Visual Computing

2024/2025

Class 5

3D Visualization, Transformations and Projections

deti departamento de electrónica, telecomunicações e informática

universidade de aveiro

SSS_CV_V2

Agenda

- 3D Transformations
- Projections
- Brief on Shaders’ Anatomy
- Hands On

3D transformations

3D Transformations

- Translation

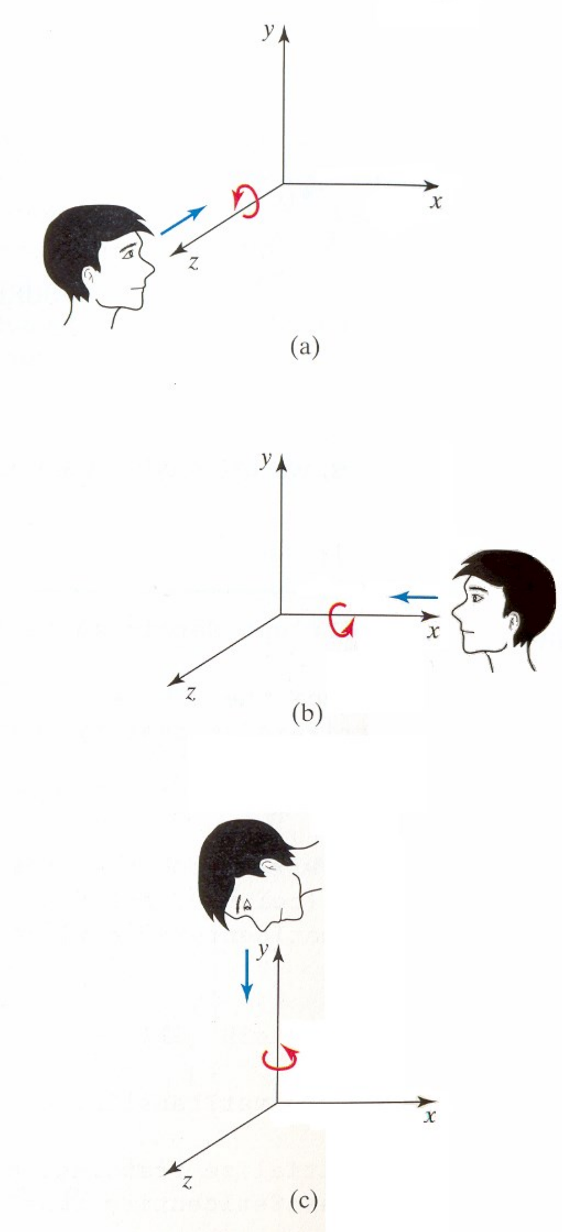
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Scaling

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Rotation

- Rotation around each one of the coordinate axis
- Positive rotations are CCW !!



Rotation around each axis

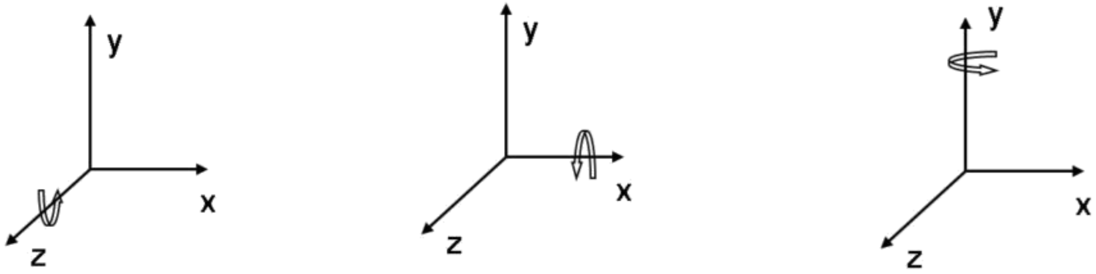
Coordinate-Axes Rotations

- Z-Axis Rotation
- X-Axis Rotation
- Y-Axis Rotation

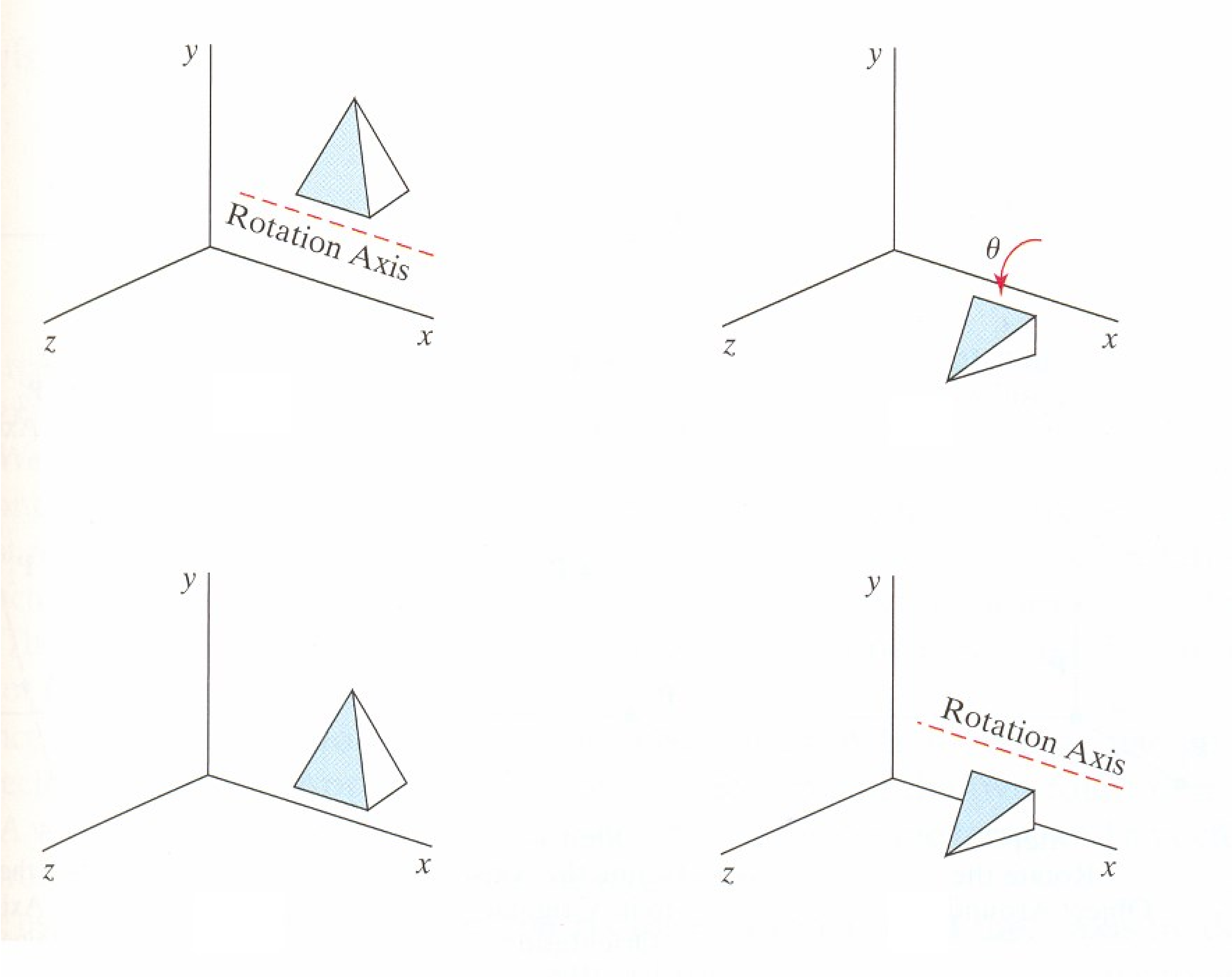
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



