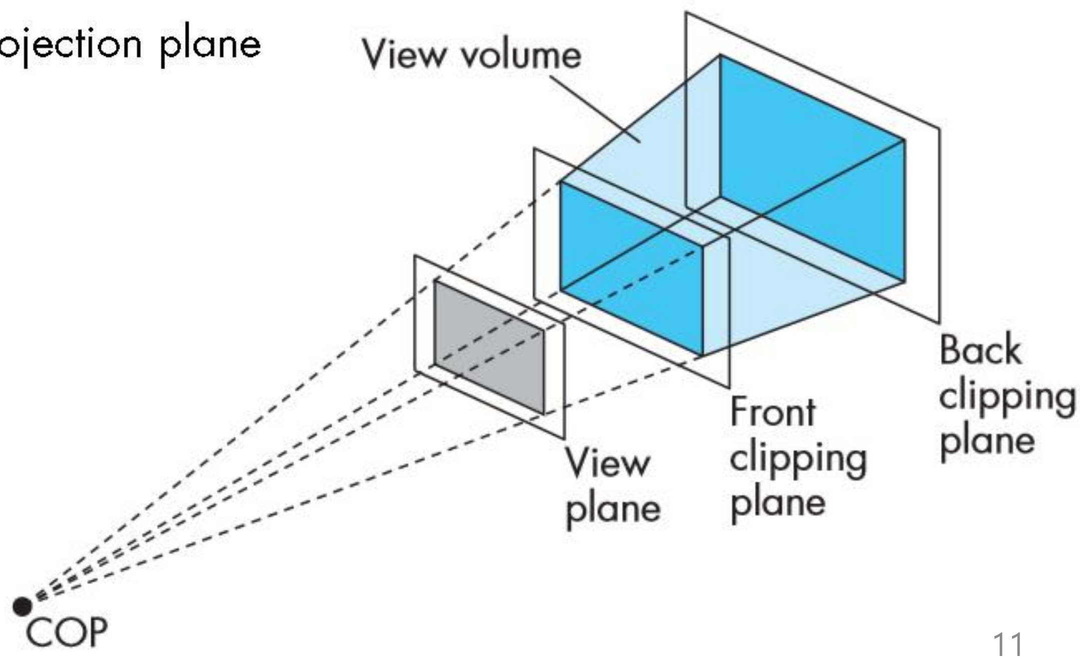
Canonical View Volume

- (= default view volume)
- Cube with planes x = ±1, y = ±1, z = ±1 and center (0, 0, 0)



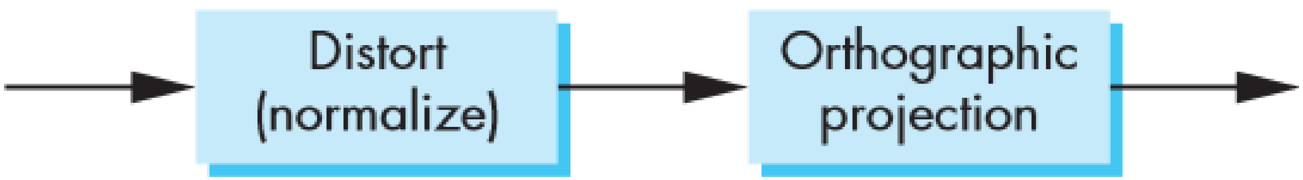
Clipping Process with Canonical View Volume





