



Universidade de Aveiro  
Departamento de Electrónica,  
Telecomunicações e Informática

# Introduction to Computer Graphics

## main concepts and methods - II



(Wikipedia)

# Topics

- Computer Graphics main tasks
- Geometric Primitives
- Geometric transformations
- 2D and 3D visualization
- Projections

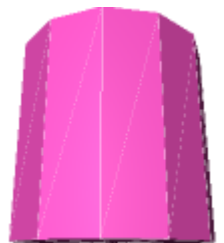
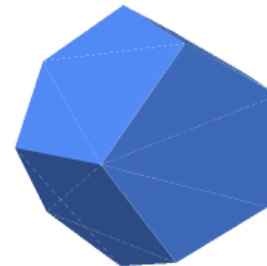
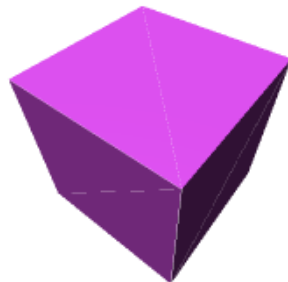
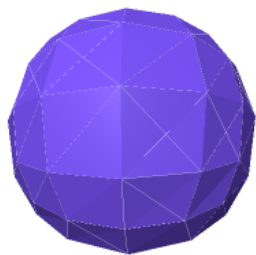
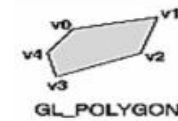
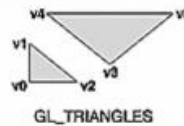
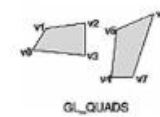
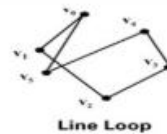
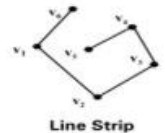
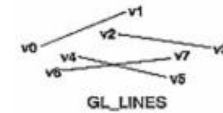
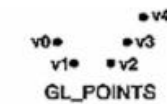
# CG Main Tasks

- Modeling
  - Construct individual models / objects
  - Assemble them into a 2D or 3D scene (using **transformations**)
- Rendering
  - Generate final images:
  - How is the scene illuminated?
  - What are the materials of the objects?
  - Where is the observer? How is he/she **looking at the scene?**
- Animation
  - Static vs. dynamic scenes
  - Movement and / or deformation

# Geometric Primitives

- Simple primitives
  - Points
  - Line segments
  - Polygons
- Geometric primitives
  - Parametric curves / surfaces
  - Cubes, spheres, cylinders, etc.

Examples:



# Computer Graphics APIs

- Create 2D / 3D scenes from simple primitives



- OpenGL and variants ...

- Rendering
- No modeling or interaction facilities



- Direct 3D – Microsoft



- VTK



- 3D CG + Image processing + Visualization

- Three.js



- Vulkan ...



# Three.js

- Cross-browser JavaScript library/API used to create and display animated 3D computer graphics in a web browser.
- Uses WebGL

three.js <sup>r87</sup>

featured projects

[submit project](#)

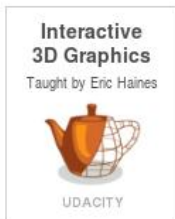
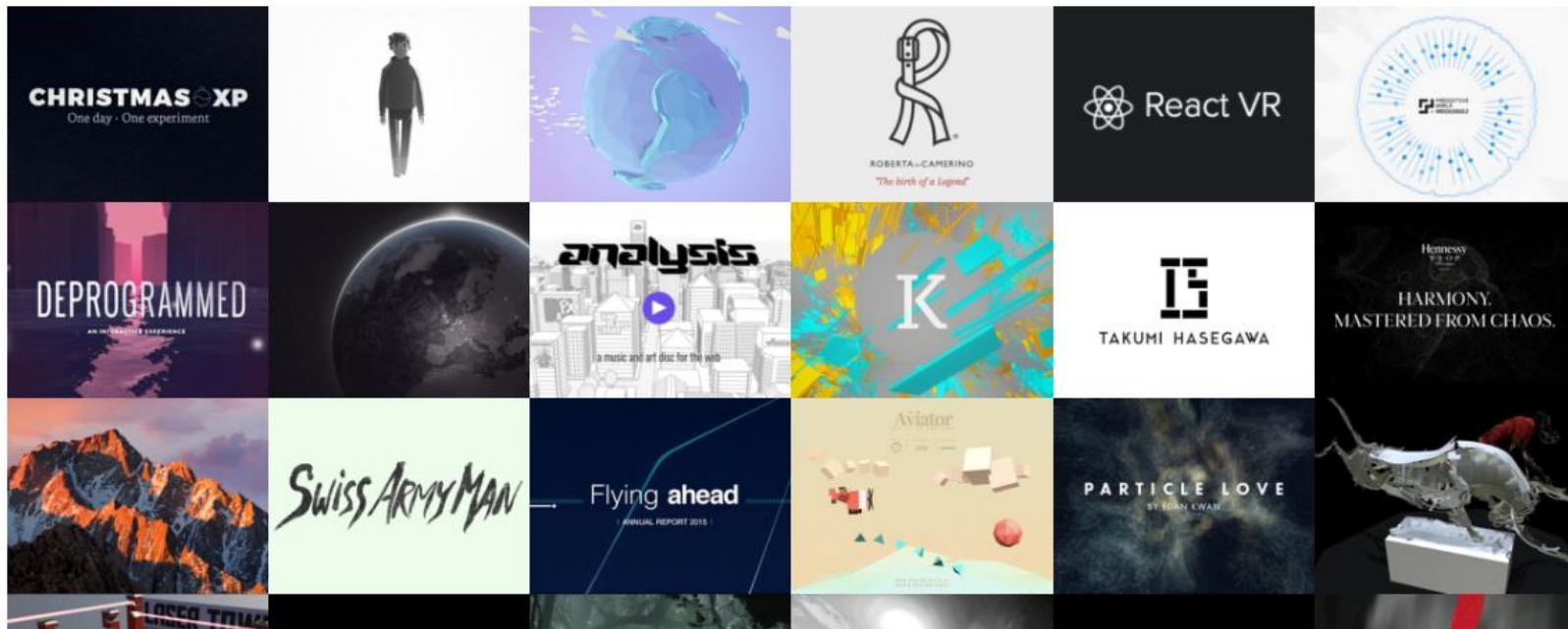
[documentation](#)  
[examples](#)

[download](#)

[source code](#)  
[questions](#)  
[forum](#)

[irc](#)  
[slack](#)  
[google+](#)

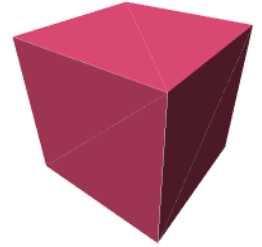
[editor](#)



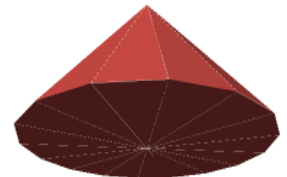
<https://threejs.org/>

# Geometric Primitives – three.js examples

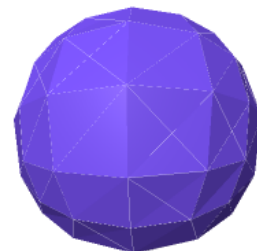
```
const width = 8; // ui: width
const height = 8; // ui: height
const depth = 8; // ui: depth
const geometry = new THREE.BoxGeometry(width, height, depth)
```



```
const radius = 6; // ui: radius
const height = 8; // ui: height
const radialSegments = 16; // ui: radialSegments
const geometry = new THREE.ConeGeometry(radius, height, radialSegments);
```

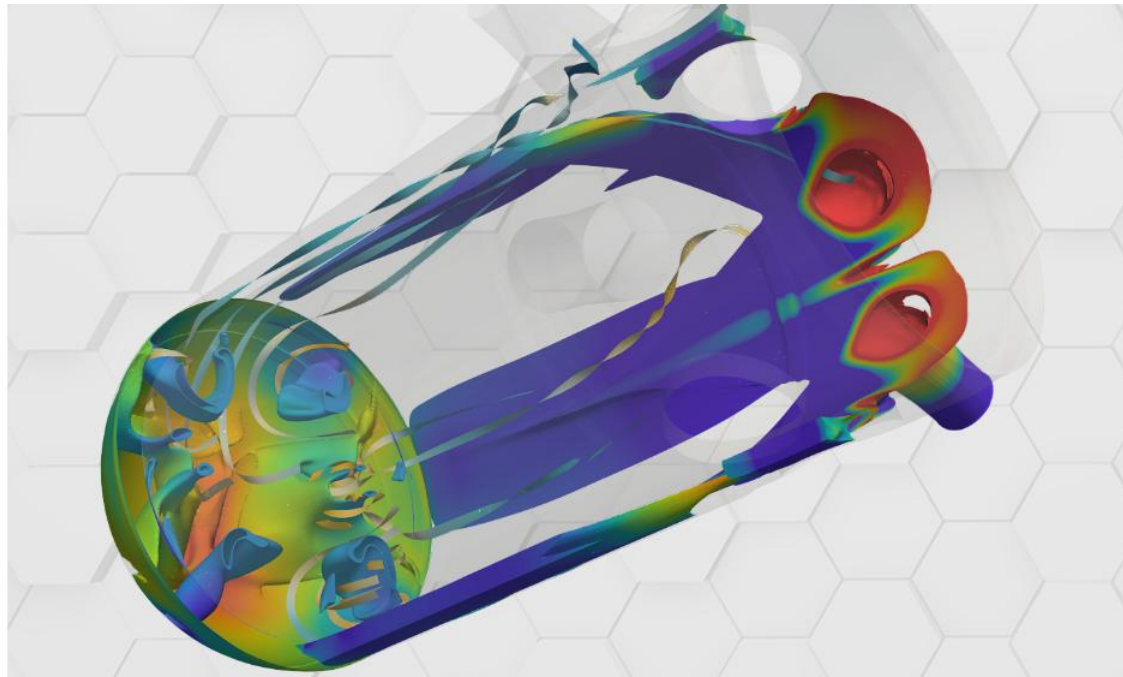


```
const radius = 7; // ui: radius
const widthSegments = 12; // ui: widthSegments
const heightSegments = 8; // ui: heightSegments
const geometry = new THREE.SphereGeometry(radius, widthSegments, heightSegments);
```