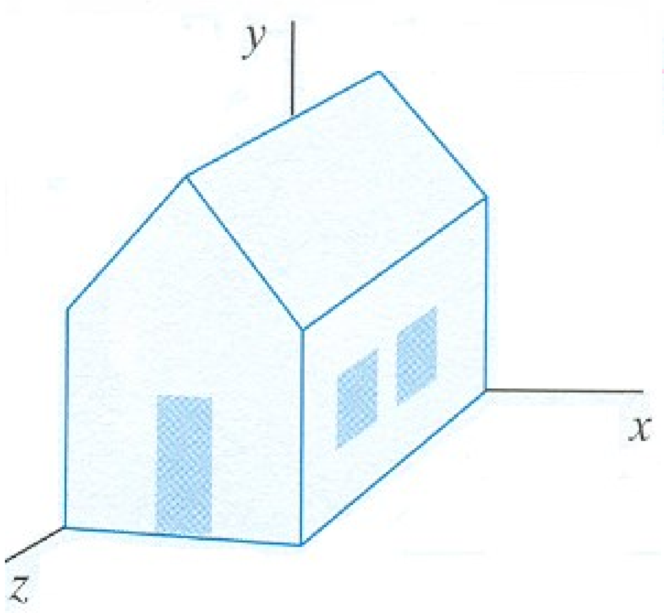
Example – Decomposition

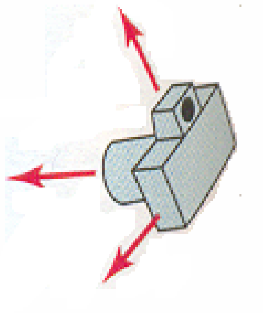


And
inverse
translation
follows

3D projections

3D Viewing





3D Viewing

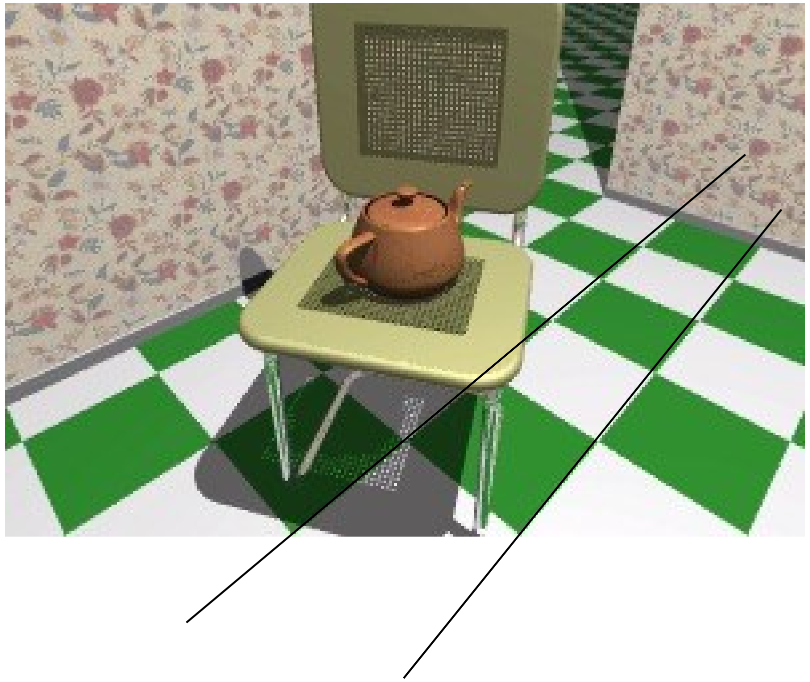
- Where is the observer / the camera ?
    - **Position** ?
    - Close to the 3D scene ?
    - Far away ?

- How is the observer looking at the scene ?
    - **Orientation** ?

- How is the scene represented as a 2D image ?
    - **Projection** ?

