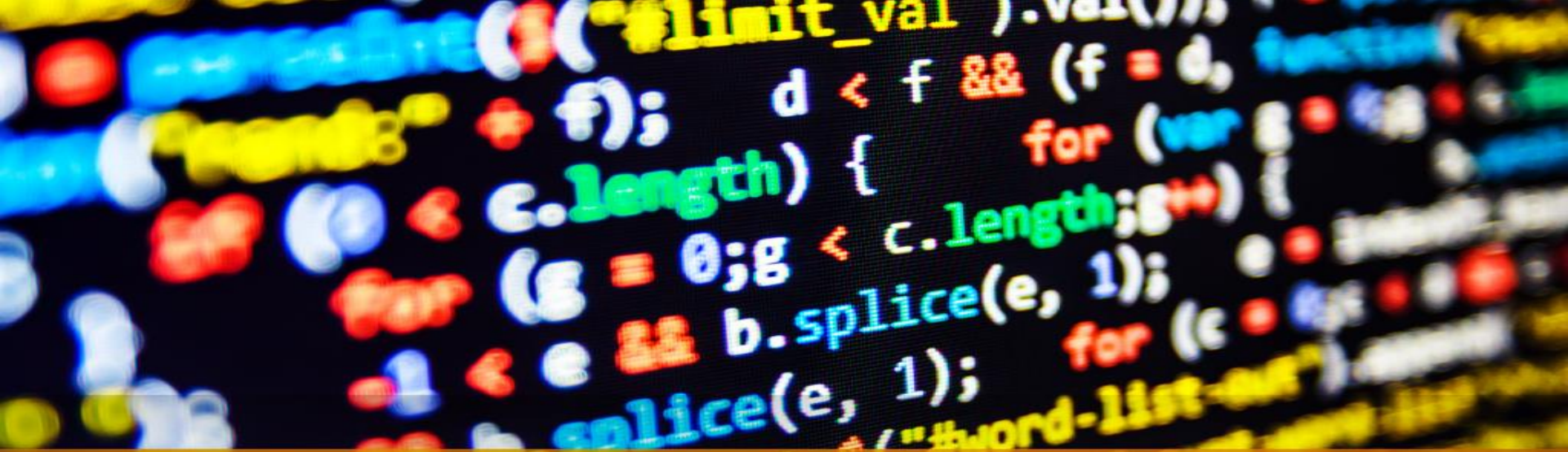


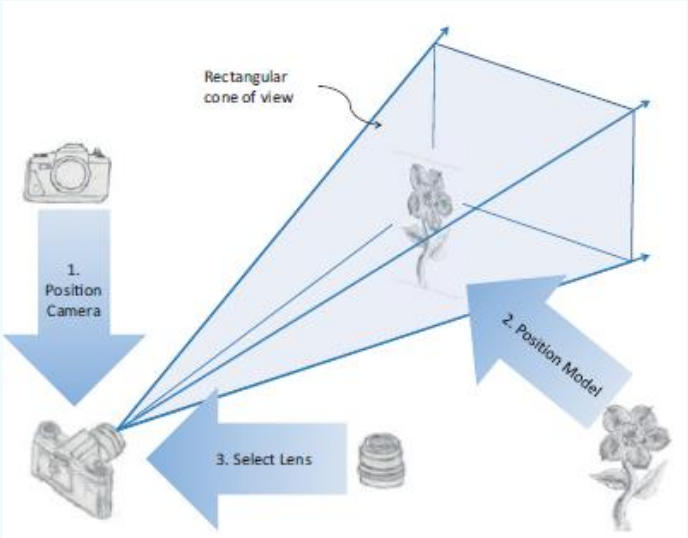
Lecture 06

Viewing & Camera

Prepared by Ban Kar Weng (William)

Understanding Viewing

What we typically do when taking a photograph with a camera.

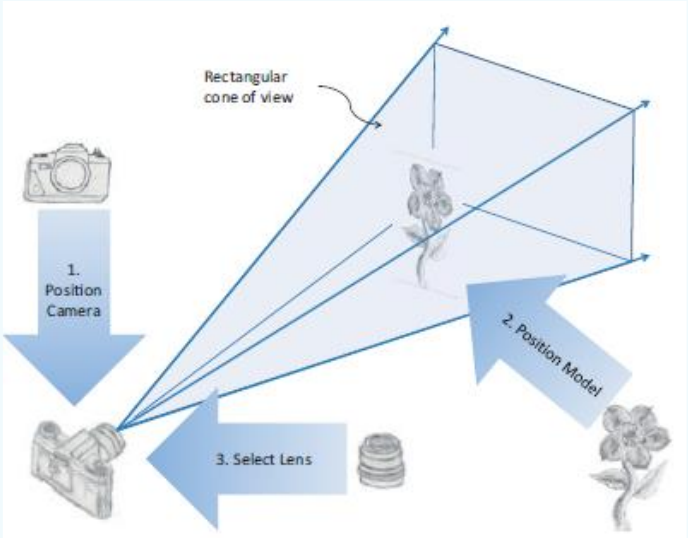


1. **Position the camera** in the location you want to shoot from and point the camera to the desired direction.
2. **Position the model** to be photographed into the desired location in the scene
3. **Select a camera lens** or adjust the zoom.
4. **Take the picture**.

Image taken from Figure 5.1, OpenGL Programming Guide: The Official Guide to Learning OpenGL 4.3.

Understanding Viewing

Viewing in computer graphics is analogous to taking a photograph with a camera.