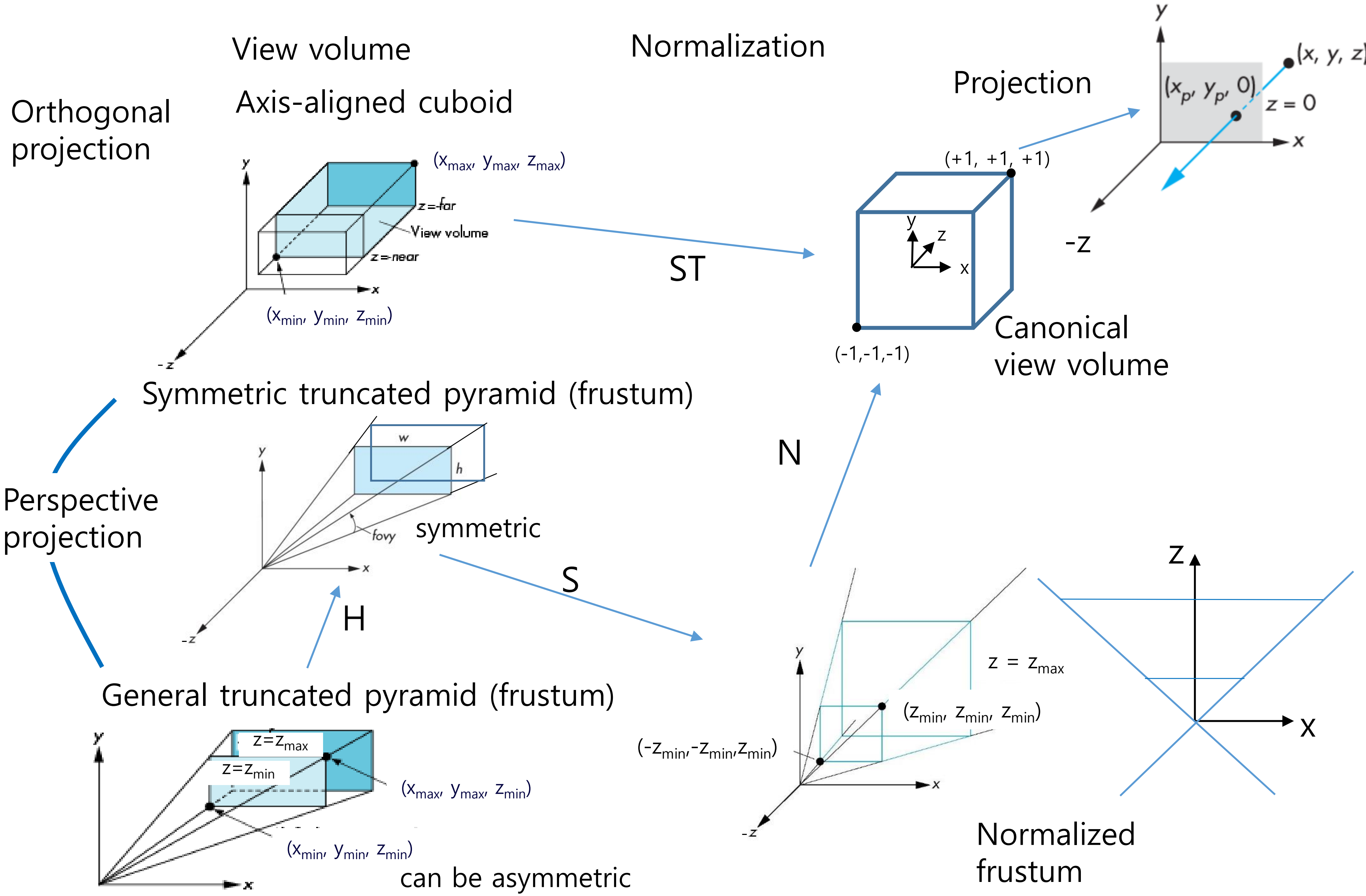
Projection Normalization Transformation

- In graphic pipeline
  - Normalization matrix for distortion
  - Simple orthogonal projection matrix