Projections

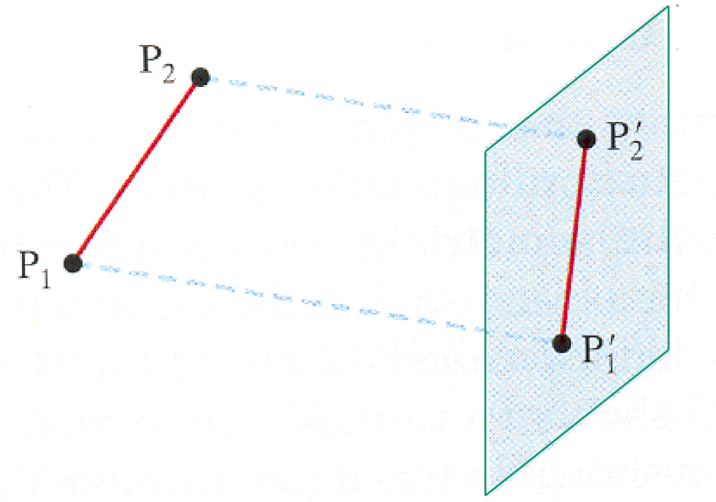
Parallel Projection



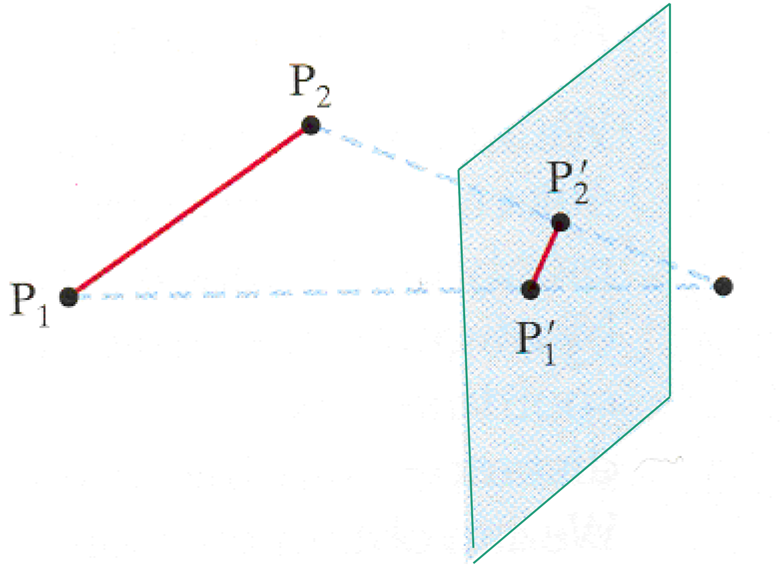
Perspective

Projection

Projections

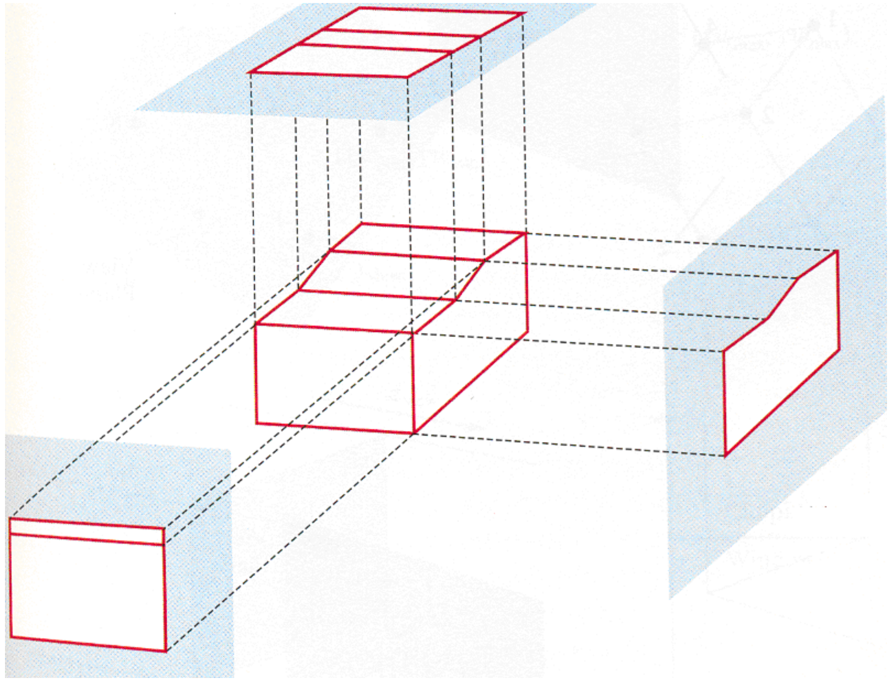


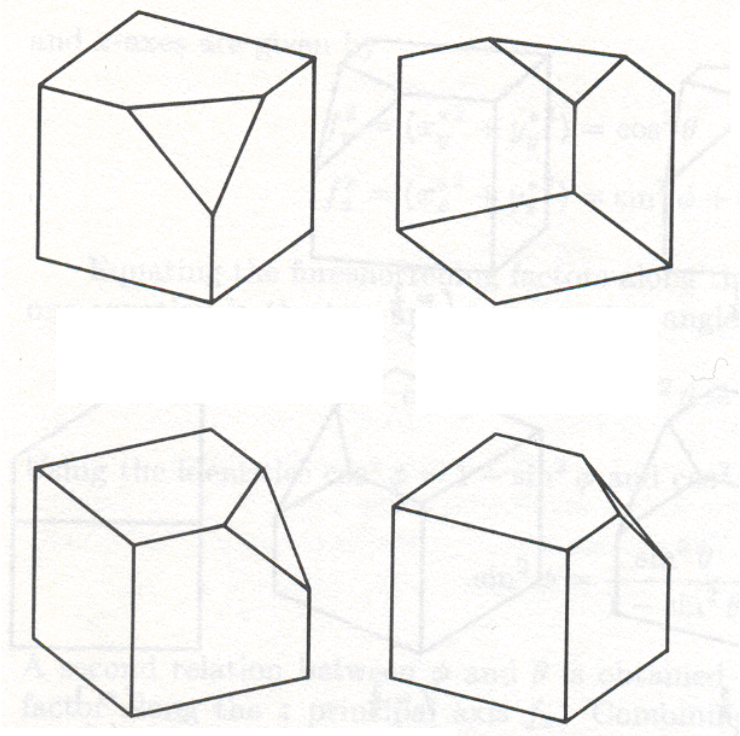
Parallel Projection



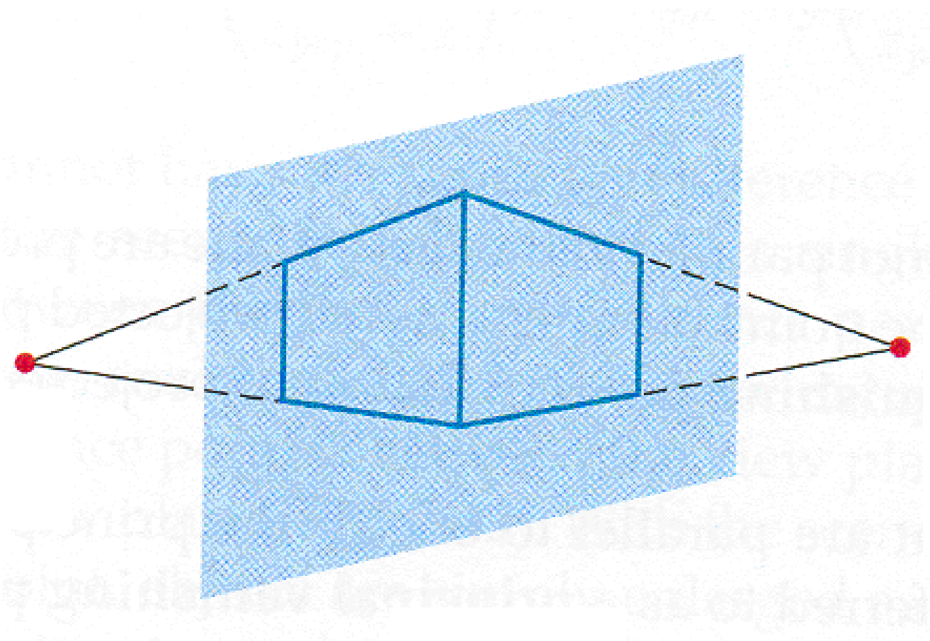
Perspective Projection

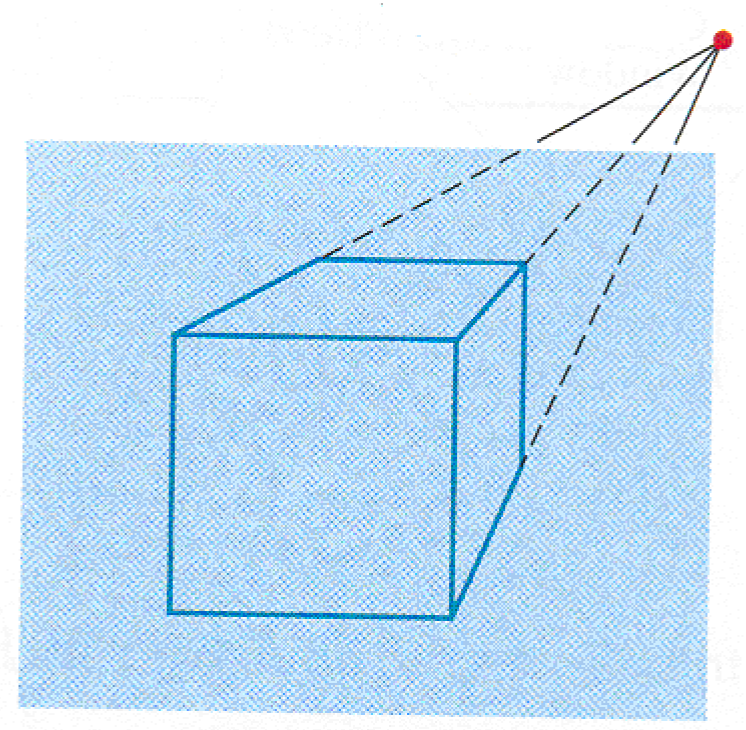
Parallel Projections



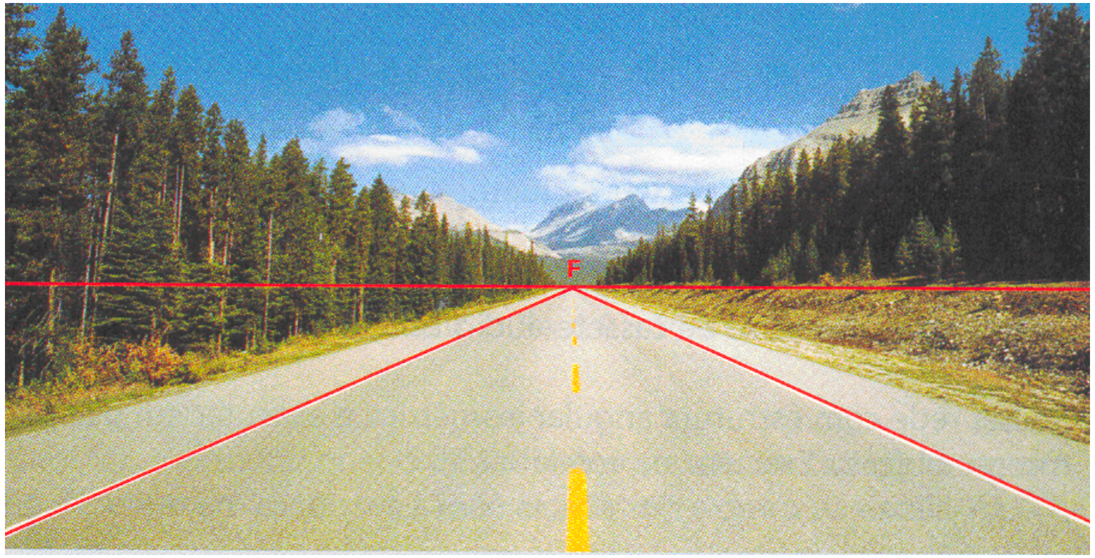


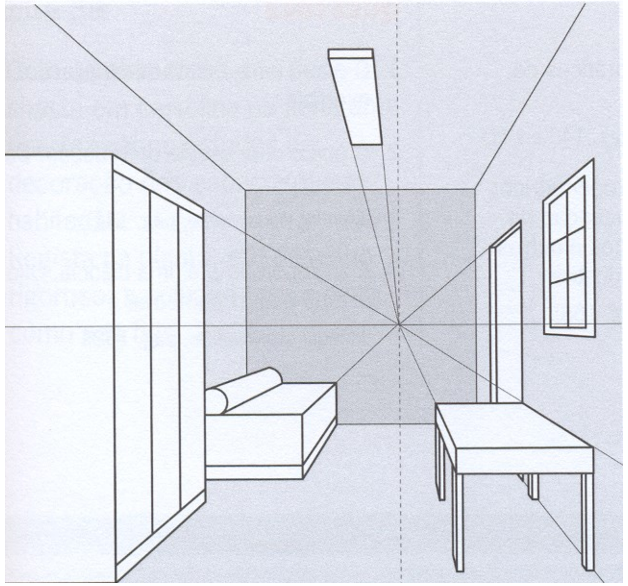
Perspective Projections





Perspective Projections





How to represent ?

- Projection matrices
- Homogeneous coordinates
- Concatenation through matrix multiplication
- Don’t worry! Graphics APIs implement usual projections !

Homogeneous Coordinates

Homogeneous Coordinates

Coordinate system for projective geometry

Formulas involving homogeneous coordinates are often simpler than in the Cartesian approach

A **single matrix** can represent **affine and projective transformations** (remember translations with and without homogeneous coordinates...)

Homogeneous Coordinates

The representation of a geometric object is **homogeneous if x and αx represent the same object** for $\alpha \neq 0$

- x = αx, in homogeneous coordinates
- x $\neq \alpha$x, in Euclidian coordinates, unless $\alpha = 1$

Homogeneous Coordinates

We add one dimension:

$$x = \begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \mathrm{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathrm{x} = \lambda\,\mathrm{x}$$

homogeneous

We can multiply by a constant, and it remains the same object:

$$x = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \mathrm{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ w1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