# VTK

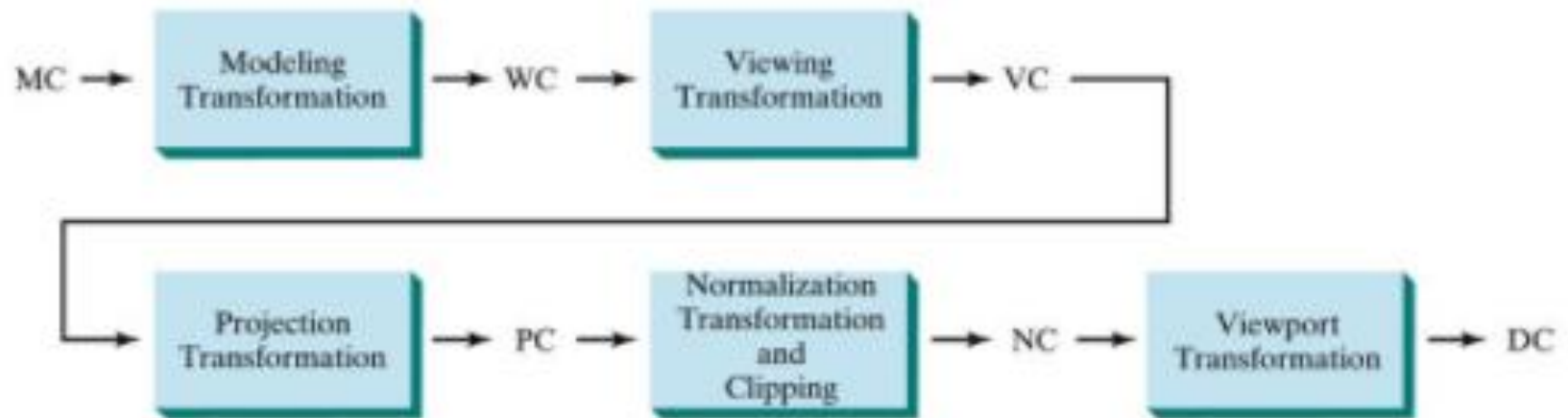
- Open-source, freely available software system for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization.
- Is designed to be platform agnostic



<https://vtk.org/>



# 3D visualization pipeline (coordinate transformations)



MC - Modelling coordinates

WC – World coordinates

VC – Viewing coordinates

PC – Projection coordinates

NC – Normalized coordinates

DC – Display coordinates

(Hearn & Baker, 2004)

# 2D and 3D visualization pipeline

- We will start by 2D transformations and viewing a 2D scene and then generalize to 3D
- Main operations represented as point transformations
  - Basic transformation matrices
  - Homogeneous coordinates
  - Matrix multiplication and composed transformations

# Basic 2D Transformations

$p = (x, y) \rightarrow$  *original point*

$p' = (x', y') \rightarrow$  *transformed point*

- Basic transformations:

- Translation

- Scaling

- Rotation

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

 Vector notation

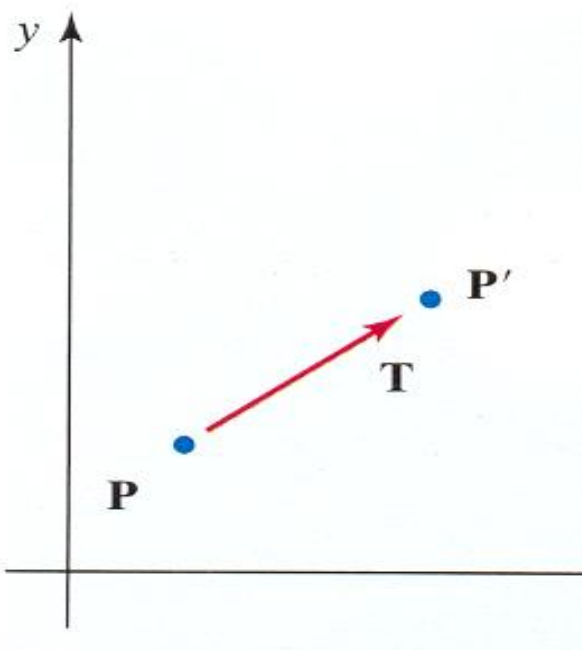
Complex transformations may be expressed as a composition of these

# Translation

- It is necessary to specify translations in  $x$  and  $y$

$$x' = x + t_x \quad y' = y + t_y$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$

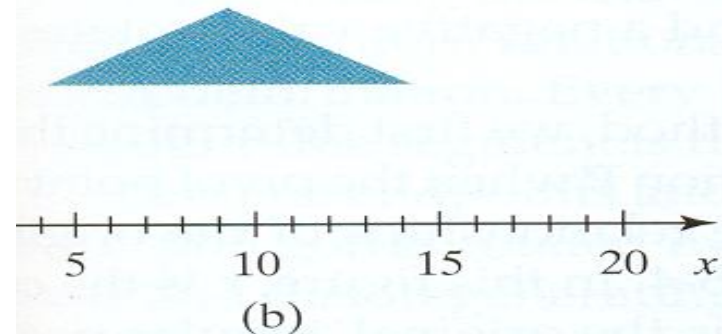
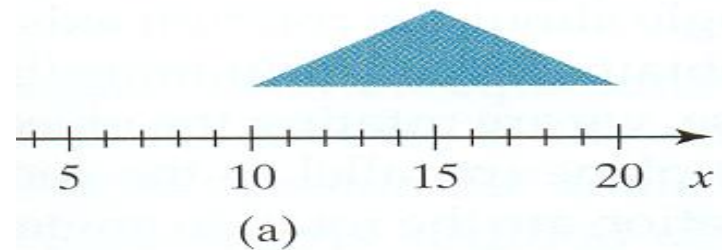
transformation matrix

# Translation

- It is a **rigid body transformation** (it does not deform the object)

- To apply a translation to a line segment we need only to transform the end points

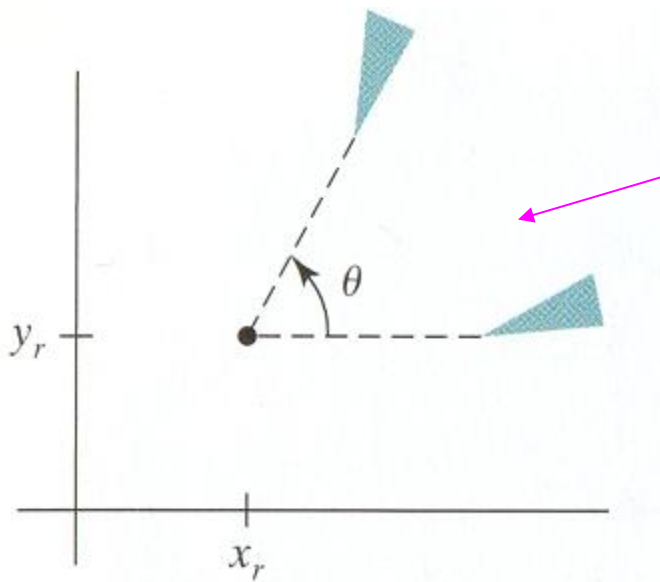
- To apply a translation to a polygon we need only to transform the vertices



(Hearn & Baker, 2004)

# Rotation

- To apply a 2D rotation we need to specify:
  - a point (center of rotation)  
 $(x_r, y_r)$
  - a rotation angle  $\theta$  (the convention is: positive  $\rightarrow$  counter-clockwise)



*Positive rotation*

- the simplest case is a rotation around the origin  $(0,0)$

# Rotation around the origin

- The simplest case:

$$x' = r \cos (\Phi + \Theta) = r \cos \Phi \cos \Theta - r \sin \Phi \sin \Theta$$

$$y' = r \sin (\Phi + \Theta) = r \cos \Phi \sin \Theta + r \sin \Phi \cos \Theta$$

Polar coordinates of the original point:

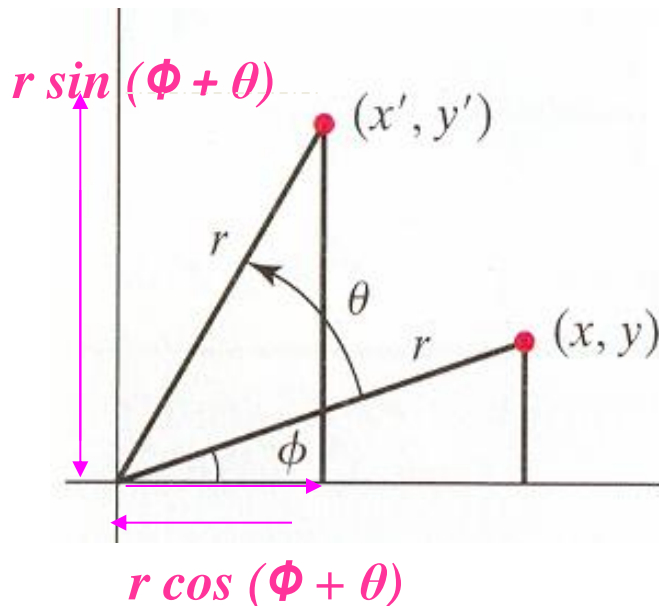
$$x = r \cos \Phi$$

$$y = r \sin \Phi$$

Replacing:

$$x' = x \cos \Theta - y \sin \Theta$$

$$y' = x \sin \Theta + y \cos \Theta$$



## 2D Rotation in matrix notation

$$x' = r \cos (\Phi + \Theta) = r \cos \Phi \cos \Theta - r \sin \Phi \sin \Theta$$

$$y' = r \sin (\Phi + \Theta) = r \cos \Phi \sin \Theta + r \sin \Phi \cos \Theta$$

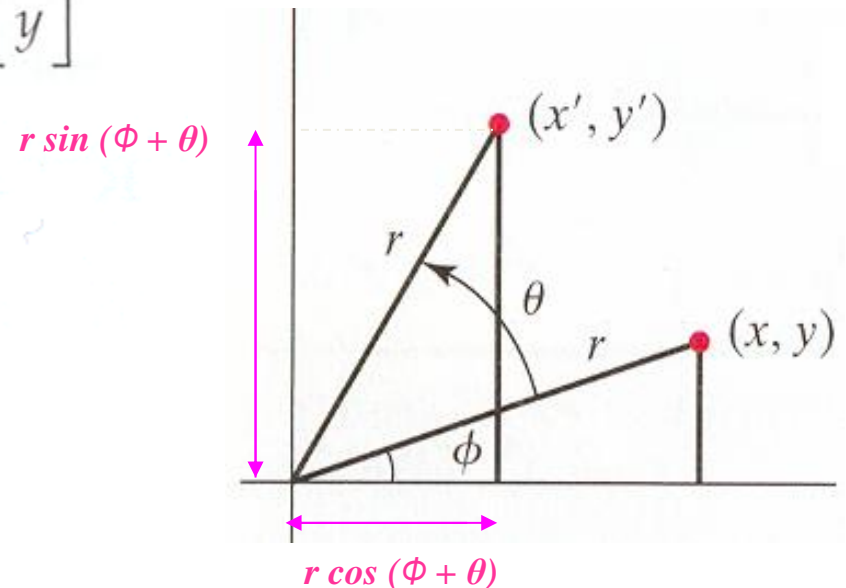
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$

Reminder:

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

$$\cos(\alpha - \beta) = \cos \alpha \cos \beta + \sin \alpha \sin \beta$$



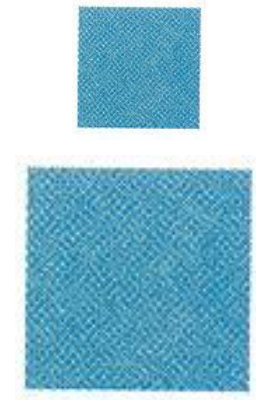


# Scaling

- Modifies the size of an object; we need to specify **scaling factors**:  $s_x$  and  $s_y$

$$\begin{aligned}x' &= x \cdot s_x \\ y' &= y \cdot s_y\end{aligned}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



Transformation matrix

$$P' = S \cdot P$$

Transforming a square into a larger square applying a scaling  $s_x=2, s_y=2$

(Hearn & Baker, 2004)

# 2D Transformations (composed)

- Matrix representation
  - Homogeneous coordinates !!
  - Concatenation = Matrix products
- Complex transformations ?
  - Decompose into a sequence of basic transformations

# Homogeneous coordinates

- Most applications involve **sequences of transformations**
- For instance:
  - visualization transformations involve a sequence of translations and rotations to render an image of a scene
  - animations may imply that an object is rotated and translated between two consecutive frames
- **Homogeneous coordinates provide an efficient way to represent and apply sequences of transformations**

- It is possible to combine in a matrix the multiplying and additive terms if we use **3x3 matrices**
- **All transformations may be represented by multiplying matrices in homogenous coordinates**
- Each point is now represented by 3 coordinates

$$(x, y) \rightarrow (x_h, y_h, h), \quad h \neq 0$$

$$x = x_h / h \quad y = y_h / h$$

$$(x.h, y.h, h)$$

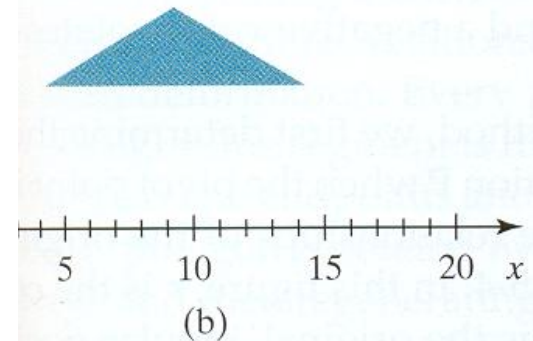
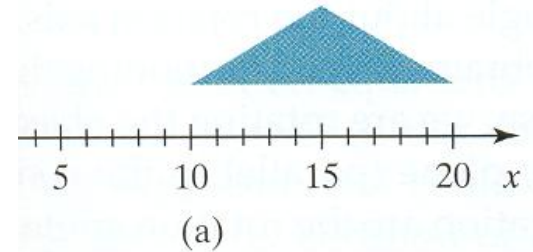
- The simplest way:  $h=1$  !

# 2D Translation (in homogeneous coordinates)

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

One more row and column  
(but not many more operations...)

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$



(Hearn & Baker, 2004)

Now, translation is applied by matrix multiplication

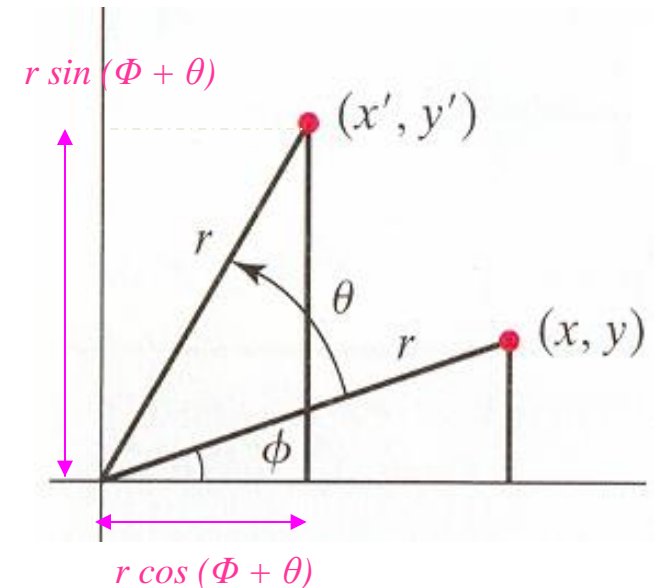
## 2D Rotation around (0,0) (in homogeneous coordinates)

$$x' = r \cos (\Phi + \Theta) = r \cos \Phi \cos \Theta - r \sin \Phi \sin \Theta$$

$$y' = r \sin (\Phi + \Theta) = r \cos \Phi \sin \Theta + r \sin \Phi \cos \Theta$$

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

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$



## 2D Scaling (in homogeneous coordinates)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$(s_x = s_y)$



$(s_x \neq s_y)$

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$$

- Now all basic transformations are applied by multiplication!
- We may concatenate multiple matrices

## Concatenation of two translations

$$\begin{aligned}\mathbf{P}' &= \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}\end{aligned}$$

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

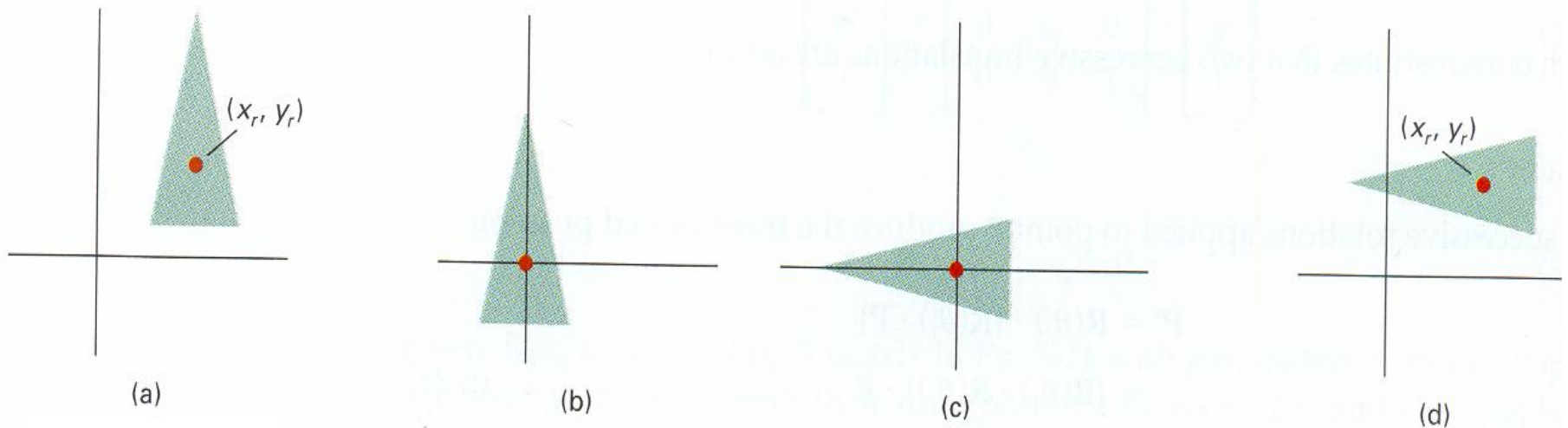


## Concatenation of two scaling transformations

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x}, s_{1y} \cdot s_{2y})$$

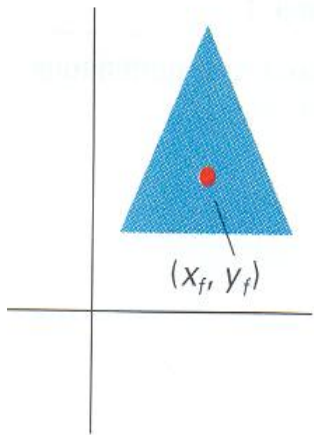
# Arbitrary Rotation (around any point)



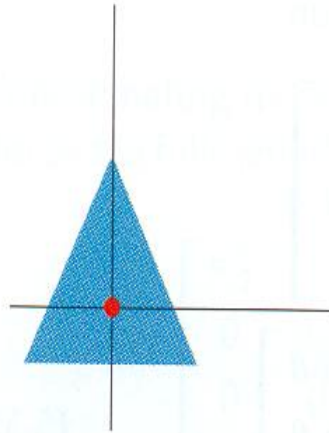
(Hearn & Baker, 2004)

Translation +      Rotation      +      Inverse Translation  
(to the origin)      (around the origin)      (to the initial position)

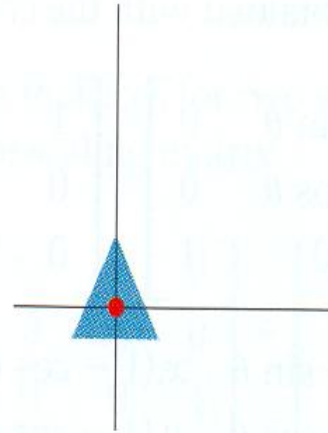
# Arbitrary Scaling



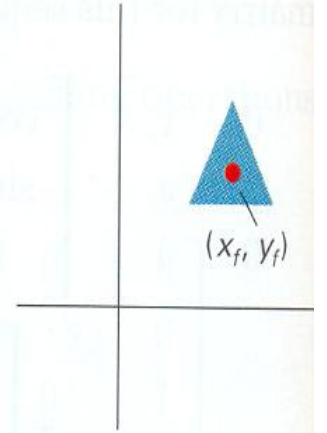
(a)



(b)



(c)

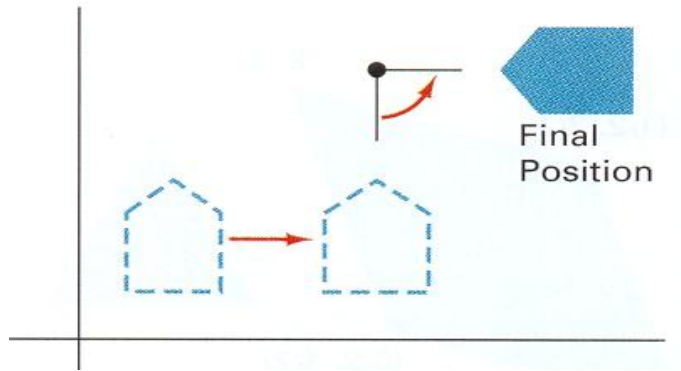


(d)

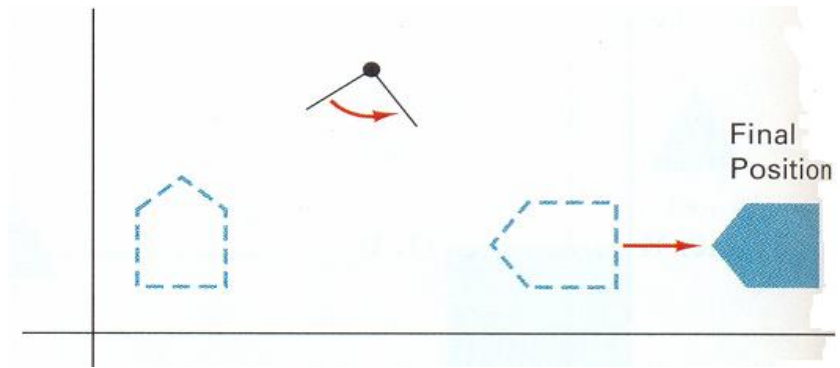
(Hearn & Baker, 2004)