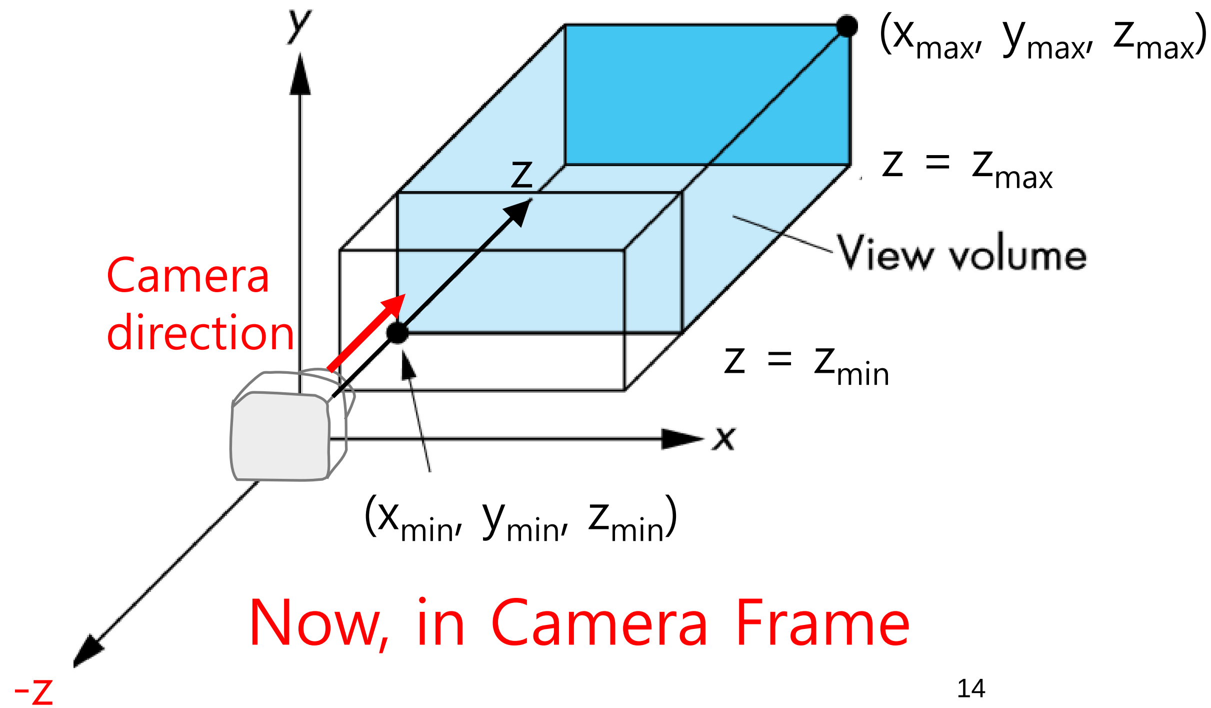
- By applying normalization matrix, we get canonical view volume
  - So, both perspective and orthogonal projections are supported by the same pipeline

Projection Matrix Construction Overview



OpenGL Orthogonal Viewing

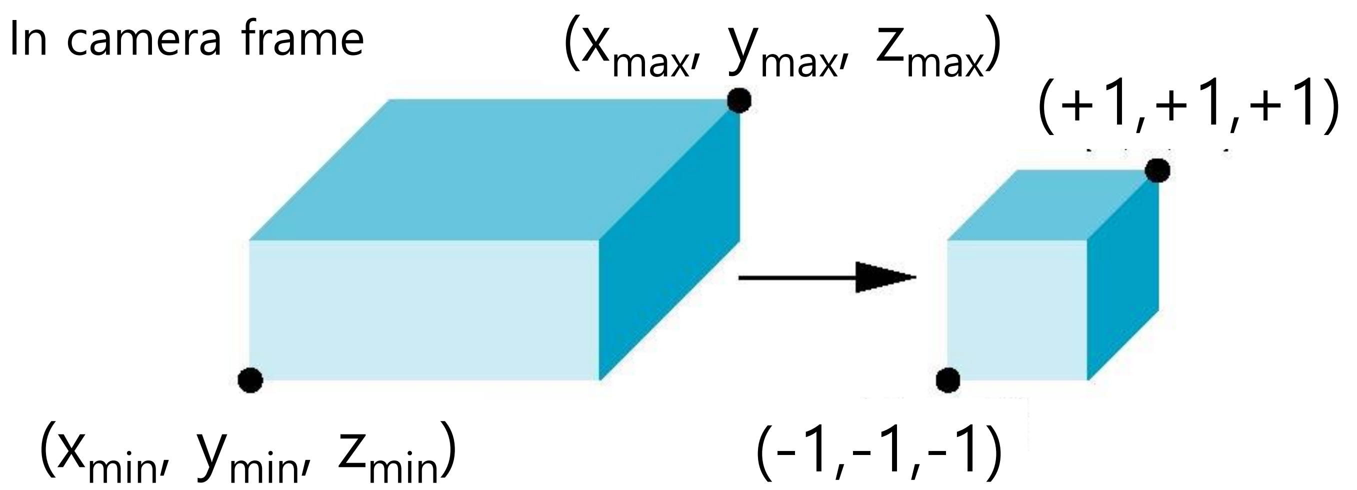
- Deprecated gl function
  - glOrtho(left, right, bottom, top, near, far)
- We will construct
  - myOrtho(m_{ortho}, x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})
  - m_{ortho}: orthogonal projection matrix