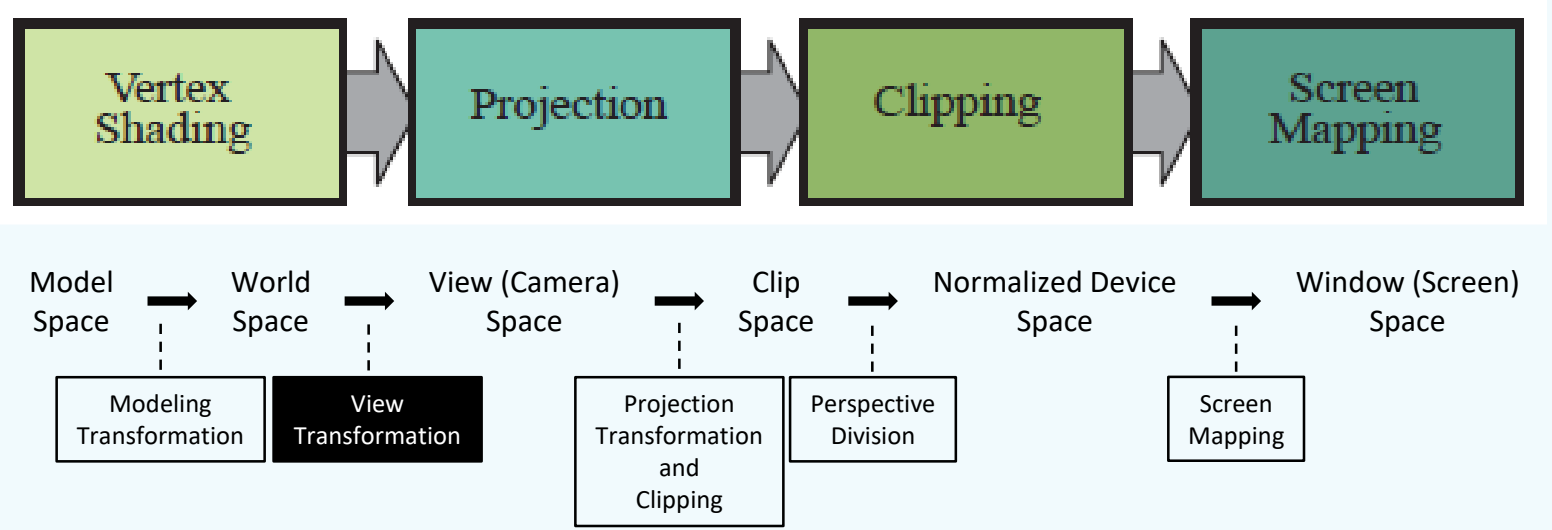


1. **Position the camera** → *Viewing Transformation.*
2. **Position the model** → *Modeling Transformation.*
3. **Select a camera lens** → *Projection Transformation*.
4. **Take the picture** → *Apply the transformations*.

Image taken from Figure 5.1, OpenGL Programming Guide: The Official Guide to Learning OpenGL 4.3.

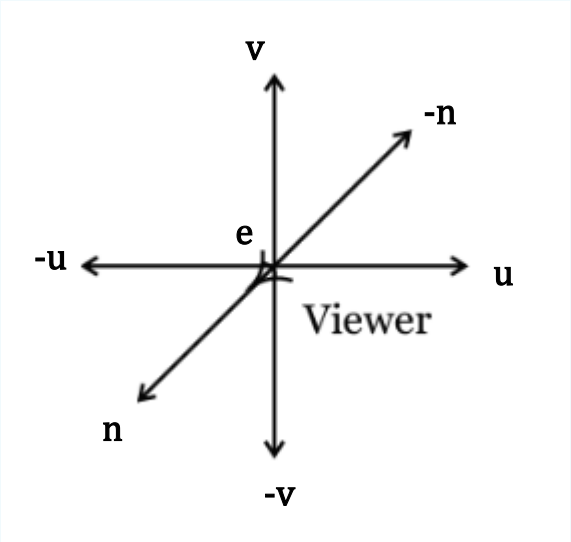
Viewing Transformation

Viewing Transformation



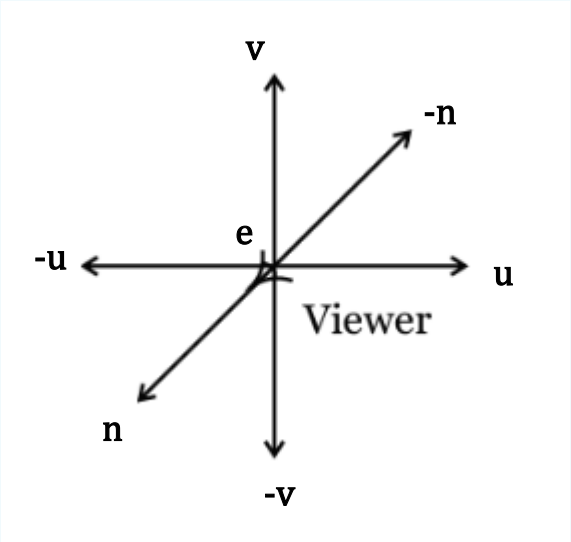
* We will look at viewing transformation and projection transformation in this lecture in the coming slides.

Viewing Transformation | **Introduction**



- Focus on how to view the world
- In OpenGL, the viewer is assumed to be located at the origin, looking at the negative z-axis.
- The viewing transformation applies to the world space.
- Therefore, the viewing transformation is a separate transformation that can be calculated independently (i.e. before any modelling transformation on objects).

Viewing Transformation | **Introduction**



- **<u>Goal</u>**: To determine the viewing transformation matrix $M_{w\to v}$ that transform any world coordinate to view coordinate, that is:

$$^{v}p = M_{w\to v}{}^{w}p$$

- This can be achieved in two steps:
  1. Find $M_{v\to w} = \begin{bmatrix} u & v & n & e \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  2. $M_{w\to v} = M_{v\to w}^{-1}$

Viewing Transformation | Step 1 - Find $\boldsymbol{M_{v \to w}}$

Step 1 : Find $\boldsymbol{M_{v \to w}} = \begin{bmatrix} \boldsymbol{u} & \boldsymbol{v} & \boldsymbol{n} & \boldsymbol{e} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$

The values of u, v, n, and e can be given manually, but there is a less laborious way, which computes u, v, and n given the following input:

Input:

$e \rightarrow$ the viewer's "eye" coordinate expressed in world coordinate

$c \rightarrow$ the centre point "looked" by the viewer.

$v_{up} \rightarrow$ the up vector that points roughly to the top of viewer.

Refererence: Interactive Computer Graphics – A Top Down Approach using WebGL (7th Edition), Mathematics for 3D Game Programming and Computer Graphics (3rd Edition)

Viewing Transformation | Step 1 - Find $\boldsymbol{M_{v\rightarrow w}}$

Step 1 : Find $\boldsymbol{M_{v\rightarrow w}} = \begin{bmatrix} \boldsymbol{u} & \boldsymbol{v} & \boldsymbol{n} & \boldsymbol{e} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$

Algorithm:

Input: e, c, v_{up} |Output: u, v, n

1. $n = \frac{e-c}{\|e-c\|}$
2. $u = \frac{v_{up}\times n}{\|v_{up}\times n\|}$
3. $v = \frac{n\times u}{\|n\times u\|}$

Viewing Transformation | Step 1 - Find $\boldsymbol{M_{v\rightarrow w}}$

Step 1 : Find $\boldsymbol{M_{v\rightarrow w}} = \begin{bmatrix} \boldsymbol{u} & \boldsymbol{v} & \boldsymbol{n} & \boldsymbol{e} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$

Example:

$$e = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, c = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, v_{up} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$n = \frac{e - c}{\|e - c\|} = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}$$

$$u = \frac{v_{up} \times n}{\|v_{up} \times n\|} = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix}$$

$$v = \frac{n \times u}{\|n \times u\|} = \begin{bmatrix} -1/\sqrt{6} \\ 1/\sqrt{6} \\ -1/\sqrt{6} \end{bmatrix}$$

Viewing Transformation | Step 2 - Find $\boldsymbol{M_{w\to v}}$

Step 2 : Find $\boldsymbol{M_{w\to v} = M_{v\to w}^{-1}}$

Let $M_{v\to w} = \begin{bmatrix} u & v & n & e \\ 0 & 0 & 0 & 1 \end{bmatrix}$ be rewritten as $\begin{bmatrix} F & e \\ \vec{0}^T & 1 \end{bmatrix}$ where $F = \begin{bmatrix} u & v & n \end{bmatrix}$