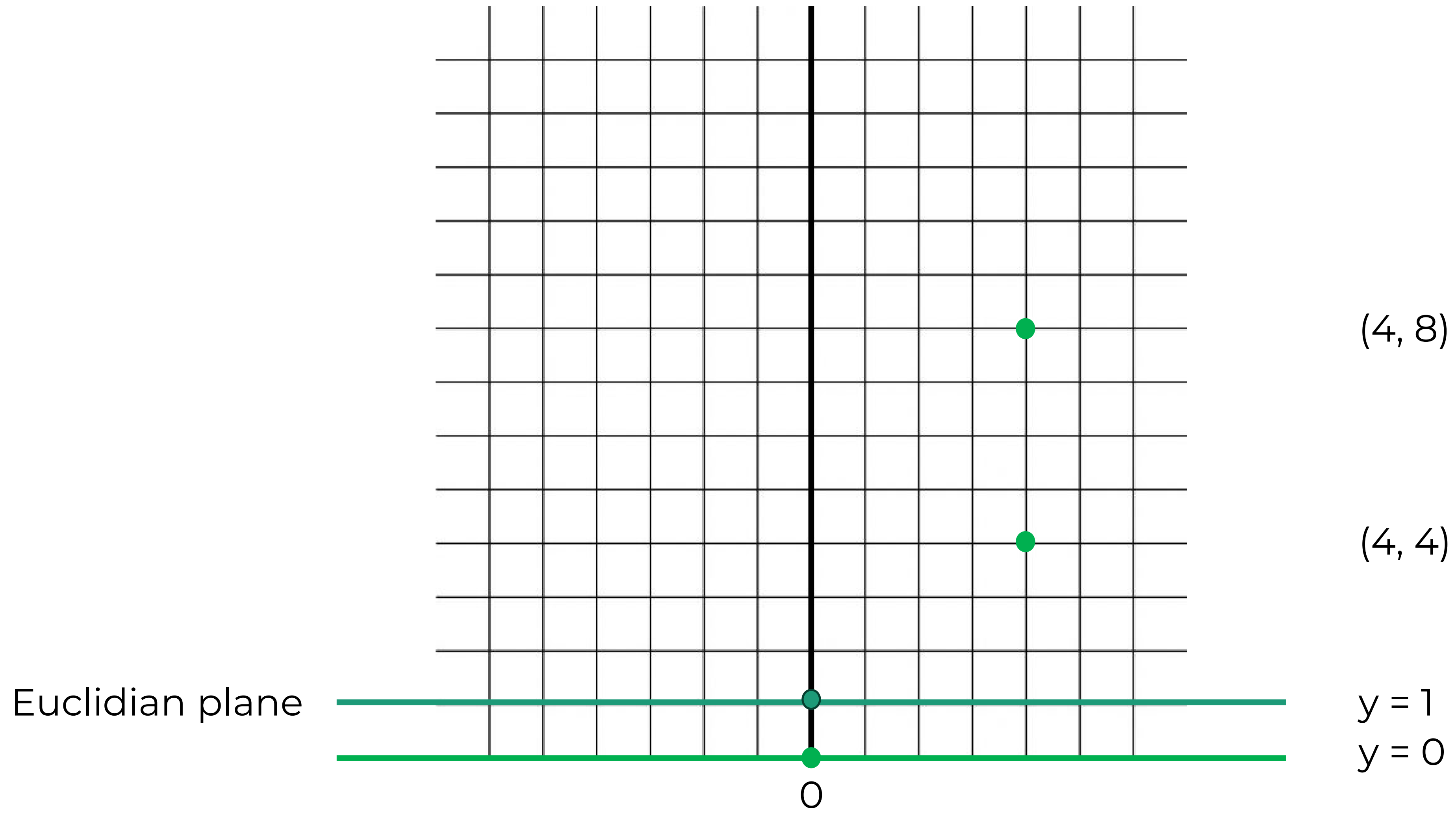
Euclidian homogeneous

Homogeneous Coordinates Converting to Euclidian Space

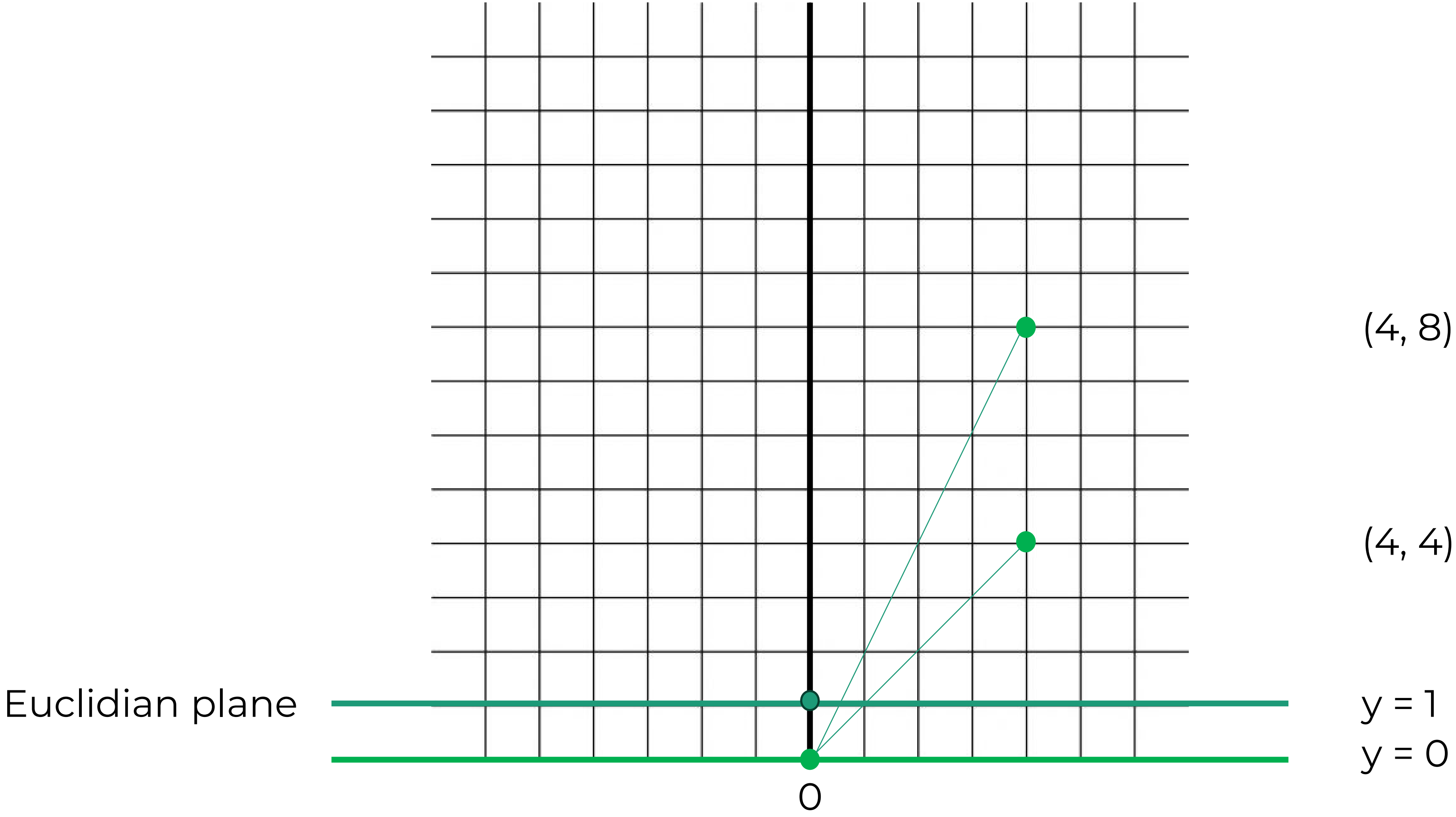
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u/w \\ v/w \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