Now, in Camera Frame

Orthogonal Normalization

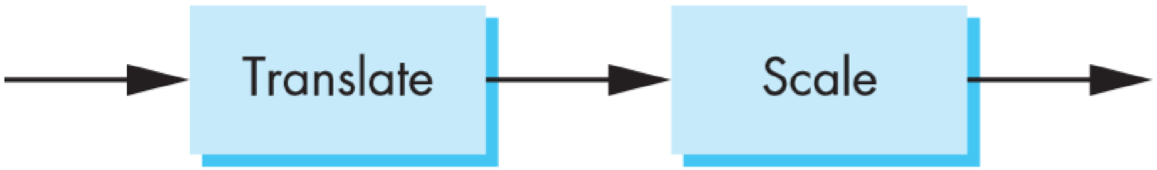
- $myOrtho(m_{ortho}, x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$
- normalization ⇒ find transformation matrix to convert specified view volume to default



- Center of the view volume
  - $((x_{max}+x_{min})/2, (y_{max}+y_{min})/2, (z_{max}+z_{min})/2))$
- Length of the view volume
  - $x_{max}-x_{min}$, $y_{max}-y_{min}$, $z_{max}-z_{min}$

Orthogonal Normalization Matrix (1)

- Affine transformation for normalization



- For orthogonal normalization
  - Translation matrix: Move center to origin
    $T(-(x_{max}+x_{min})/2, -(y_{max}+y_{min})/2, -(z_{max}+z_{min})/2))$
  - Scale matrix: Scale to have edges of length 2
    $S(2/(x_{max}-x_{min}), 2/(y_{max}-y_{min}), 2/(z_{max}-z_{min}))$

Orthogonal Normalization Matrix (2)

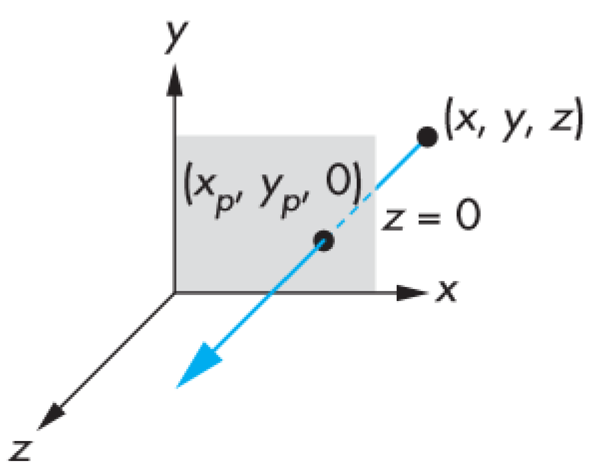
$$ST = \begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & 0 \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & 0 \\ 0 & 0 & \frac{2}{z_{max}-z_{min}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{x_{max}+x_{min}}{2} \\ 0 & 1 & 0 & -\frac{y_{max}+y_{min}}{2} \\ 0 & 0 & 1 & -\frac{z_{max}+z_{min}}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & -\frac{x_{max}+x_{min}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & -\frac{y_{max}+y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & \frac{2}{z_{max}-z_{min}} & -\frac{z_{max}+z_{min}}{z_{max}-z_{min}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Default Orthogonal Projection (1)

- OpenGL default setting
  - orthogonal projection with identity projection matrix
  - view volume: $x = \pm 1$, $y = \pm 1$, $z = \pm 1$
    - Canonical view volume
  - view plane is $z=0$
- For a point (x, y, z) within the canonical view volume, its projection is

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$



Default Orthogonal Projection (2)