Translation + Scaling + Inverse Translation

# Order is important !



a) Translation + rotation



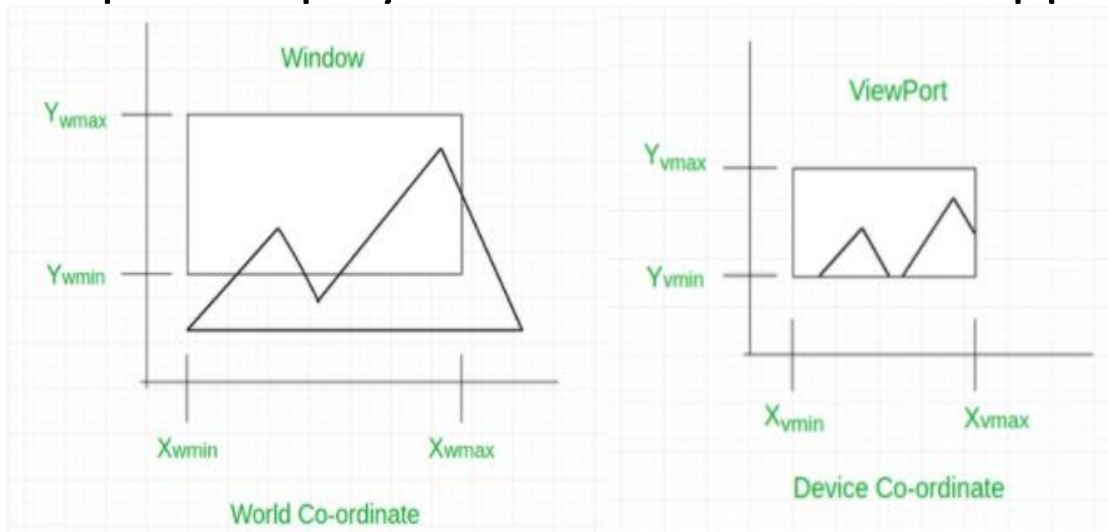
b) Rotation + translation

(Hearn & Baker, 2004)

Results may be different if transformations are applied in a different order!

# The case of viewing 2D scenes

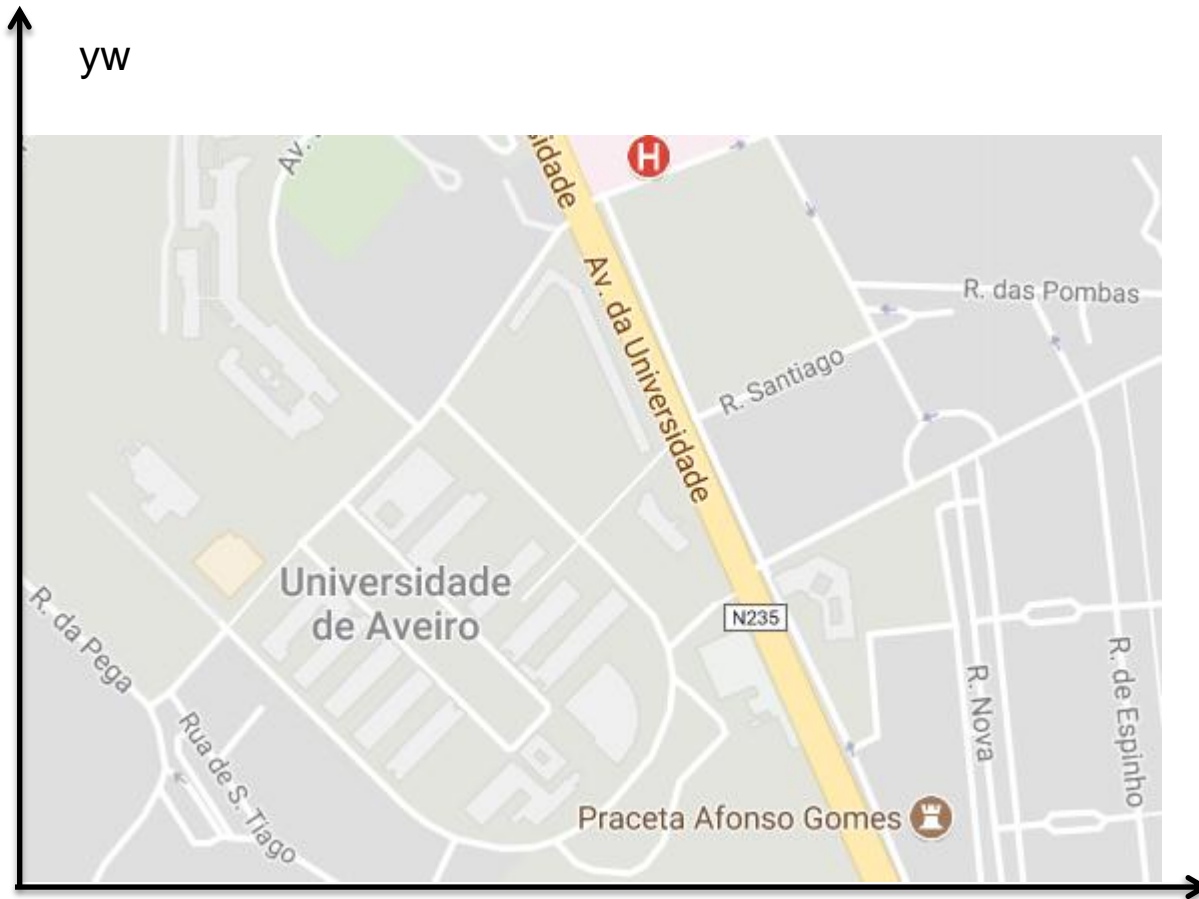
- Define a 2D scene in the **world coordinate system**
- Select a **clipping window** in the XOY plane
  - The window contents will be displayed
- Select a **viewport** in the display
- The viewport displays the contents of the clipping window