homogeneous Euclidian

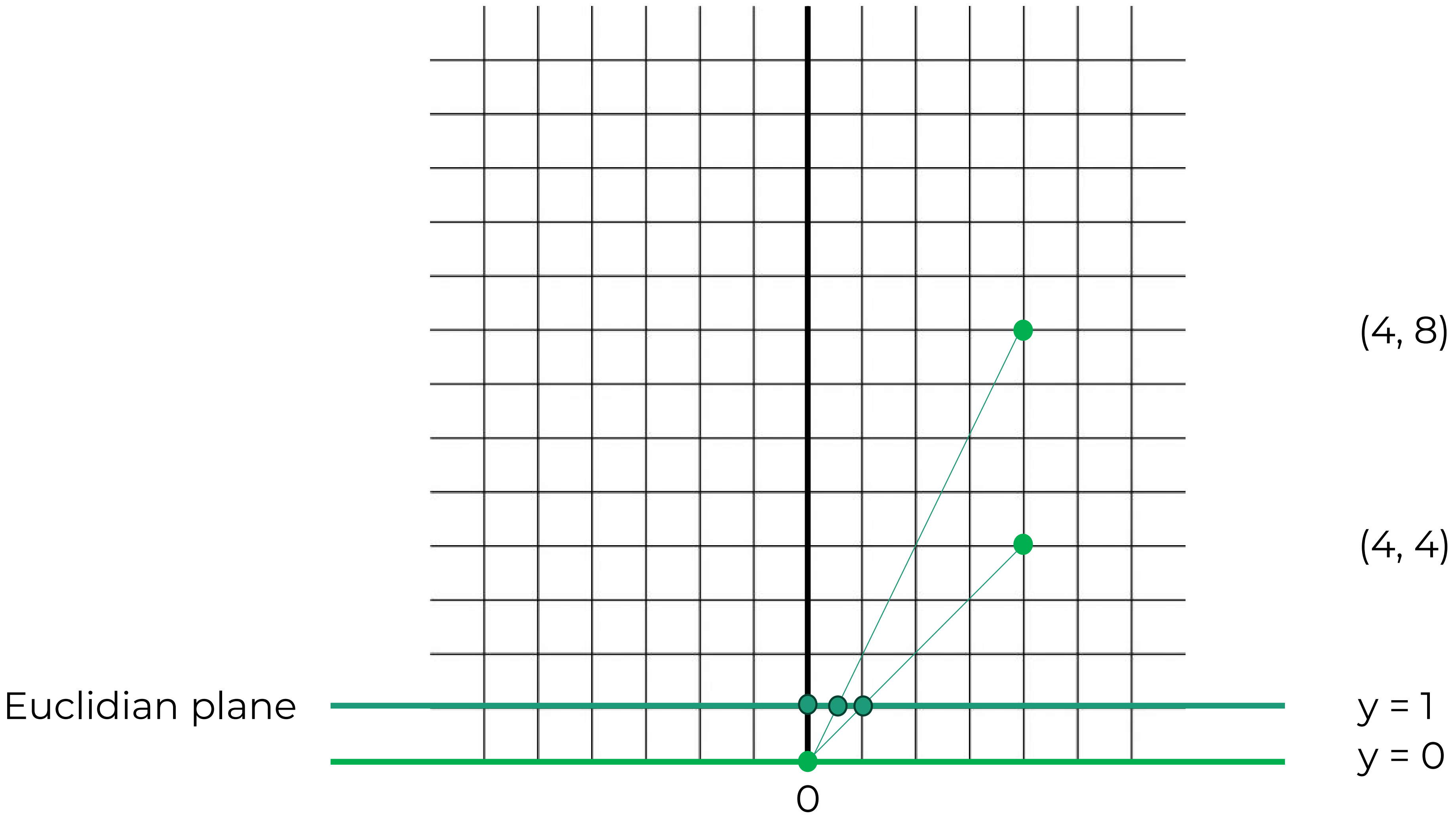
Perspective Projection 2D Example



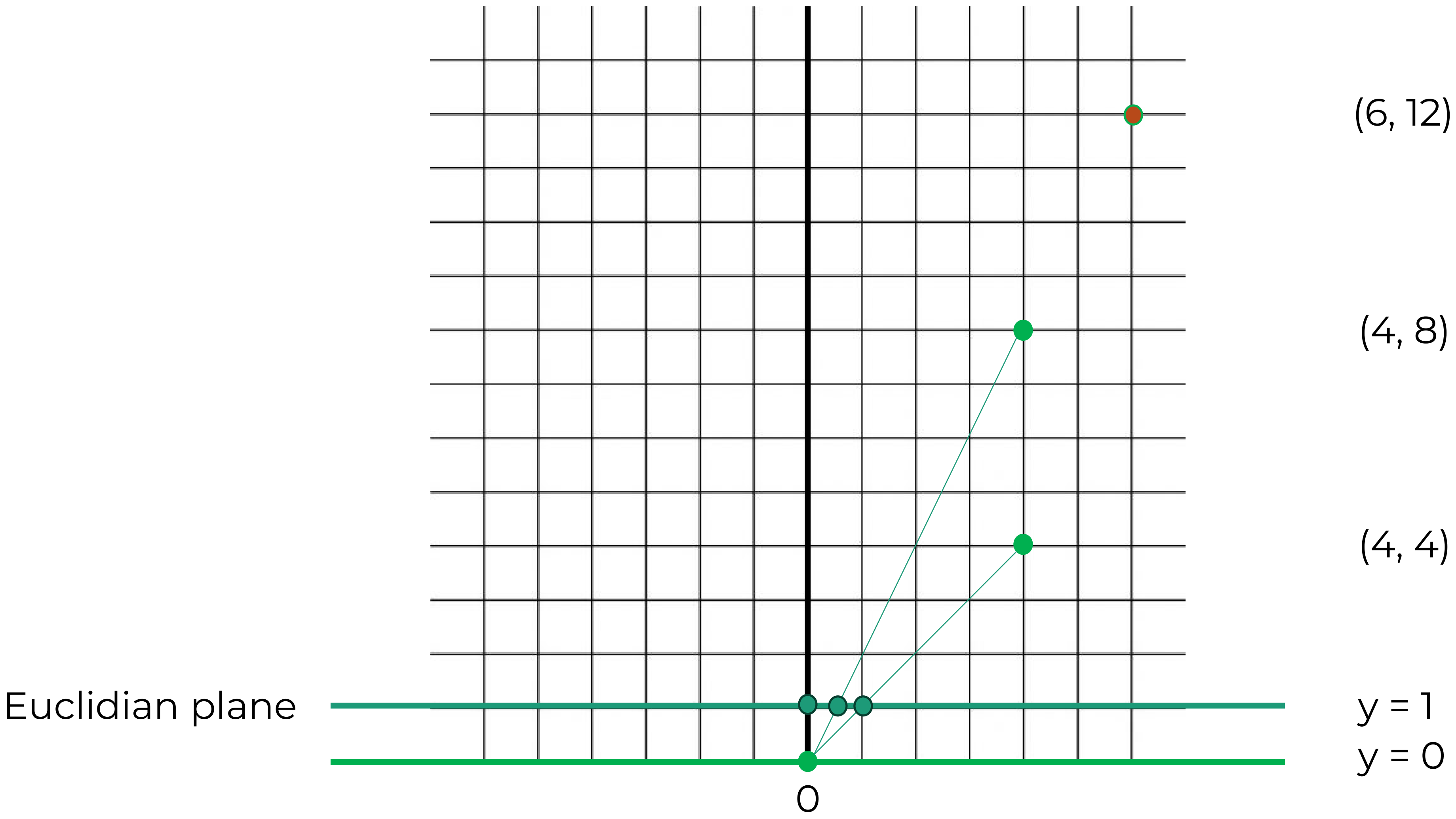
Perspective Projection
2D Example



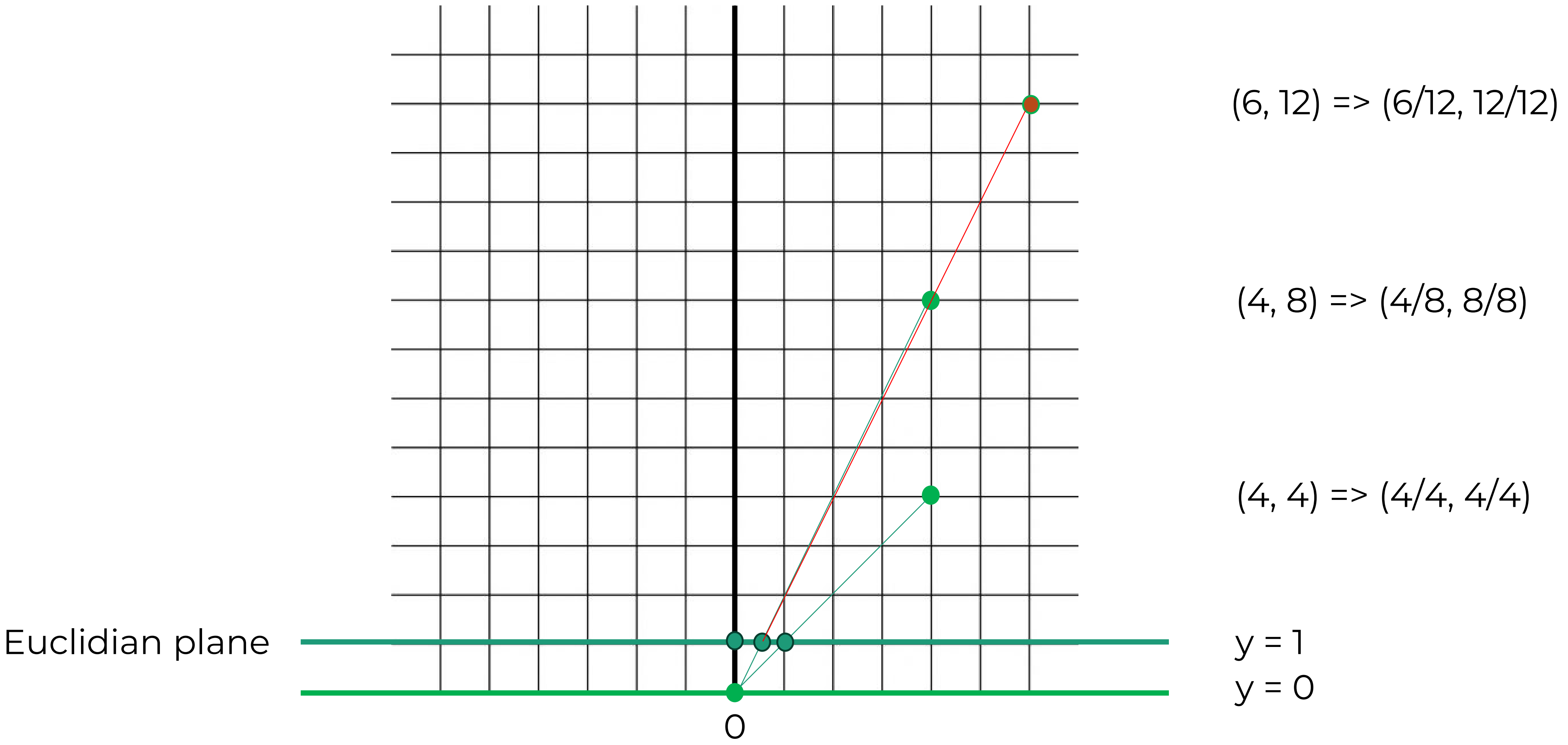
Perspective Projection
2D Example



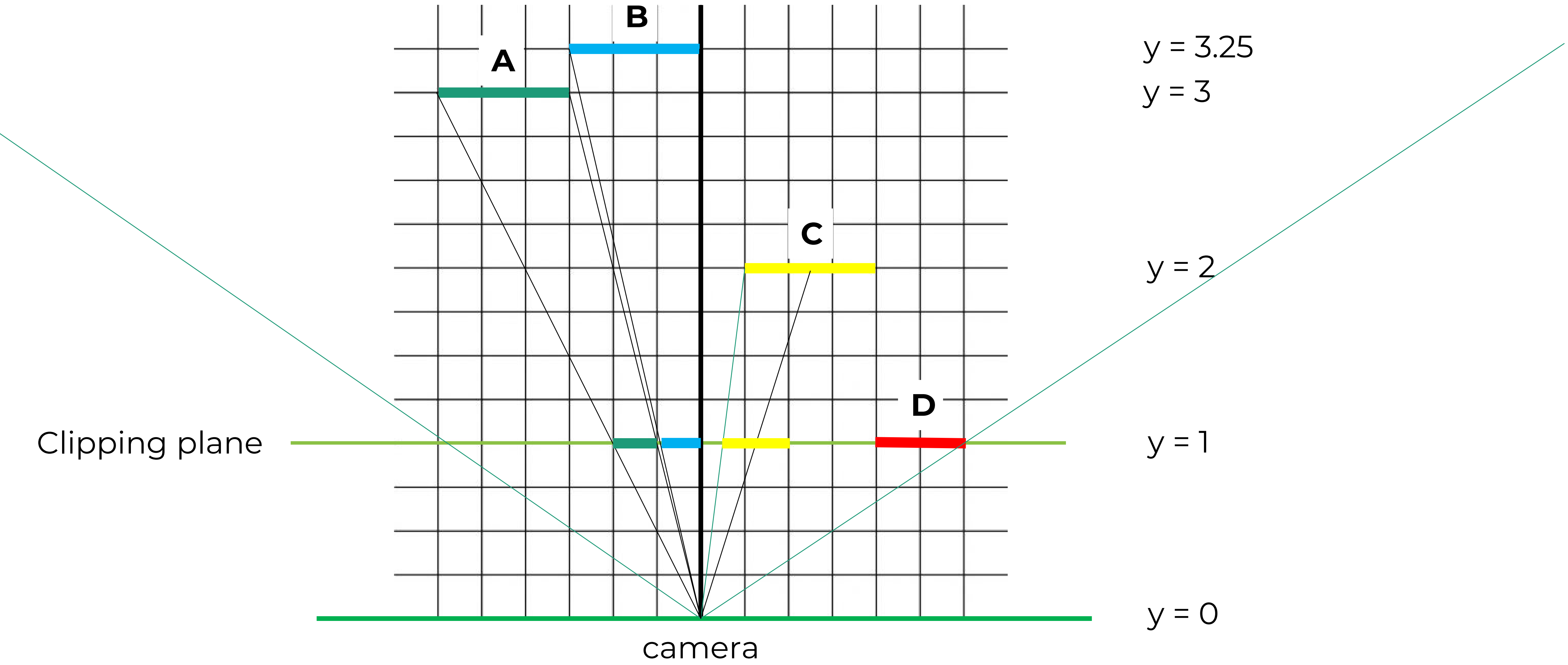
Perspective Projection 2D Example



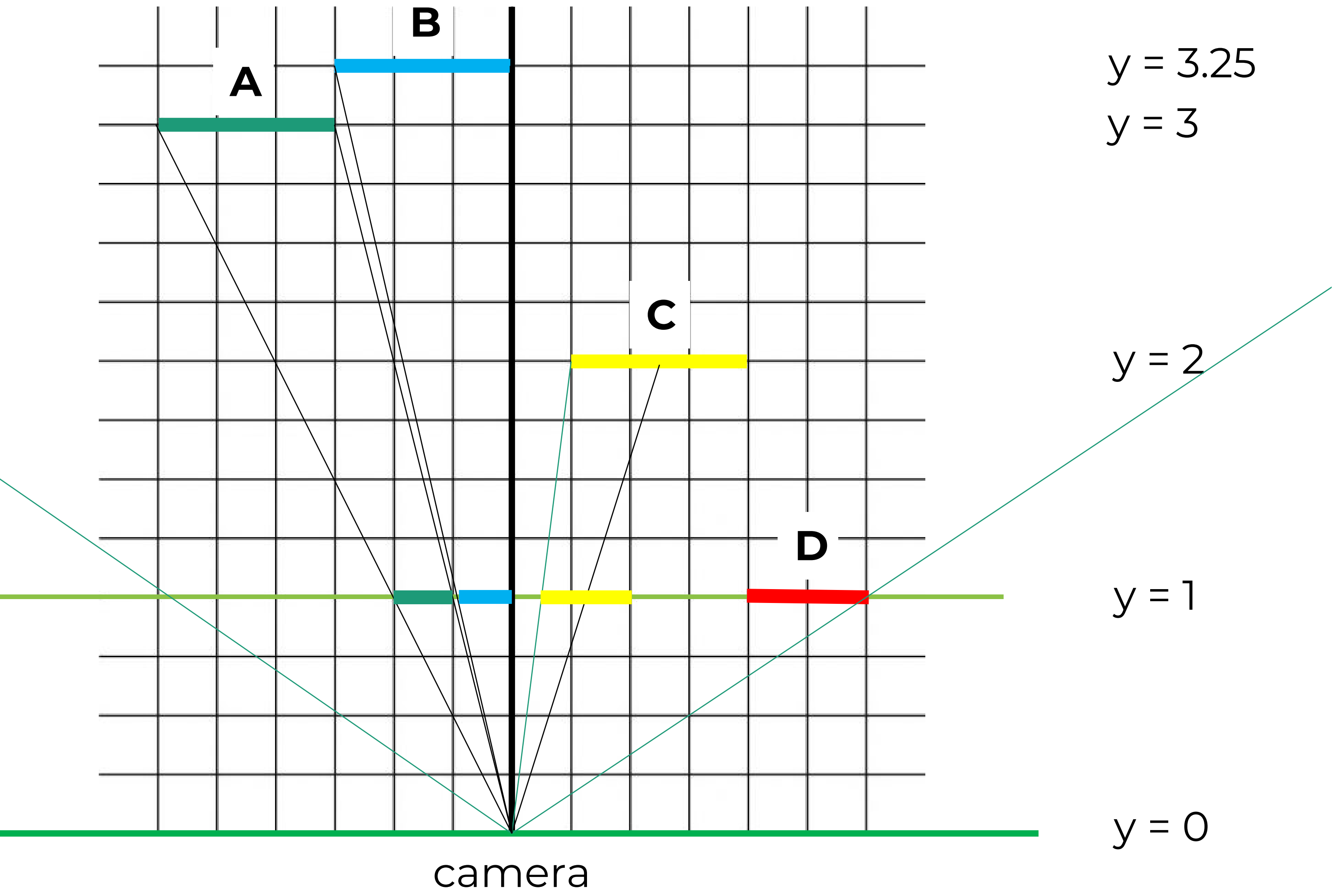
Perspective Projection 2D Example



Perspective Projection 2D Example



Perspective Projection
2D Example



We do the basic projection by dividing all coordinates by the last coordinate!

But, we want to do so using matrices!

How?

We know why we use homogeneous coordinates

Let's go step by step:

1. We have a point $P = \begin{bmatrix} x \\ y \end{bmatrix}$

2. We represent it in homogeneous coordinates to be able to do rotations, scaling, and translations always doing matrix multiplication

$$P_h = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