- In homogeneous coordinates
  - $\mathbf{p}_p = M_{proj}\,\mathbf{p}$,

  where $\mathbf{p}_p = [x \;\; y \;\; 0 \;\; 1]^T$ and $\mathbf{p} = [x \;\; y \;\; z \;\; 1]^T$
- In practice, we can let $M_{proj} = \mathbf{I}$ and set the z term to zero later.
- After orthogonal normalization, M_{proj} is applied as a default projection

$$\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{}}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Orthogonal Projection Matrix

- Hence, general orthogonal projection in 4D is

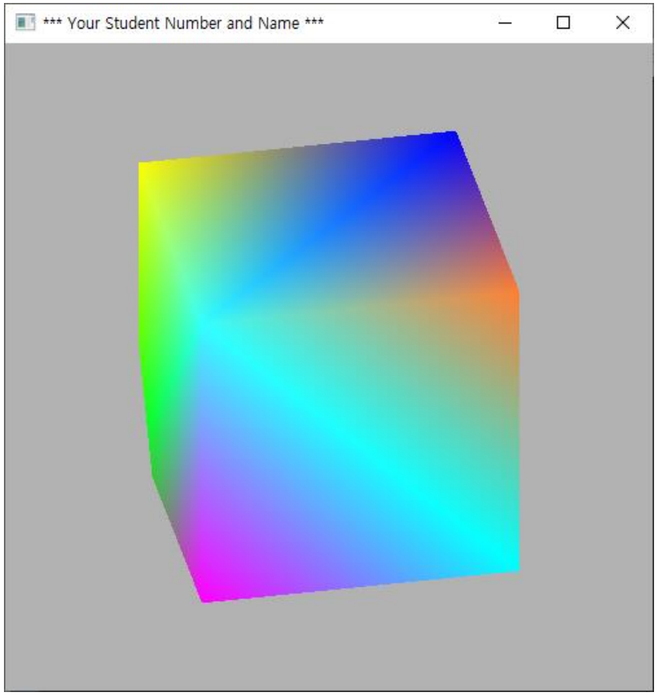
$\mathbf{p}_p = M_{proj}\, S\, T\, \mathbf{p}$

$\quad = M_{proj}\, M_{ortho}\, \mathbf{p}$

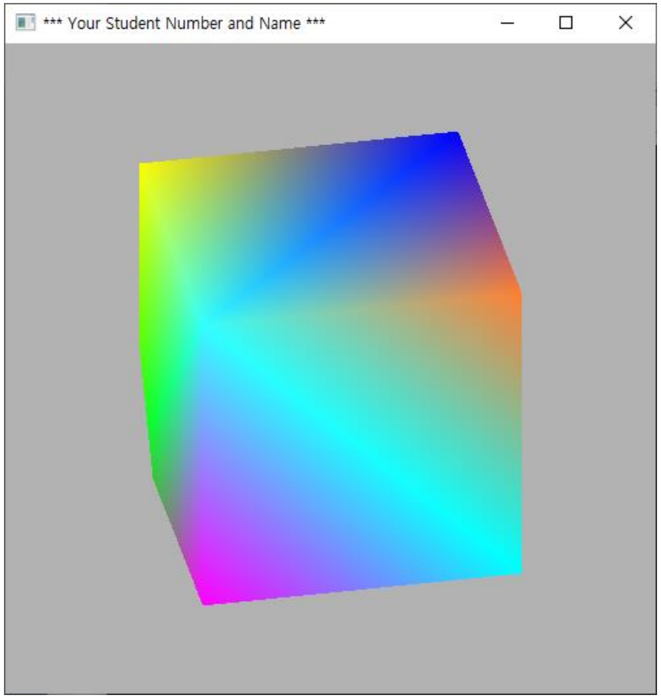
$$= M_{proj} \begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & -\frac{x_{max}+x_{min}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & -\frac{y_{max}+y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & \frac{2}{z_{max}-z_{min}} & -\frac{z_{max}+z_{min}}{z_{max}-z_{min}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}$$

Orthogonal Projection Example (1)

- A cube
  - Center: (0, 0, 0), Length of all edges: 1
- LookAt (eye, at, up) :float eye[4] = {0.5, 0.3, 0.1, 1.0}, at[4] = { 0.0, 0.0, 0.0, 1.0 }, up[4] = { 0.0, 1.0, 0.0, 0.0 };