Since $M_{w\to v} = M_{v\to w}^{-1}$

$$M_{v\to w} M_{w\to v} = I_4$$

$$\begin{bmatrix} F & e \\ \vec{0}^T & 1 \end{bmatrix} M_{w\to v} = \begin{bmatrix} I_3 & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix}$$

$$\boldsymbol{M_{w\to v} = \begin{bmatrix} F^{-1} & -F^{-1}e \\ \vec{0}^T & 1 \end{bmatrix}}$$

Viewing Transformation | Step 2 - Find $\boldsymbol{M_{w\to v}}$

Step 2 : Find $\boldsymbol{M_{w\to v} = M_{v\to w}^{-1}}$

$$M_{w\to v} = \begin{bmatrix} F^{-1} & -F^{-1}e \\ \vec{0}^T & 1 \end{bmatrix}$$

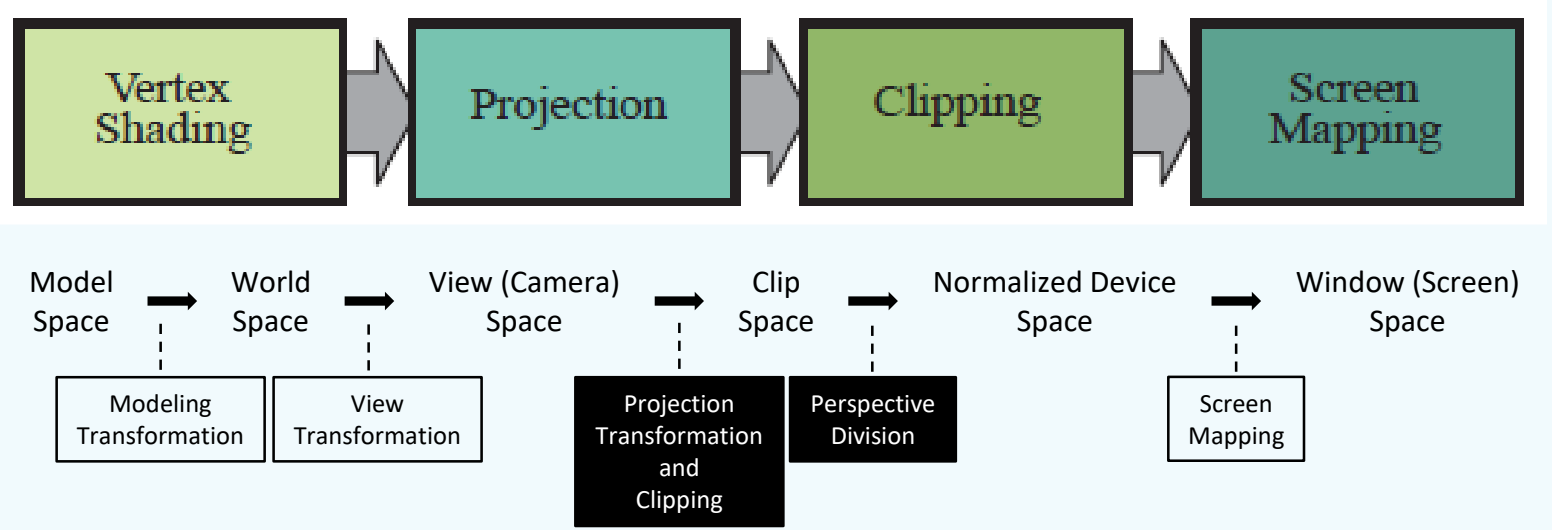
What is F^{-1}?

Since F is an orthogonal matrix with unit column vectors, $\boldsymbol{F^{-1} = F^T}$!!!
Therefore:

$$M_{w\to v} = \begin{bmatrix} F^T & -F^T e \\ \vec{0}^T & 1 \end{bmatrix} = \begin{bmatrix} u^T & -u^T e \\ v^T & -v^T e \\ n^T & -n^T e \\ 0 & 1 \end{bmatrix}$$

Projection

Projection



* We will look at projection transformation and perspective division in this lecture in the coming slides.

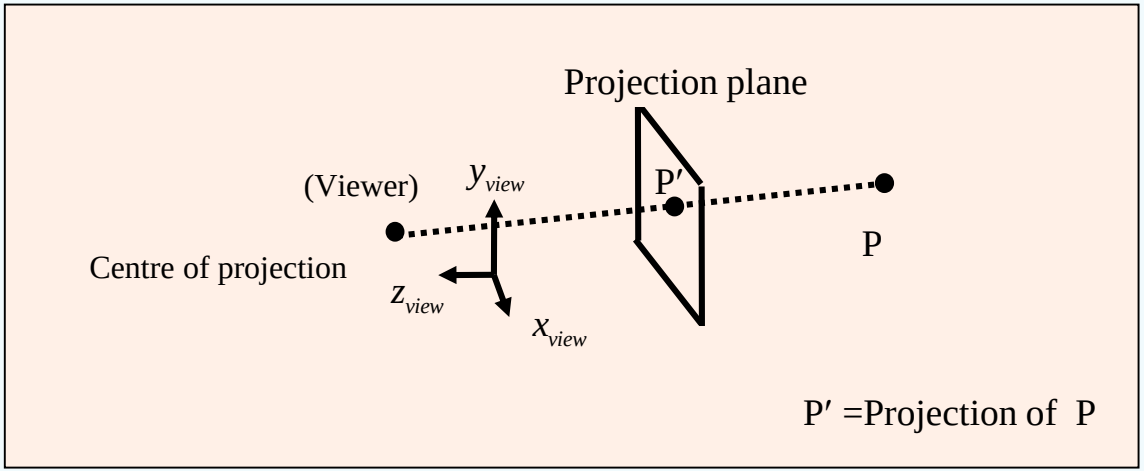
Projection | **Introduction**

The projection stage does the following:

1. Determines where on a 2D screen each vertex in the 3D scene to be drawn.
2. Defines **view volume** for clipping and culling.
3. Transforms view volume into **clip space**, which **after perspective divide**, results in **normalized device space**, which has a shape of a **unit cube**.

Projection | **Introduction**

1. Determines where on a 2D screen each vertex in the 3D scene to be drawn.



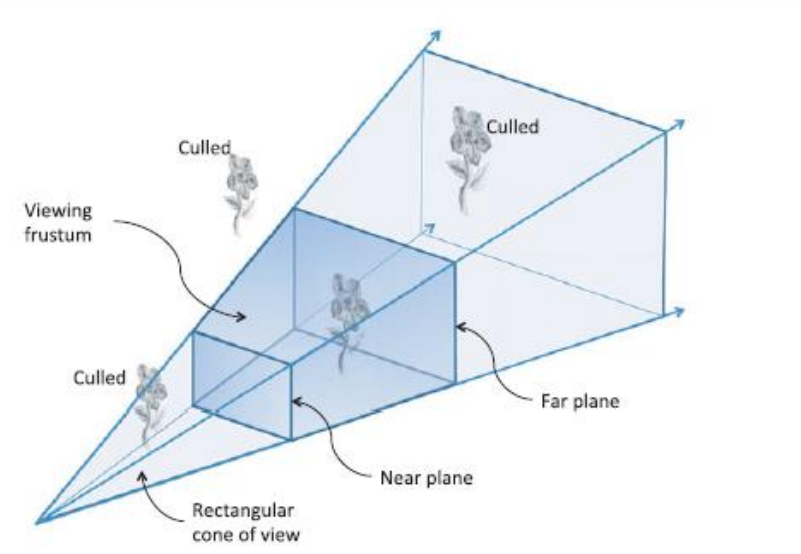
Projectors:
The lines through the centre of projection from the 3D vertex.