Note: Clipping window and viewport are the traditional terms in CG

World ---> display

Clipping Window



World Coordinates

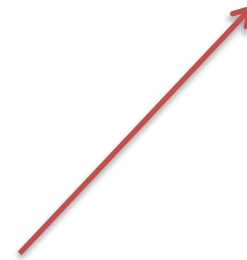
xw

Viewport

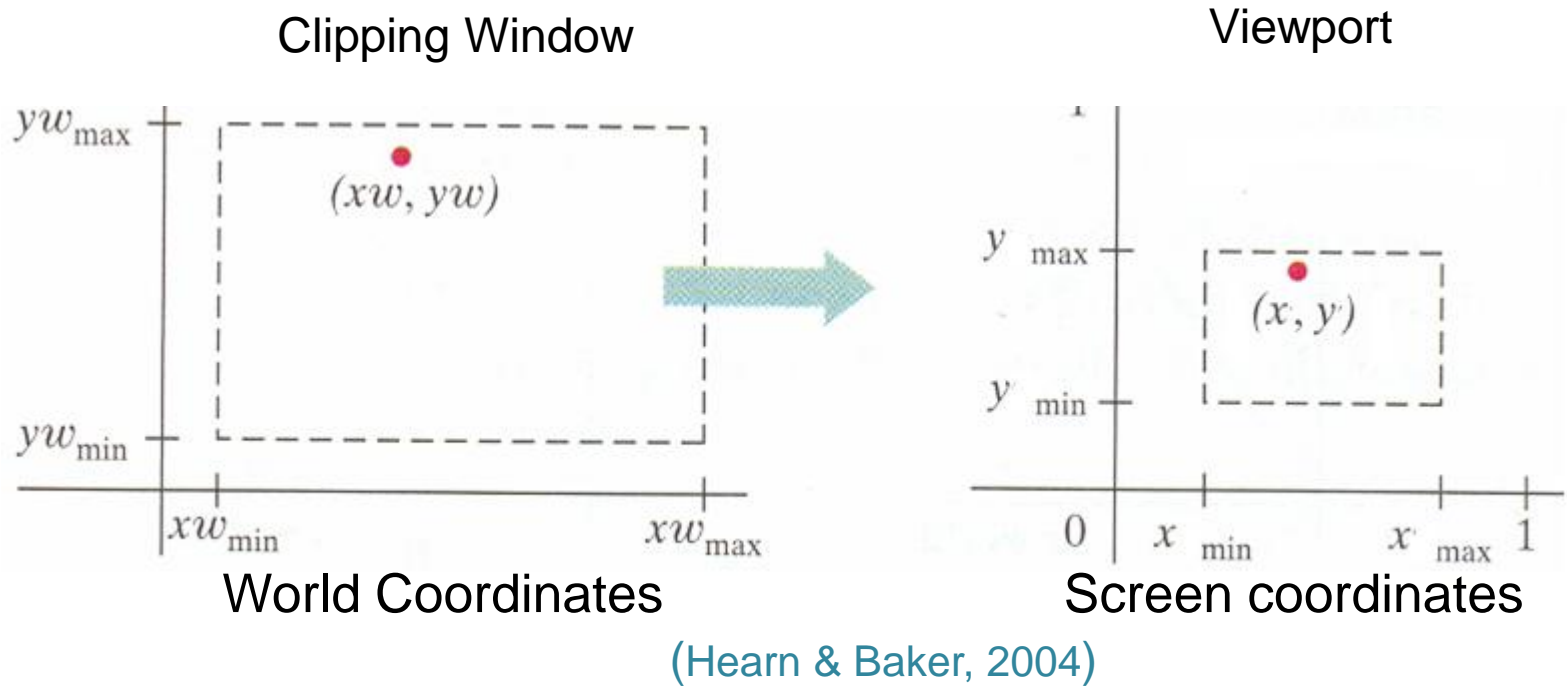


Display Coordinates

x



# Coordinate mapping

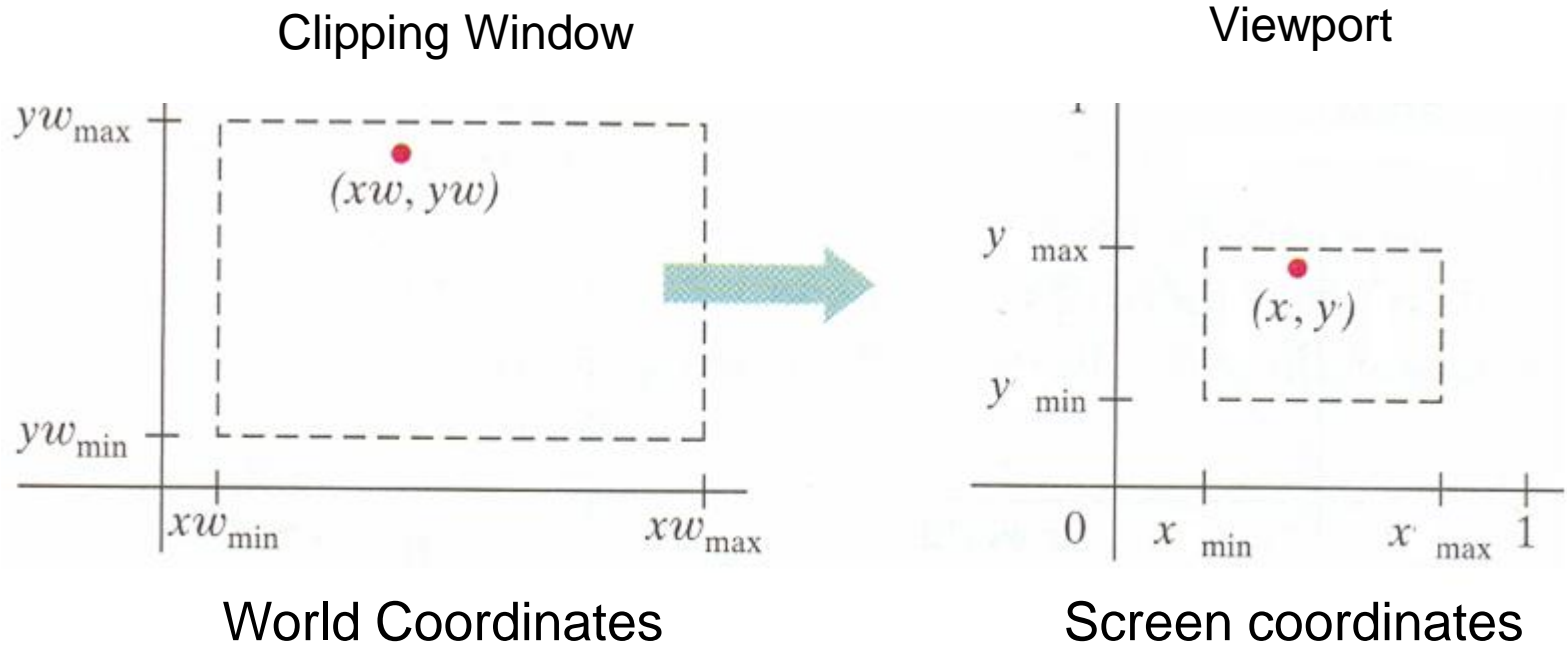


Objects inside the clipping window are mapped to the viewport (the area on the screen where they will be displayed).

## Home work:

Compute  $(x, y)$  given  $(xw, yw)$

# Coordinate mapping



$(x, y)$  given  $(xw, yw)$  :

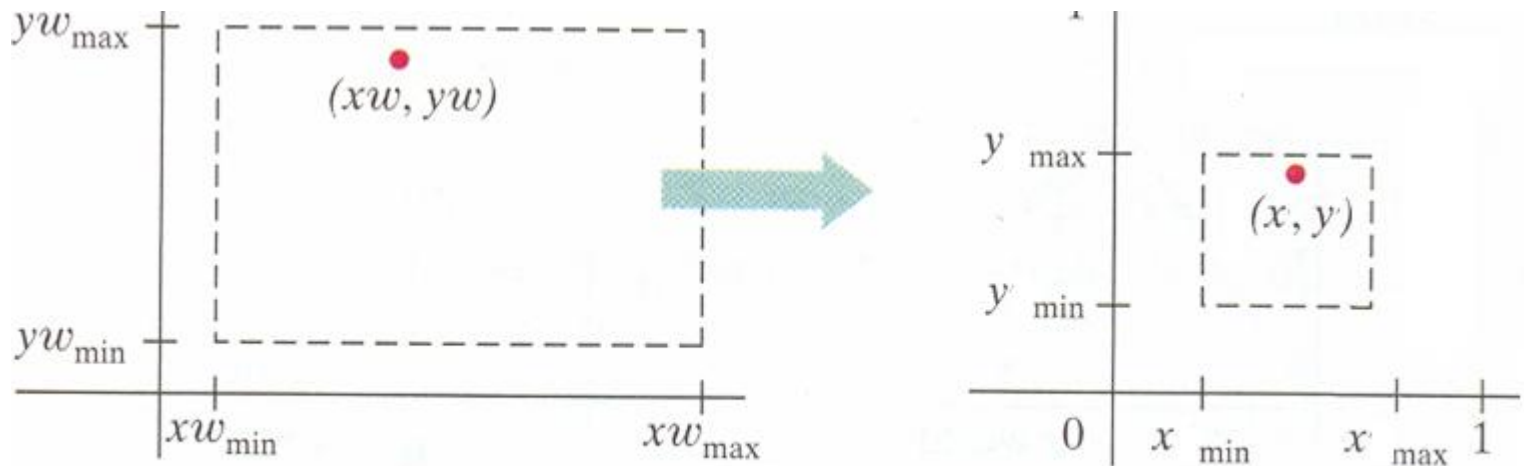
$$\frac{xw - xw_{min}}{xw_{max} - xw_{min}} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

$$\frac{yw - yw_{min}}{yw_{max} - yw_{min}} = \frac{y - y_{min}}{y_{max} - y_{min}}$$



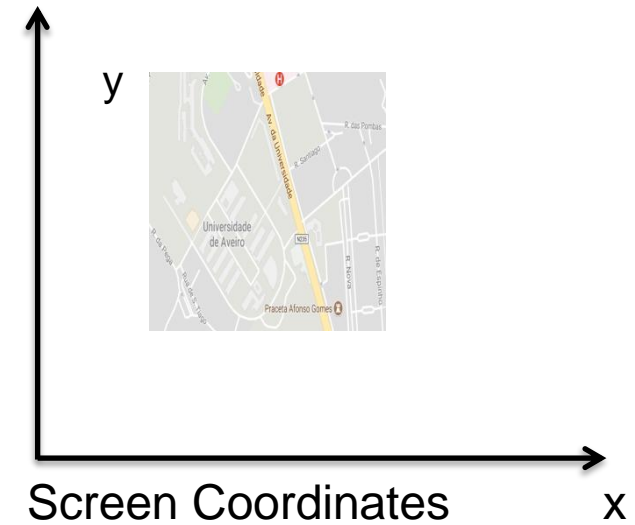
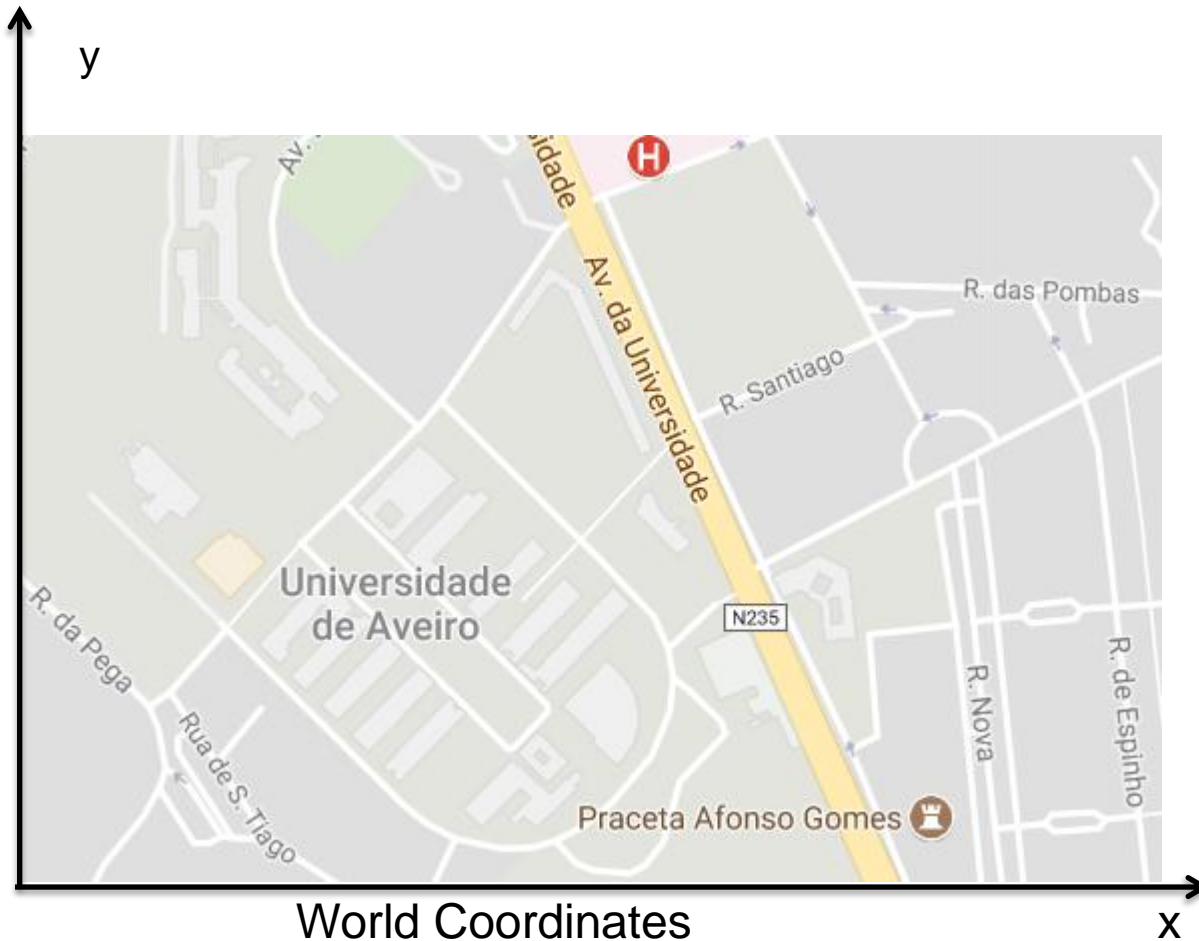
# Coordinate mapping

If the **aspect ratio** is not the same in both situations the result is distortion



(Hearn & Baker, 2004)

World -> screen



The **aspect ratio** is not the same in x and y: distortion!