w

We use them for transformations and then get back

Translations:

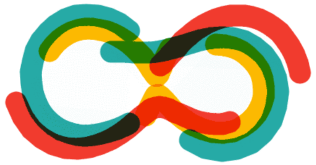
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3. Whenever we want to go back to original coordinates, we just divide everything by *w*

$$\underbrace{\begin{bmatrix} u \\ v \\ w \end{bmatrix}}_{\text{homogeneous}} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} \rightarrow \underbrace{\begin{bmatrix} u/w \\ v/w \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}}_{\text{Euclidian}}$$

But what matrix can be used for perspective projections?

Loading dumb intelligence model to draw figure
Please wait



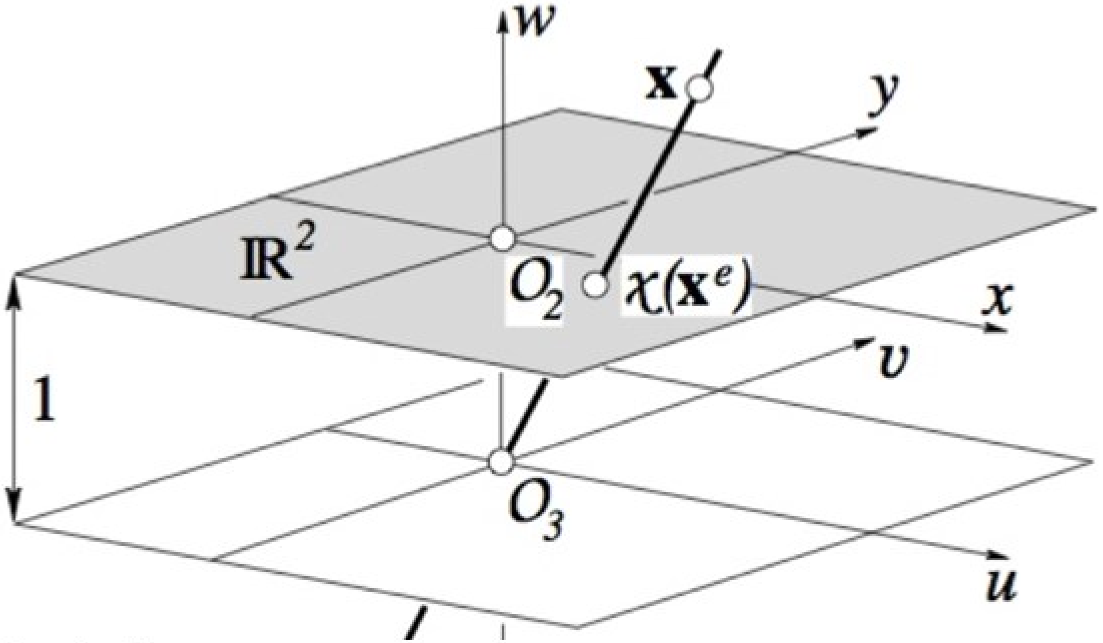
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x/y \\ y/y \\ y/y \end{bmatrix} = \begin{bmatrix} x/y \\ 1 \\ 1 \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} x/y \\ 1 \end{bmatrix}$$

Error! No artistic qualities!