Projection plane:
The plane on which the projectors intersect. Typically, it is defined on a chosen z value.

Images taken from past materials of Computer Graphics Fundamentals Course.

Projection | Introduction | **View Volume**

2. Defines view volume for culling and clipping.



View Volume

A region of interest in 3D scene which will be used to generate display on the screen.

Culling

Discard primitives that completely fall outside the view volume.

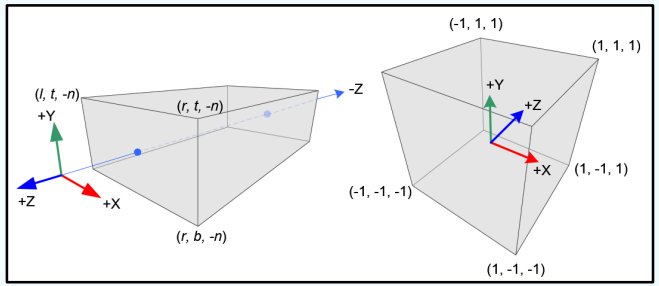
Clipping

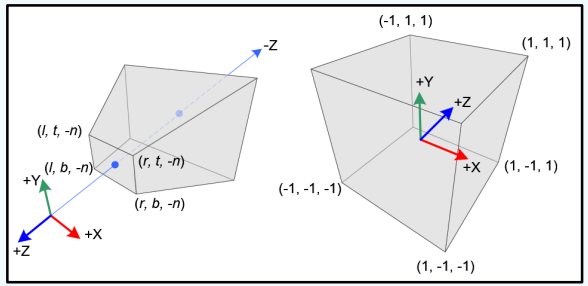
Form new geometry for primitives that partially fall outside the view volume.

Image taken from Figure 5.1, OpenGL Programming Guide: The Official Guide to Learning OpenGL 4.3.

Projection | Introduction | **Projection Transformation**

3. Define **projection transformation** that transforms view volume into clip space, which after **perspective divide**, results in normalized device space, which has a shape of a unit cube.





View (Camera) Space → Clip Space → Normalized Device Space

Projection Transformation

Perspective Division

All images are taken online. To get the link, right click the image and select “Edit Hyperlink” in the context menu.

Projection | Introduction | **Projection Transformation**

- Projection transformation $M_{v \to c}$ converts view coordinates to clip coordinates.

$$^{c}p = M_{v \to c}{}^{v}p$$

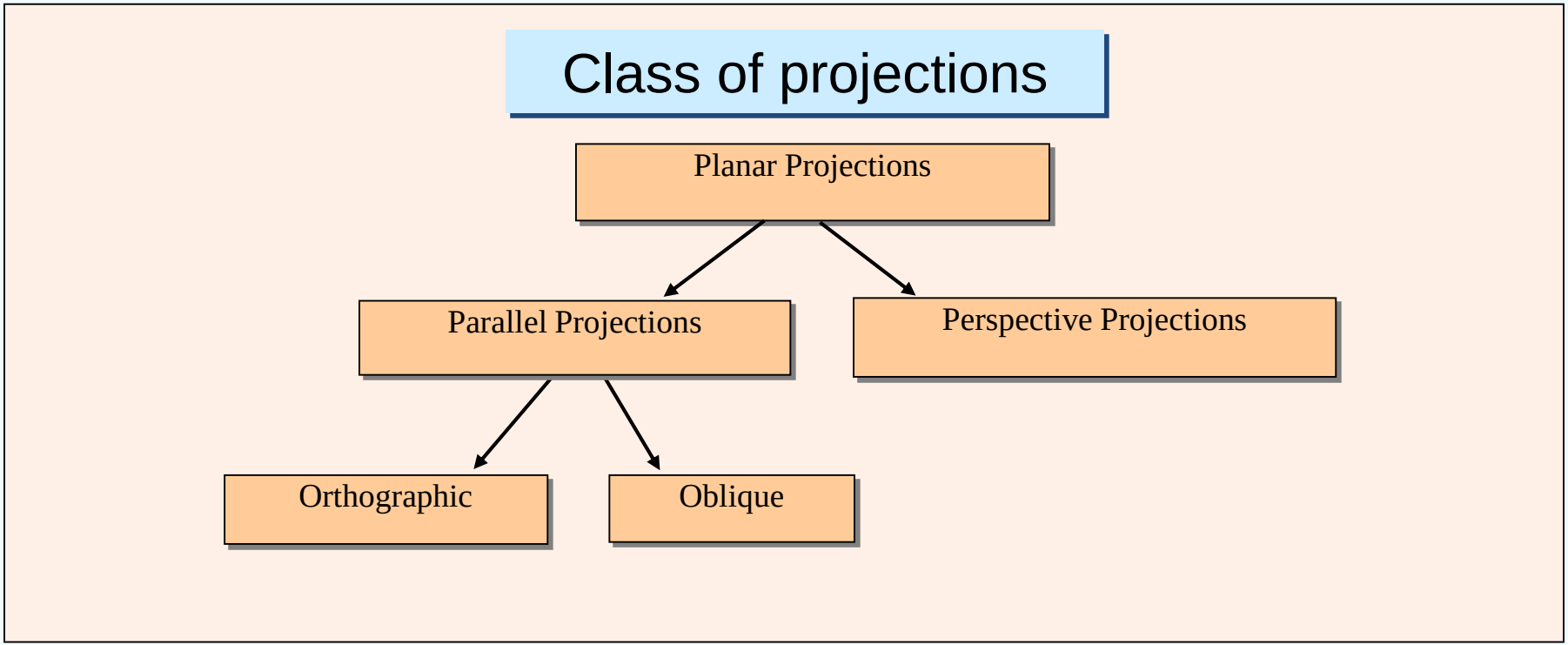
- $M_{v \to c}$ must be formulated such that for ^{c}p within the view volume, performing perspective divide results in points that is within the normalized device space.

Projection | Introduction | **Perspective Division**

- Clip coordinate ${}^{c}p$ is a homogeneous coordinate with $w \neq 1$
- **Perspective divide** converts ${}^{c}p$ to normalized device coordinate by dividing ${}^{c}x, {}^{c}y$, and ${}^{c}z$ by w.

$$ {}^{nd}p = \begin{bmatrix} {}^{nd}x \\ {}^{nd}y \\ {}^{nd}z \end{bmatrix} = \begin{bmatrix} {}^{c}x/w \\ {}^{c}y/w \\ {}^{c}z/w \end{bmatrix} $$

Projections | **Class of Projections**



Projections | Class of Projections

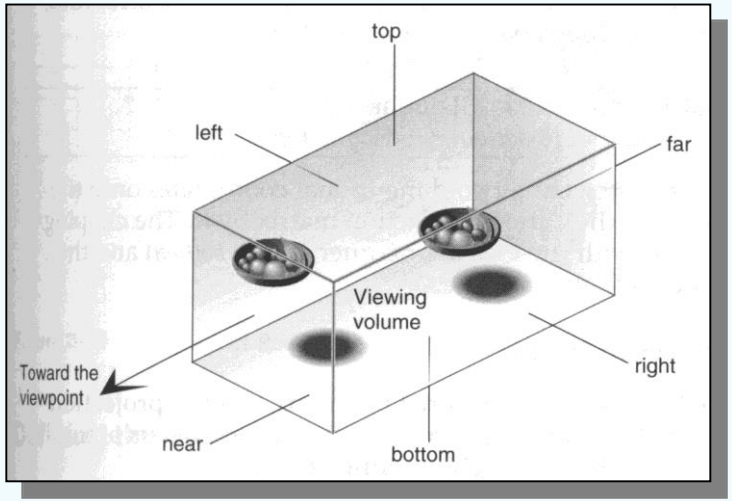
Parallel Projections:

- The centre of projection is at infinity.
- The projectors are parallel to each other.