# 3D Transformations

(in homogeneous coordinates)

- Translation

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

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Scaling

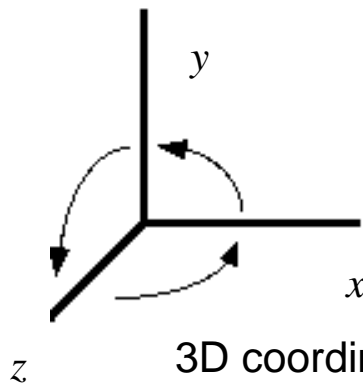
$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

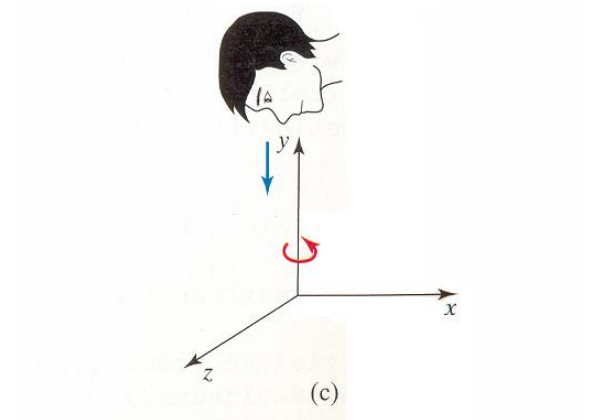
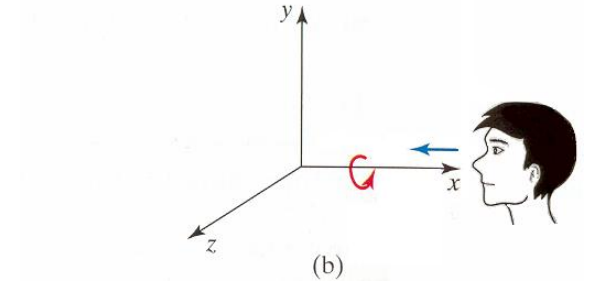
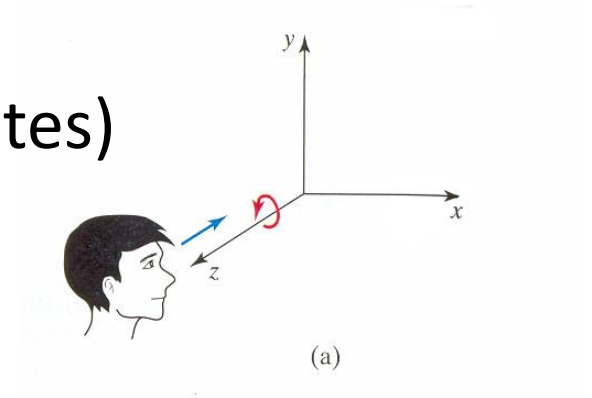
A 3D point in vector notation

# 3D Rotation (in homogeneous coordinates)

- Rotation around each one of the coordinate axis
- Positive rotations are CCW (counter clock wise)!!

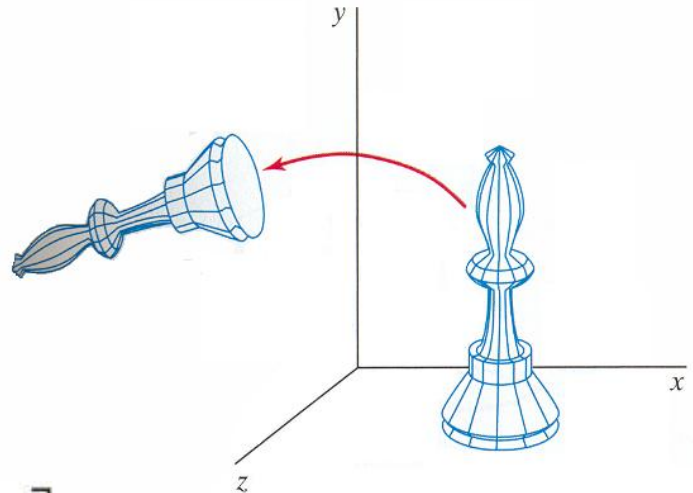


3D coordinate system in CG



(Hearn & Baker, 2004)

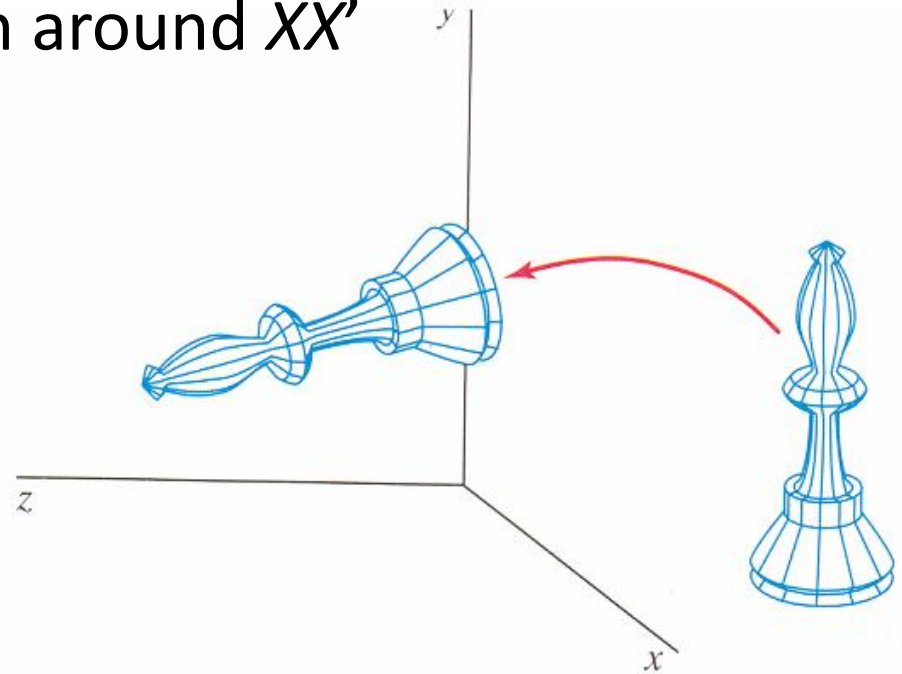
## Rotation around $zz'$



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

The **z coordinate** is  
**maintained**

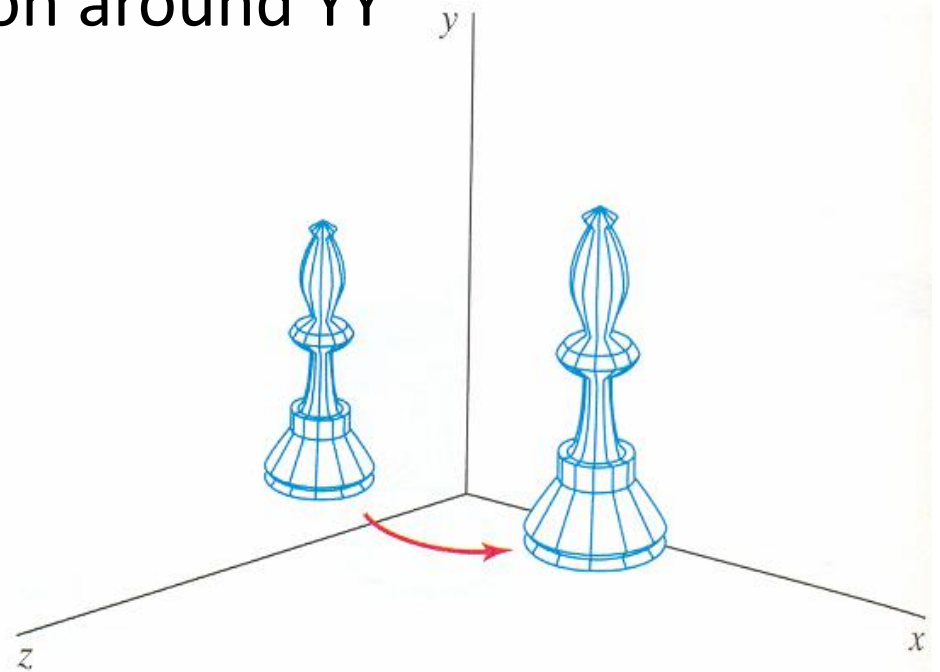
## Rotation around $XX'$



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

(Hearn & Baker, 2004)

## Rotation around YY'



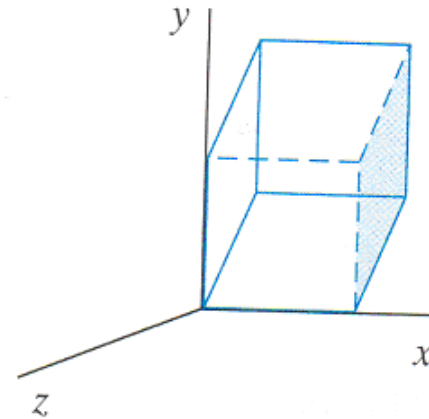
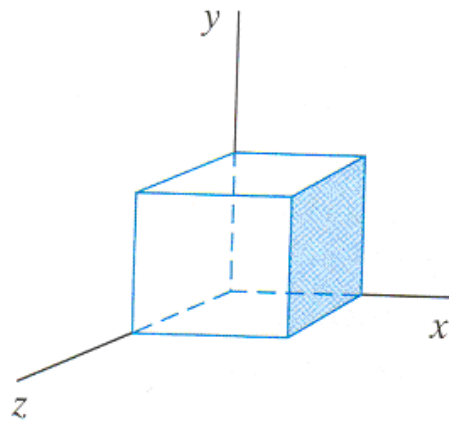
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(Hearn & Baker, 2004)

# Other useful 3D Transformations

- Shears
- Reflections

Shear in  $ZZ'$





# Transformations in three.js

## Matrix4

A class representing a 4x4 [matrix](#).

The most common use of a 4x4 matrix in 3D computer graphics is as a [Transformation Matrix](#). For an introduction to transformation matrices as used in WebGL, check out [this tutorial](#).

This allows a [Vector3](#) representing a point in 3D space to undergo transformations such as translation, rotation, shear, scale, reflection, orthogonal or perspective projection and so on, by being multiplied by the matrix. This is known as *applying* the matrix to the vector.

<https://threejs.org/docs/#api/math/Matrix4>

`.makeTranslation ( x, y, z )`

x - the amount to translate in the X axis.

y - the amount to translate in the Y axis.

z - the amount to translate in the Z axis.

Sets this matrix as a translation transform:

```
1, 0, 0, x,  
0, 1, 0, y,  
0, 0, 1, z,  
0, 0, 0, 1
```

`.makeScale ( x, y, z )`

x - the amount to scale in the X axis.

y - the amount to scale in the Y axis.

z - the amount to scale in the Z axis.

Sets this matrix as scale transform:

```
x, 0, 0, 0,  
0, y, 0, 0,  
0, 0, z, 0,  
0, 0, 0, 1
```

`.makeRotationX ( theta )`

[theta](#) — Rotation angle in radians.

Sets this matrix as a rotational transformation around the X axis by [theta](#) ( $\theta$ ) radians. The resulting matrix will be:

```
1 0 0 0
0 cos(θ) -sin(θ) 0
0 sin(θ) cos(θ) 0
0 0 0 1
```

`.makeRotationY ( theta )`

[theta](#) — Rotation angle in radians.