Homogeneous Coordinates
2D Example



$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u/w \\ v/w \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

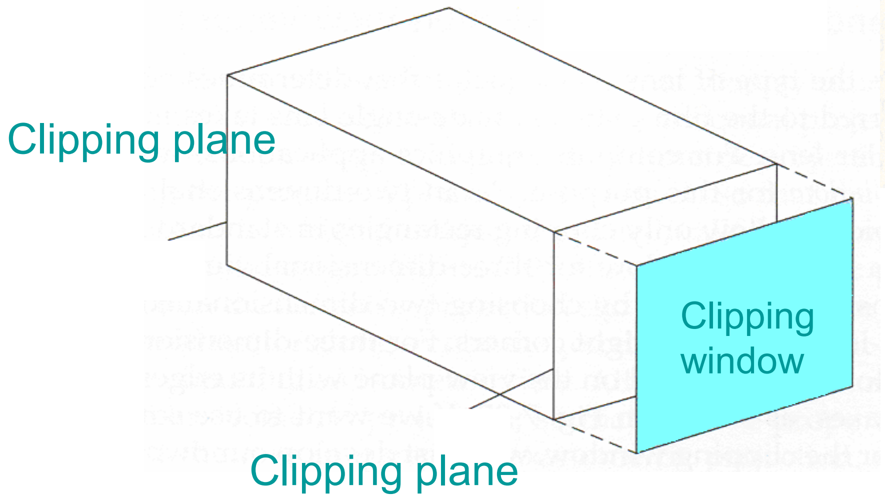
Image courtesy: Förstner 20

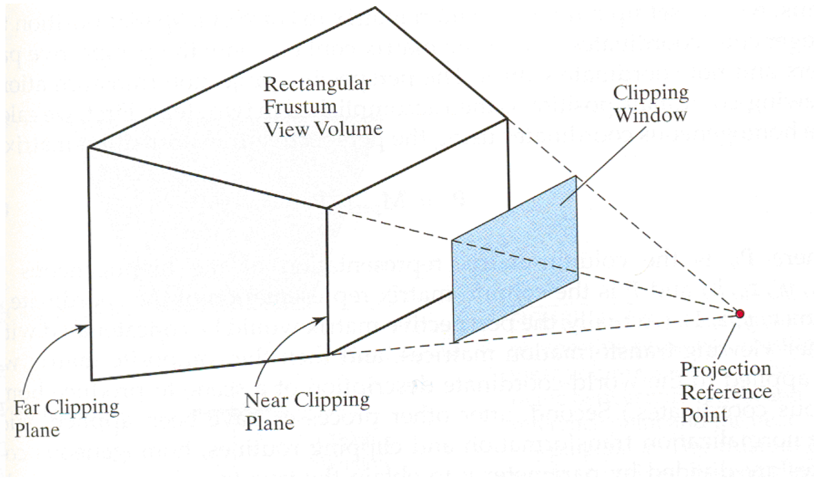
View volume & clipping

How to limit what is observed ?

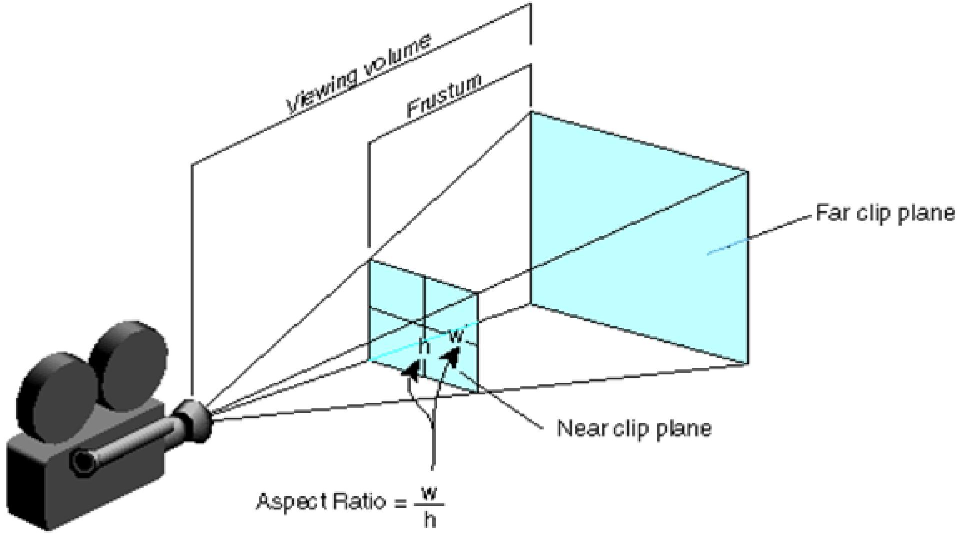
- **Clipping window** on the projection plane