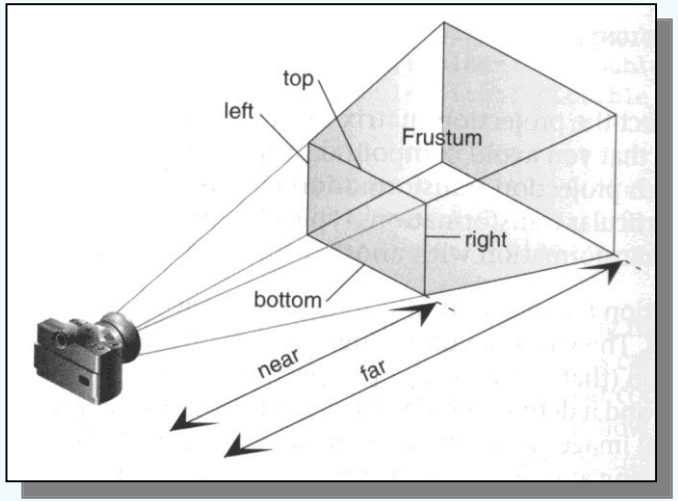
Perspective Projections:

- The centre of projection is a finite point.
- The projectors intersect at the centre of projection.

Projections | Class of Projections | **View Volume**



Orthographic Projection



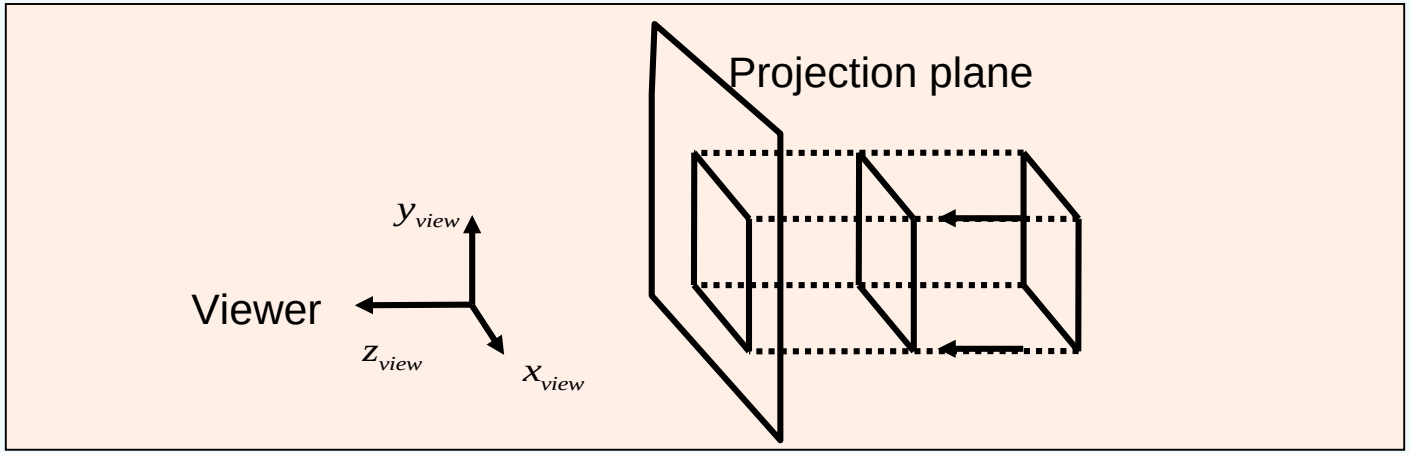
Perspective Projection

Images taken from past materials of Computer Graphics Fundamentals Course.

Parallel Projections | **Orthographic Projection**

The lines of projection are parallel, and at the same time orthogonal to the projection plane (i.e. the projectors are parallel to the view direction)

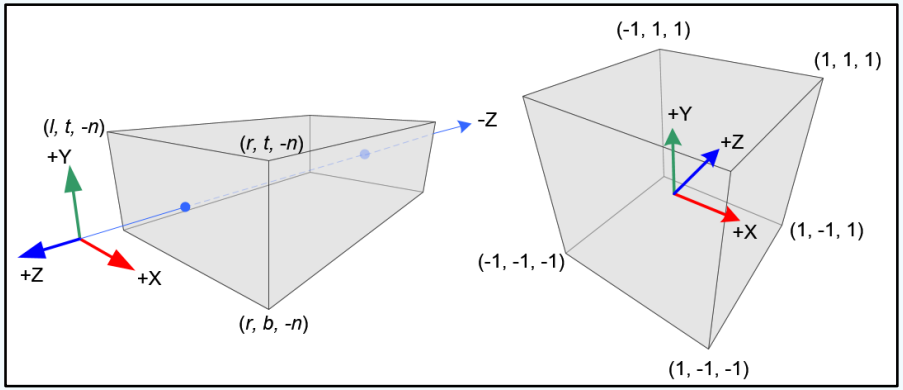
Since the projectors are parallel to z-axis, $(x, y, z) \rightarrow (x, y)$



Images taken from past materials of Computer Graphics Fundamentals Course.