Sets this matrix as a rotational transformation around the Y axis by [theta](#) ( $\theta$ ) radians. The resulting matrix will be:

```
cos(θ) 0 sin(θ) 0
0 1 0 0
-sin(θ) 0 cos(θ) 0
0 0 0 1
```

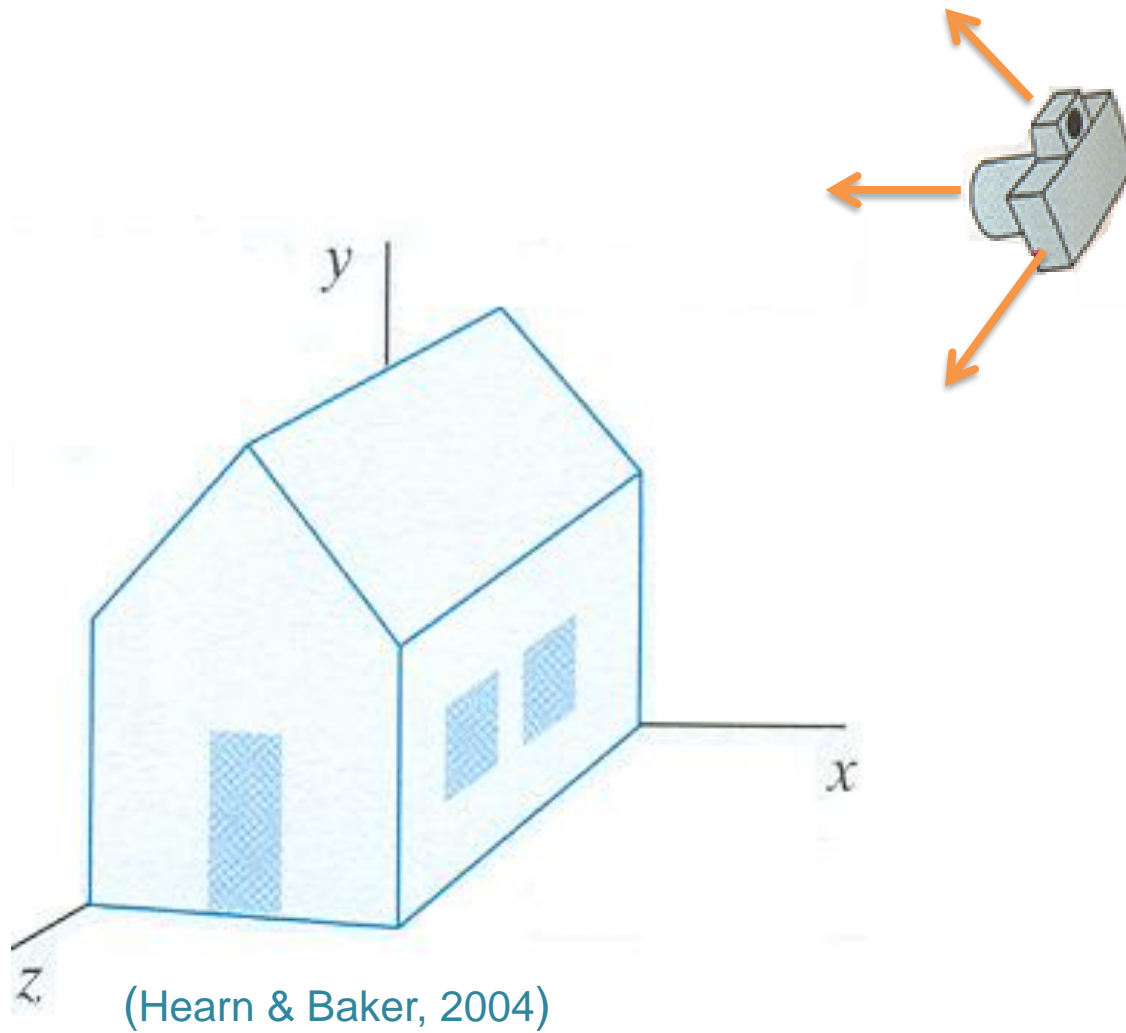
`.makeRotationZ ( theta )`

[theta](#) — Rotation angle in radians.

Sets this matrix as a rotational transformation around the Z axis by [theta](#) ( $\theta$ ) radians. The resulting matrix will be:

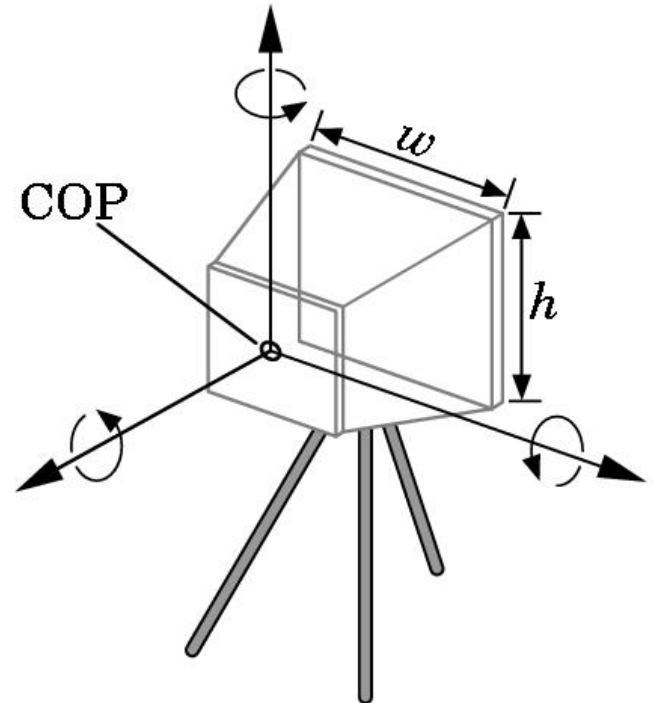
```
cos(θ) -sin(θ) 0 0
sin(θ) cos(θ) 0 0
0 0 1 0
0 0 0 1
```

# 3D Viewing



# Camera specification

- Position and orientation
- Lens
- Image size
- Orientation of image plane

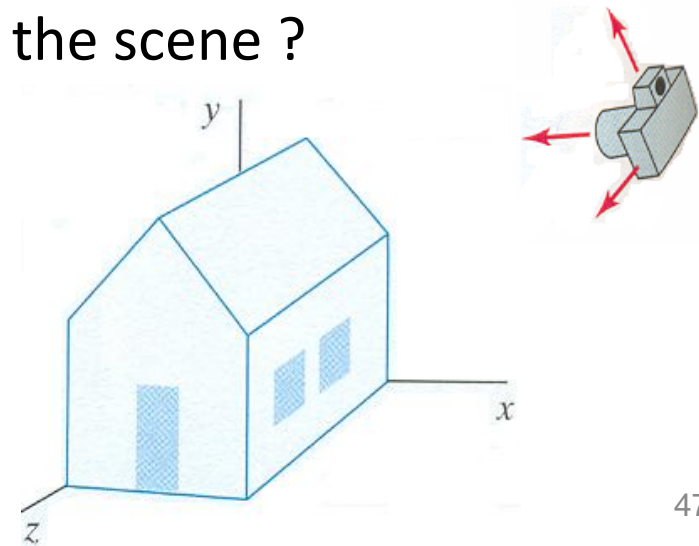


(Angel, 2012)


# 3D Viewing

## (2D representation of a 3D scene)

- Where is the observer / the camera ?
  - **Position** ?
  - Close to the 3D scene ?
  - Far away ?
- How is the camera/observer looking at the scene ?
  - **Orientation** ?
- How to represent as a 2D image ?
  - **Projection** ?



# 3D visualization pipeline

- Instantiate **models of the scene**
  - Position, orientation, size
- Establish **viewing parameters**
  - Camera position and orientation
- Compute **illumination** and **shade polygons**
- Perform clipping
- Project into 2D 
- Rasterize

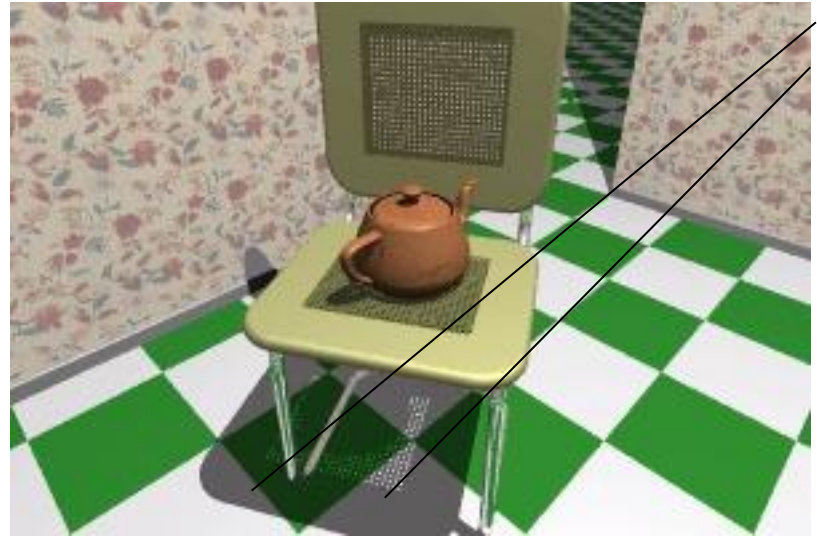
# Projection (from 3D to 2D)

2D representations of a 3D scene may be done in different ways (in CG geometric planar projections are used: straight projectors/planar viewing plane)



Parallel Projection

(allows measures)