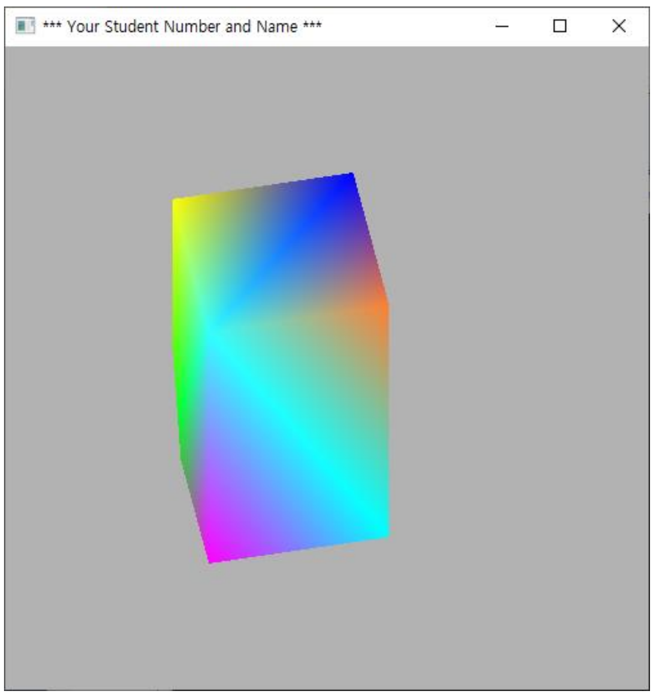


Default state
(M_{ortho} is not applied in vertex shader)

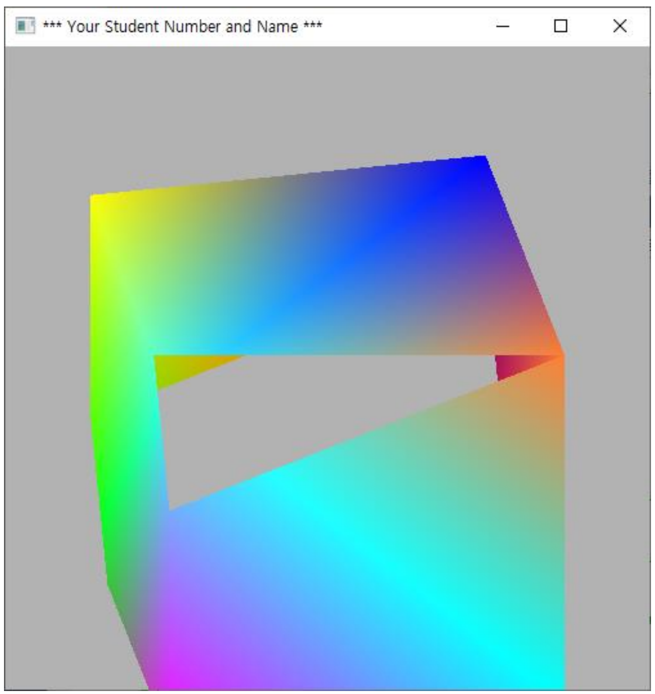


Ex1) myOrtho (M_{ortho}, -1.0, +1.0, -1.0, +1.0, -1.0, +1.0)

Orthogonal Projection Example (2)



Ex2) myOrtho (M_{ortho}, -1.5, +2.0, -1.2, +1.2, -1.0, +1.0)



Ex3) myOrtho (M_{ortho}, -0.8, +0.8, -0.6, +1.0, -0.0, +1.0)

HW#19 Orthogonal Projection

- No submission
- Render the cube as in the right image.
- Cube information is given in the Lecture Note board.
- Use LookAt (eye, at, up) with float eye[4] = {0.5, 0.3, 0.1, 1.0}, at[4] = { 0.0, 0.0, 0.0, 1.0 }, up[4] = { 0.0, 1.0, 0.0, 0.0 };
- Construct myOrtho function and use it as: myOrtho (M_{ortho}, -1.5, +2.0, -1.2, +1.2, -1.0, +1.0)