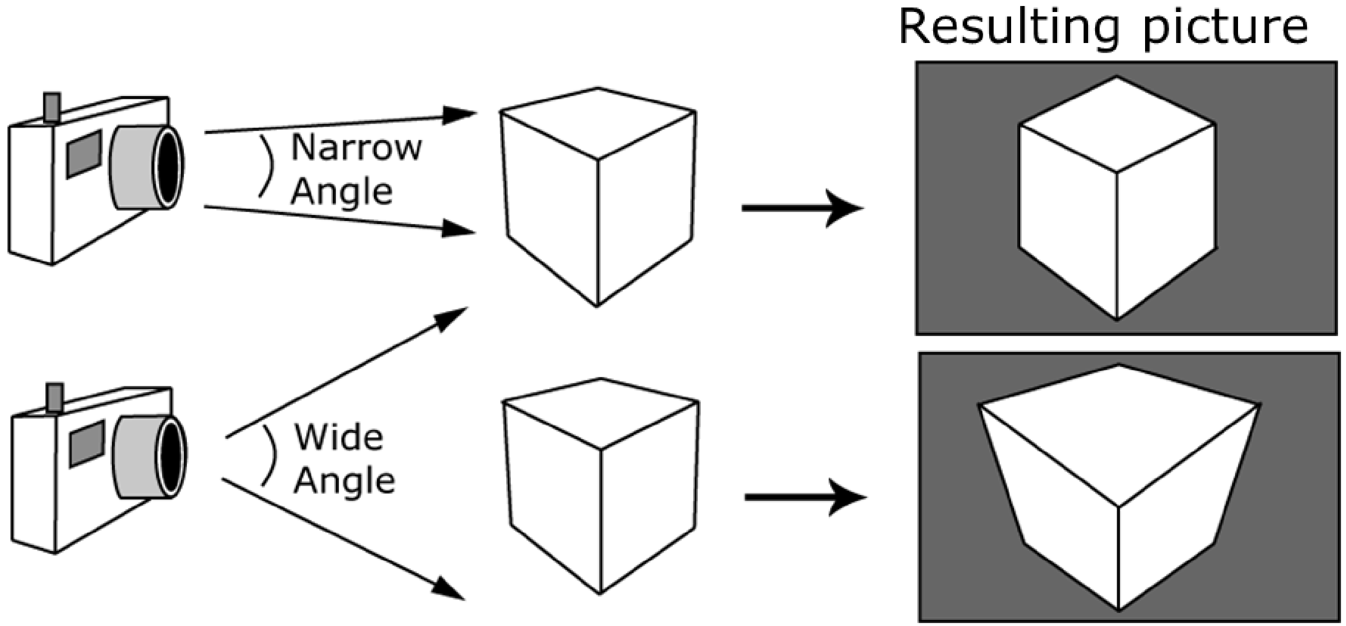
- **View volume** in 3D





View Angle

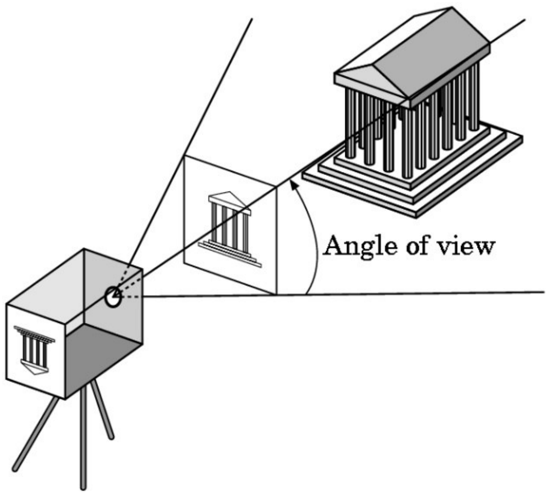


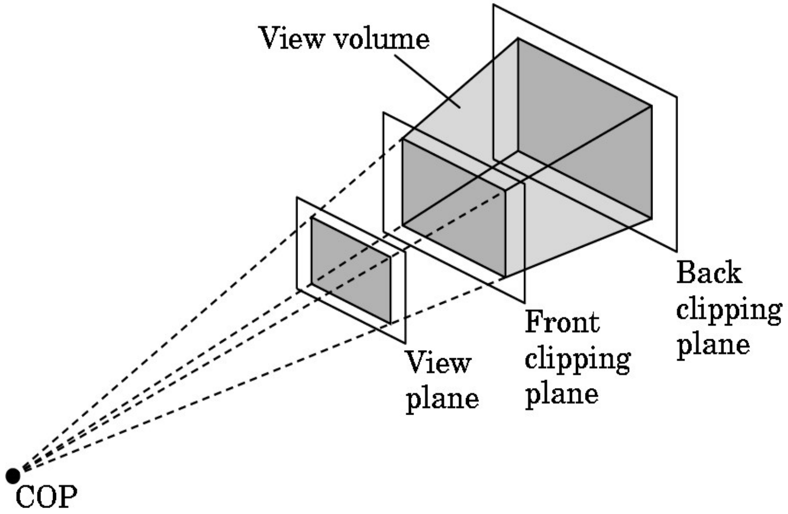


[van Dam]

Clipping
View Volume

The virtual camera only “sees” part of the world or object space

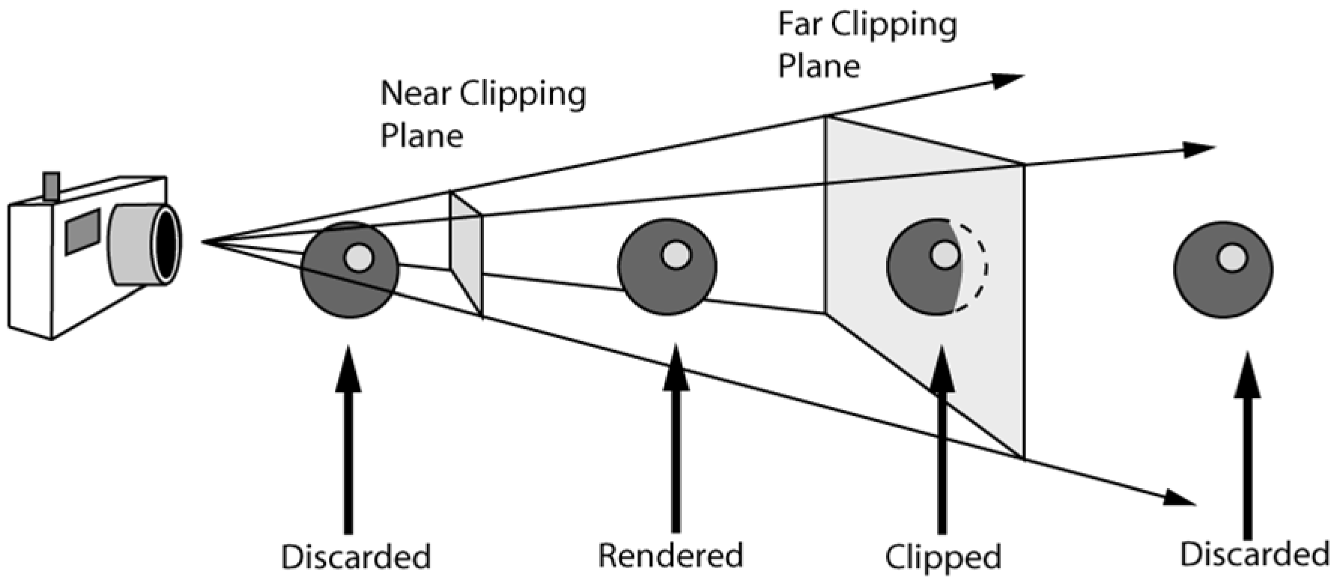




[Angel]

Clipping Planes

View Volume



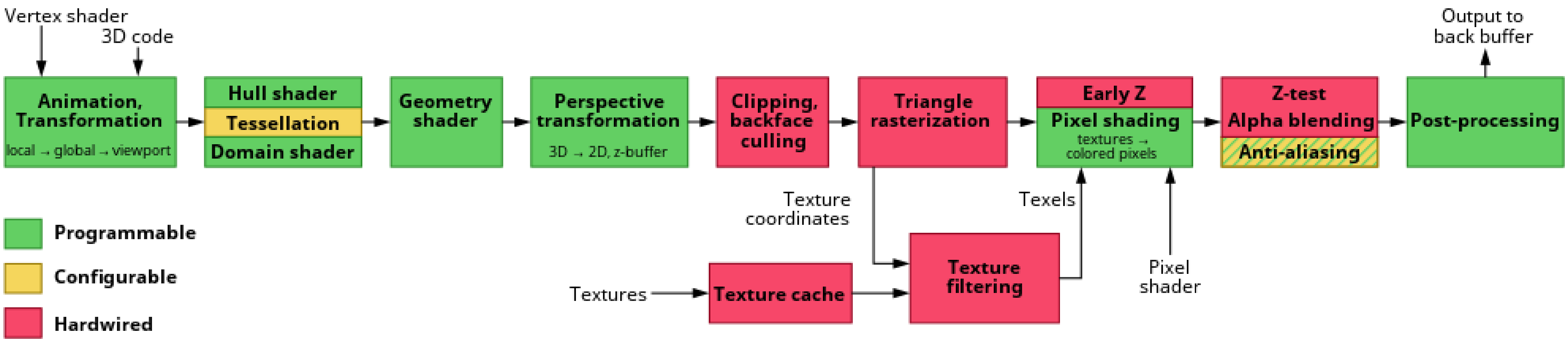
[van Dam]

3D Viewing

- How to view primitives that are **outside** the view volume ?
  - **Translate** !
- How to view **a side face** of a model ?
  - **Rotate** !
- …

The 3D visualization pipeline

3D visualization pipeline



3D visualization pipeline

- Instantiate **models**
  - Position, orientation, size
- Establish **viewing parameters**
  - Camera position and orientation
- **Define projection**
- Perform **clipping**
- **Rasterize**
- Compute **illumination** and **shade polygons**

3D visualization pipeline

- Main operations represented as **point transformations**
  - Homogeneous coordinates
  - Transformation matrices
  - Projection matrix
  - Matrix multiplication
- Each object is processed **separately**

Shaders

What you’ve seen on the hands on...

Down the shader's hole

Script side

- Object defined by its vertices, colors, normals, etc

```
self.triangle_vertices = array([
        [0, 1, 0, 1, 1, 1],
        [1, -1, 0, 0, 1, 1],
        [-1, -1, 0, 1, 1, 0]
    ], 'f')
```

- Data placed on the GPU:

```
self.triangle_vbo = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER, self.triangle_vbo)
glBufferData(GL_ARRAY_BUFFER, 4 * self.triangle_vertices.size, self.triangle_vertices, GL_STATIC_DRAW)
```

Down the shader’s hole

Script side

- Just before rendering, what attributes are available and where are they are placed in the GPU?

```
self.triangle_vertices = array([
    [0, 1, 0, 1, 1, 1],
    [1, -1, 0, 0, 1, 1],
    [-1, -1, 0, 1, 1, 0]
  ], 'f')
```

```
glBindBuffer(GL_ARRAY_BUFFER, self.triangle_vbo)

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6*4, None)
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*4, ctypes.c_void_p(3 * 4))

glEnableVertexAttribArray(0)
glEnableVertexAttribArray(1)
```

Down the shader's hole

Script side

- Do I need to pass additional data to the shader? Remember the transformation matrices

- We will see, ahead, how do we receive this data on the shader

```
# translation
self.modelMatrix = glm.translate(self.modelMatrix, tx, ty, tz)

modelLoc = glGetUniformLocation(self.shader, "model")
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, self.modelMatrix)
```

Down the shader's hole

Shader side

- What attributes am I expecting and their locations

- Using the attributes

Vertex shader

```
#version 330
    layout (location = 0) in vec3 position;
    layout (location = 1) in vec3 color;

    uniform mat4 model;
    uniform mat4 view;
    uniform mat4 projection;

    out vec3 vColor;

    void main() {
    gl_Position =  projection * view * model * vec4(position, 1.0);
    vColor = vec3(color);
}
```

Down the shader's hole

Shader side

- Passing attributes out from a shader
- Receiving attributes in the next shader

Vertex shader

```
#version 330
    layout (location = 0) in vec3 position;
    layout (location = 1) in vec3 color;

    uniform mat4 model;
    uniform mat4 view;
    uniform mat4 projection;

    out vec3 vColor;

    void main() {
    gl_Position =  projection * view * model * vec4(position, 1.0);
    vColor = vec3(color);

}
```

Fragment shader

```
#version 330
    in vec3 vColor;
    out vec4 out_color;

    void main() {
        out_color = vec4(vColor, 1.0);
```