Orthographic Projections

Let $M_{v \to c} = M_{ortho}$

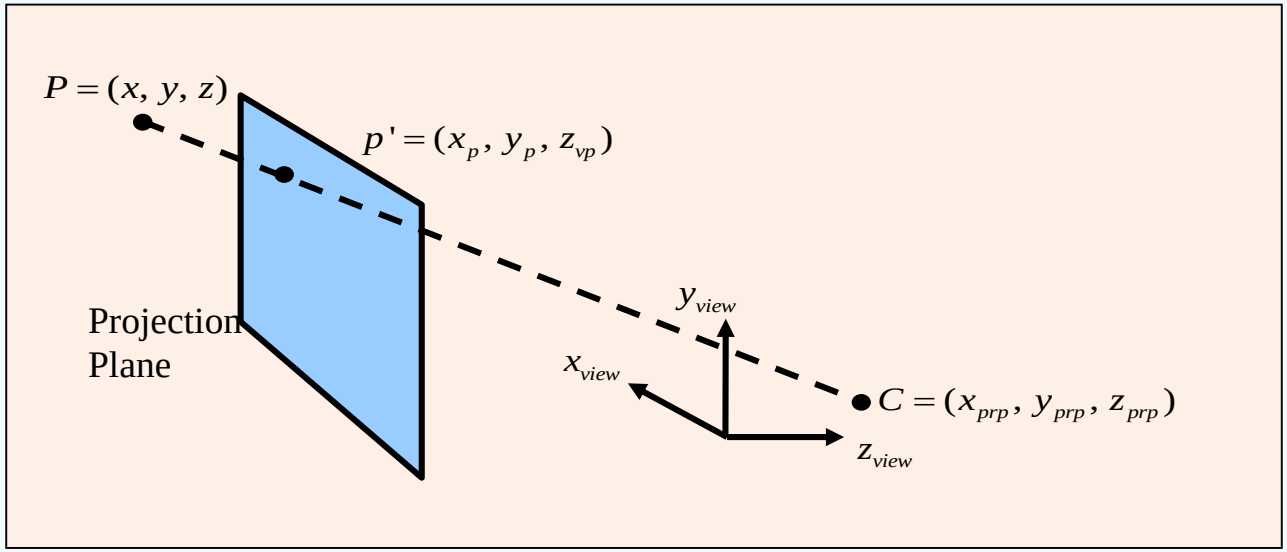


$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Check this out for the full proof of M_{ortho}

Perspective Projections

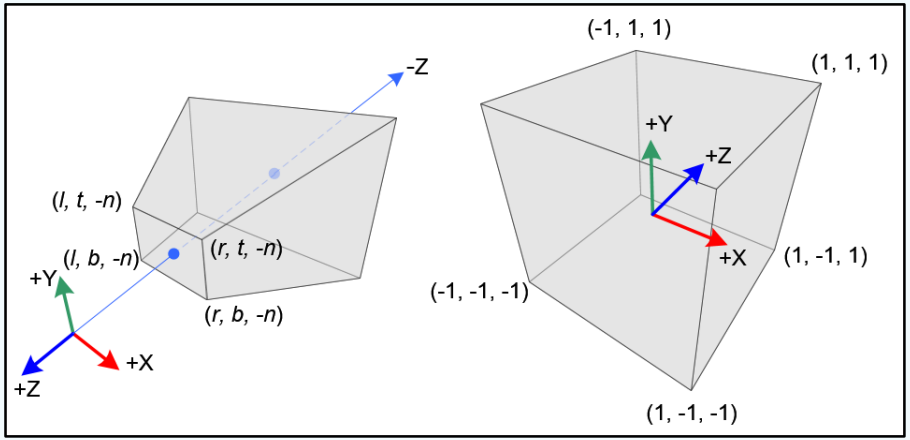
Perspective projections are characterized by the existence of a finite projection reference point (or center of projection).



Images taken from past materials of Computer Graphics Fundamentals Course.

Perspective Projections

Let $M_{v \to c} = M_{persp}$

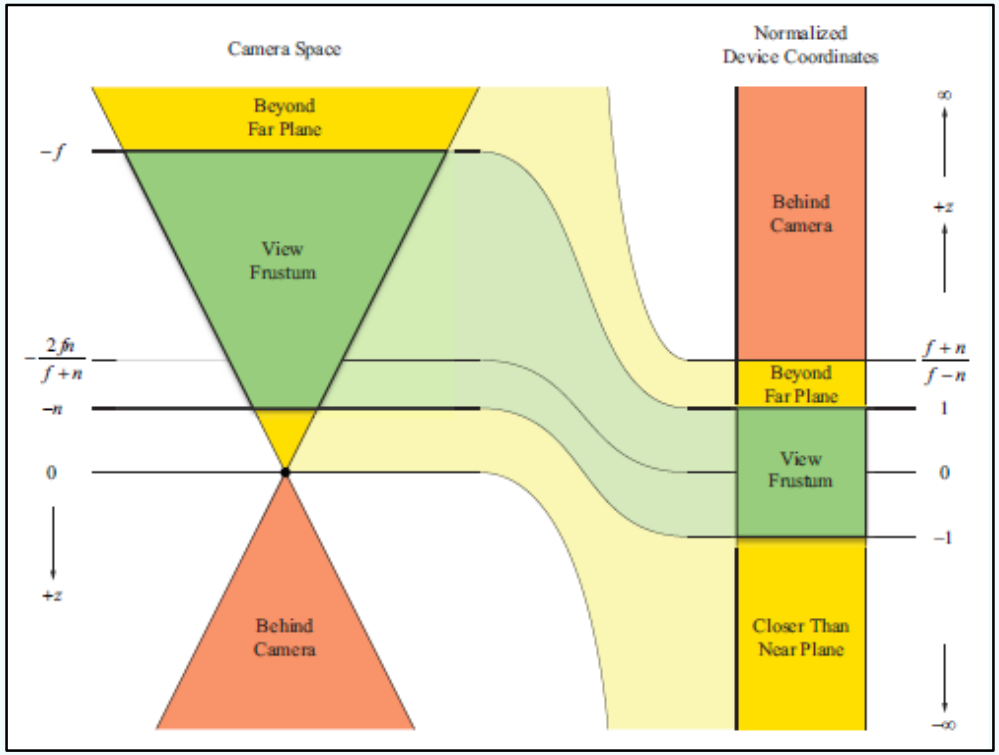


$$M_{persp} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Check this out for the proof of M_{persp}

All images are taken online. To get the link, right click the image and select “Edit Hyperlink” in the context menu.

Perspective Projection



Clipping

Clipping | **Introduction**

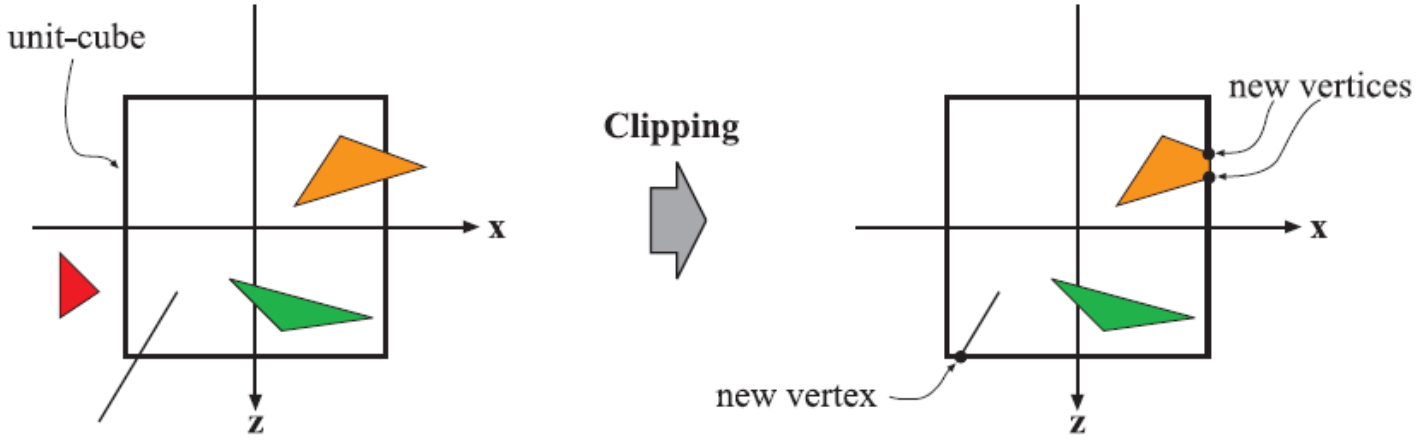


Figure 2.6. After the projection transform, only the primitives inside the unit cube (which correspond to primitives inside the view frustum) are needed for continued processing. Therefore, the primitives outside the unit cube are discarded, and primitives fully inside are kept. Primitives intersecting with the unit cube are clipped against the unit cube, and thus new vertices are generated and old ones are discarded.

Image taken from Real-time Rendering (4th Edition), Figure2.6

Clipping | Introduction

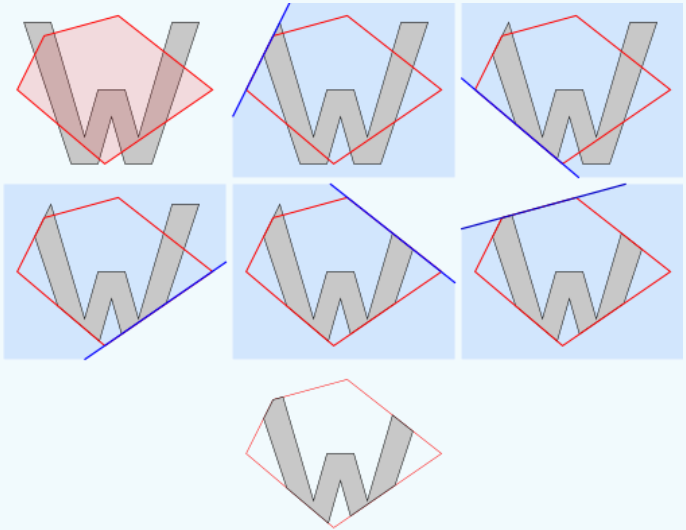
Line Clipping Algorithms

- Cohen-Sutherland
- Liang-Barsky
- *many more ……*

Polygon Clipping Algorithms

- Sutherland-Hogman
- Weiler-Atherton
- *many more ……*

Sutherland-Hogman Algorithm

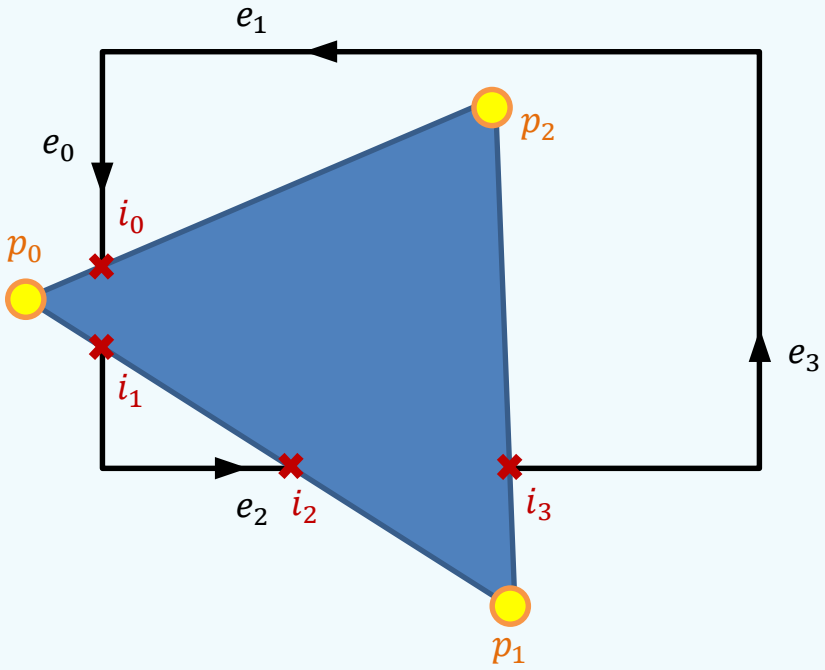


- Clips one **source polygon** against a second, **convex clip polygon**.
- Works by incrementally clipping the source against the line through each edge of the boundary polygon.

- **Input**:
  - A list of vertices in the source polygon.
  - A list of edges in a clip polygon.
- **Output**:
  - A list of vertices of the clipped source polygon.

All images are taken online. To get the link, right click the image and select “Edit Hyperlink” in the context menu.

Sutherland-Hogman Algorithm



Example:

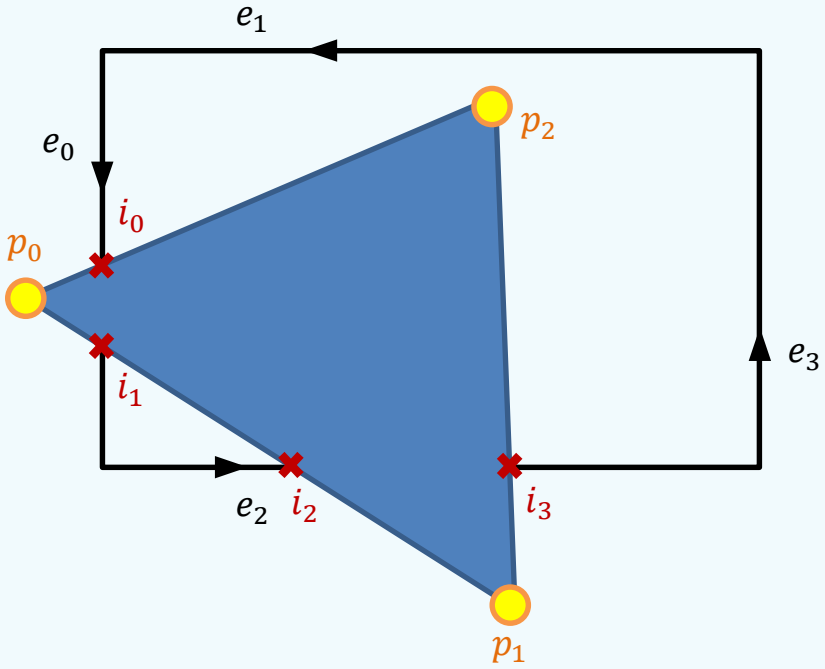
Input:

- $P_{in} = \{p_0, p_1, p_2\}$
- $E = \{e_0, \dots, e_3\}$

Expected Output:

- $P_{out} = \{i_0, i_1, i_2, i_3, p_2\}$

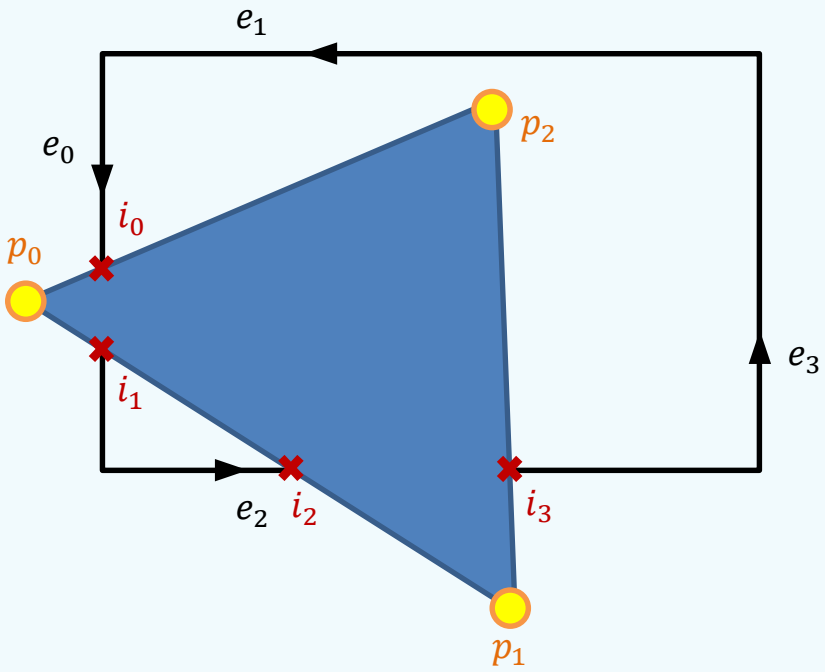
Sutherland-Hogman Algorithm




```
P_out = P_in;
for (Edge e in E) do
    P_temp = P_out;
    P_out.clear();
    for (int i = 0; i < P_temp.count; i += 1) do
        p_cur = P_temp[i];
        p_prev = P_temp[(i - 1) % inputList.count];
        i = ComputeIntersection(p_prev, p_cur, e)
        if (p_cur inside e) then
            if (p_prev not inside e) then
                P_out.add(i);
            end if
            P_out.add(p_cur);
        else if (p_prev inside e) then
            P_out.add(i);
        end if
    done
done
```

Algorithm pseudocode adapted from this page.

Sutherland-Hogman Algorithm



After processing e_0

- $P_{out} = \{i_0, i_1, p_1, p_2\}$

After processing e_1

- $P_{out} = \{i_0, i_1, p_1, p_2\}$

After processing e_3

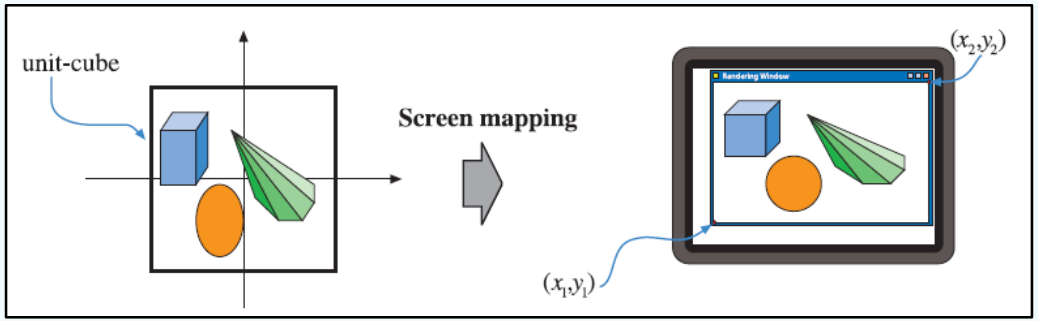
- $P_{out} = \{i_0, i_1, i_2, i_3, p_2\}$

After processing e_3

- $P_{out} = \{i_0, i_1, i_2, i_3, p_2\}$

Screen Mapping

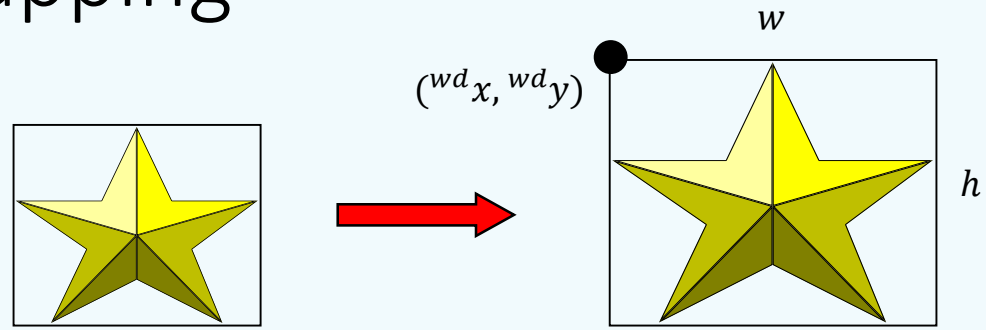
Screen Mapping



- **Goal**: Convert normalized device coordinates ${}^{nd}p$ to windows coordinate ${}^{wd}p$.
- More commonly known as **viewport transformation**.
- **Viewport** = A pixel rectangle region in the window into which the final image is mapped.
- x and y components are scaled and translated, z component is mapped to [0, 1] to be stored in depth buffer.

Image taken from Real-time Rendering (4th Edition), Figure 2.7, (page 20)

Screen Mapping



Let $^{wd}p = (^{wd}x, ^{wd}y, ^{wd}z)$, $^{nd}p = (^{nd}x, ^{nd}y, ^{nd}z)$

$$\frac{^{wd}x - x_0}{w} = \frac{^{nd}x - (-1)}{1 - (-1)}$$

$$\boldsymbol{^{wd}x = (^{nd}x + 1)\frac{w}{2} + x_0}$$

$$\frac{^{wd}y - y_0}{h} = \frac{^{nd}y - (-1)}{1 - (-1)}$$

$$\boldsymbol{^{wd}y = (^{nd}y + 1)\frac{h}{2} + y_0}$$

$$\boldsymbol{^{wd}z = \frac{^{nd}z + 1}{2}}$$

GLM

[Viewing Transformation & Projection]

GLM | Viewing Transformation

```
#include <glm/gtc/matrix_transform.hpp>

glm::vec3 camera_position(3.0f, 3.0f, 3.0f);
glm::vec3 camera_target(0.0f, 0.0f, 3.0f);
glm::vec3 up_vector(0.0f, 1.0f, 0.0f);
glm::mat4 view_transform =
glm::lookAt(camera_position, camera_target,
up_vector);
```

GLM | Orthographic Projection

```
#include <glm/gtc/matrix_transform.hpp>

float left = -3.0f, right = 3.0f;
float bottom = -3.0f, top = 3.0f;
float near = 1.0f, far = 10.0f;
glm::mat4 ortho_transform = glm::ortho(left, right,
bottom, top, near, far);
```

GLM | Perspective Projection | **glm::frustum**

```
#include <glm/gtc/matrix_transform.hpp>

float left = -3.0f, right = 3.0f;
float bottom = -3.0f, top = 3.0f;
float near = 1.0f, far = 10.0f;
glm::mat3 frustum_transform = glm::frustum(left,
right, bottom, top, near, far);
```

GLM | Perspective Projection | **glm::perspective**

```
#include <glm/gtc/matrix_transform.hpp>

float fovy = 45.0f;
float aspect_ratio = 1.0f;
float near = 1.0f, far = 10.0f;
glm::mat3 projective_transform =
glm::perspective(fovy, aspect_ratio, near, far);
```

Q & A

Acknowledgement

- This presentation has been designed using resources from PoweredTemplate.com