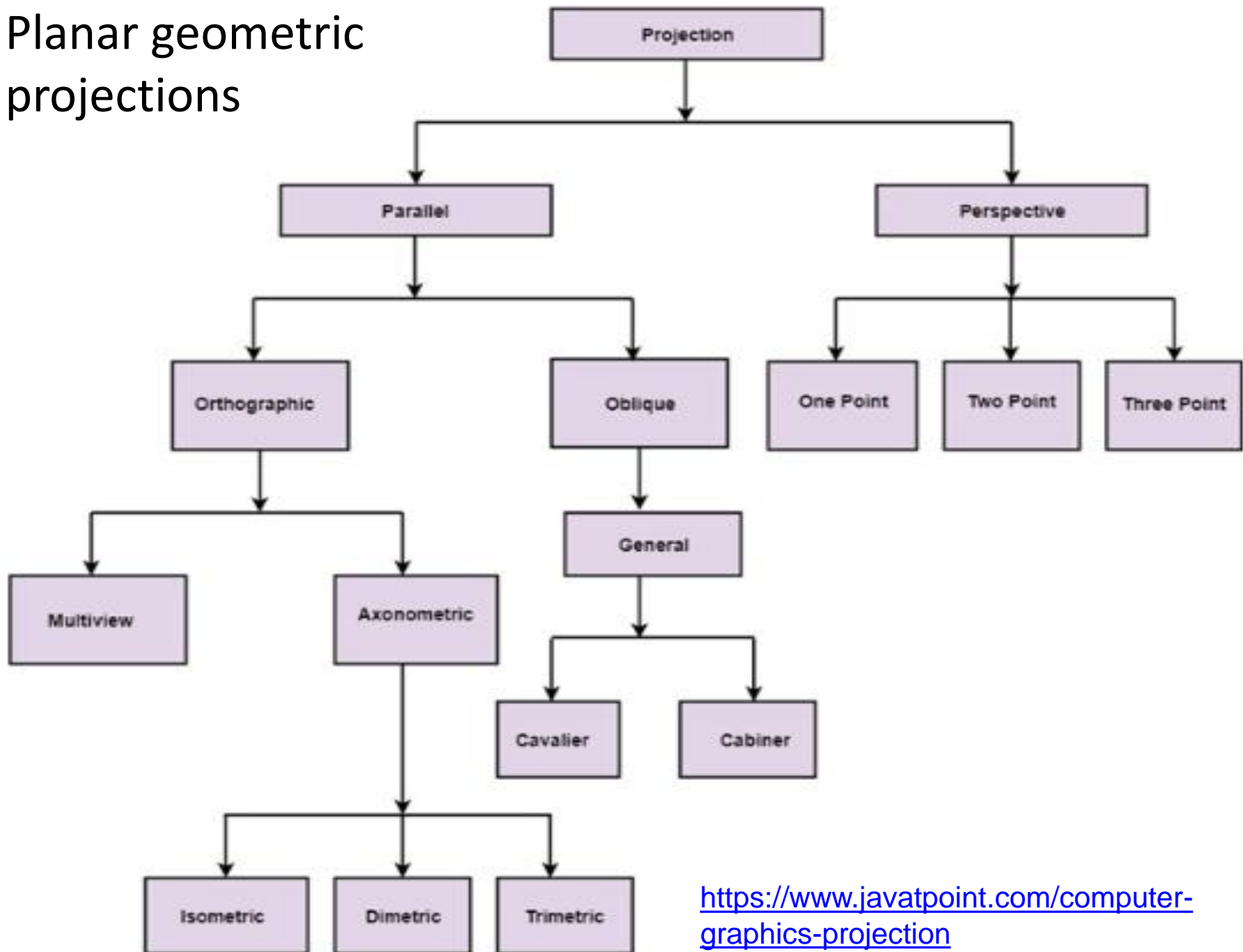
Perspective Projection

(more realistic images)

<https://www.britannica.com/science/projection-geometry>

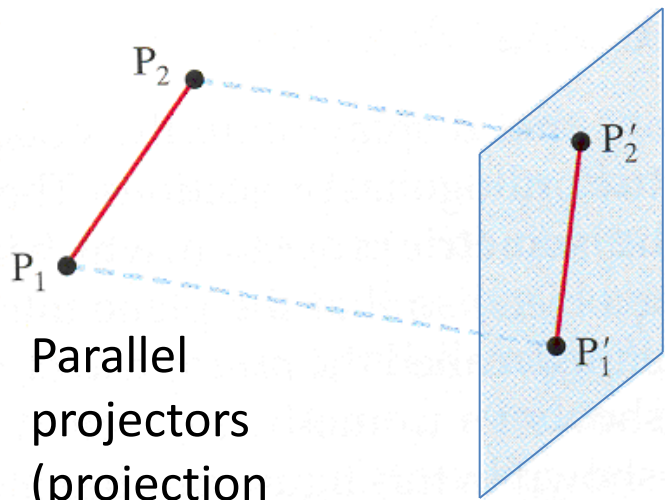


# Planar geometric projections

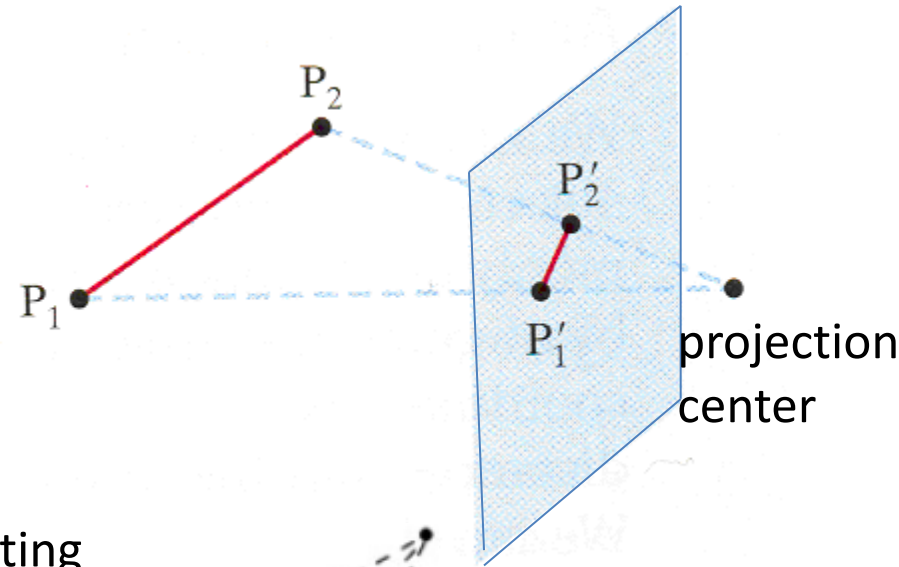


<https://www.javatpoint.com/computer-graphics-projection>

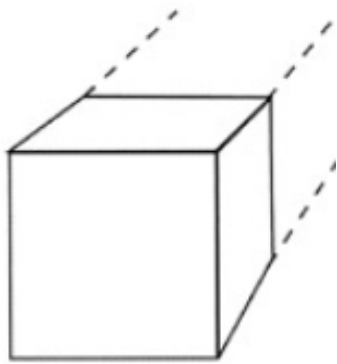
# Projections



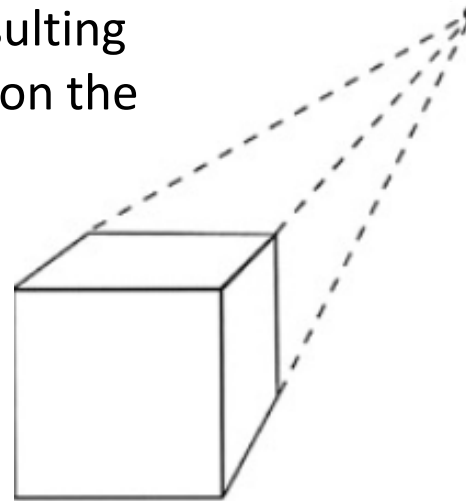
Parallel  
projectors  
(projection  
center at  
infinity)



Examples of resulting  
representation on the  
viewing plane



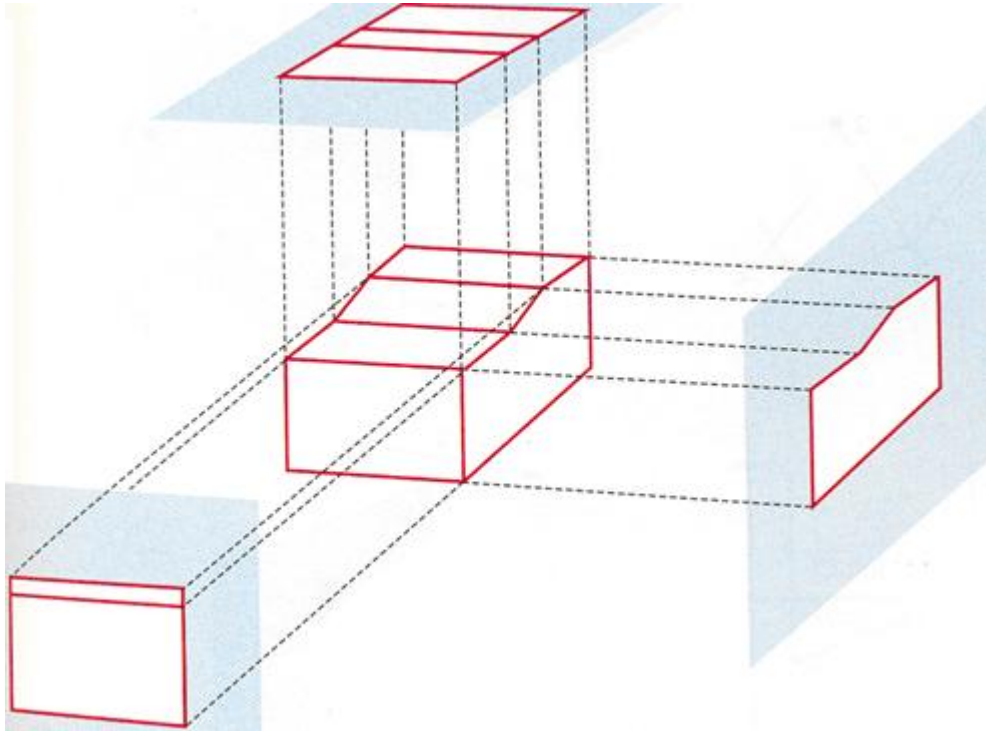
Parallel Projection



Perspective Projection

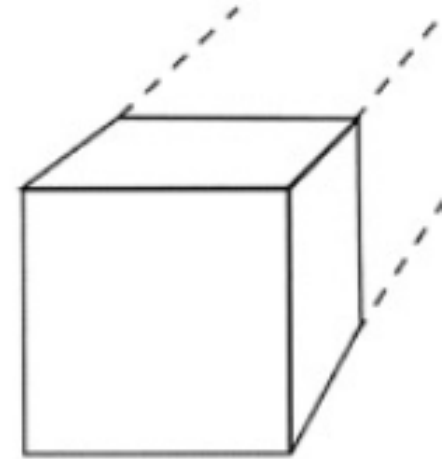
(Hearn & Baker, 2004)

# Parallel Projections



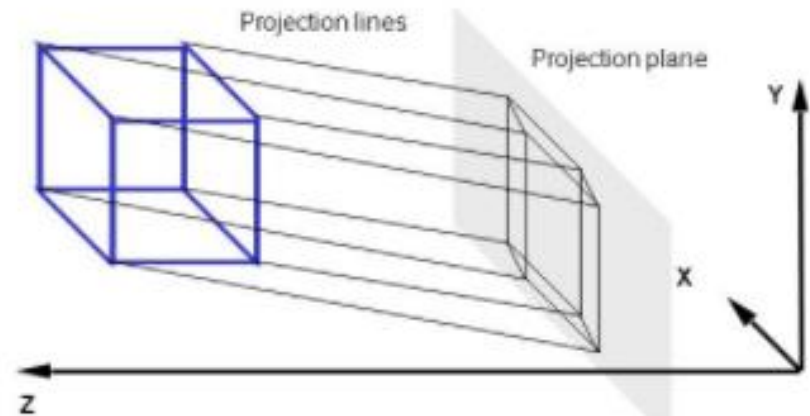
Orthographic/ Multiview projection

(Hearn & Baker, 2004)



Orthographic /  
Axonometric projection

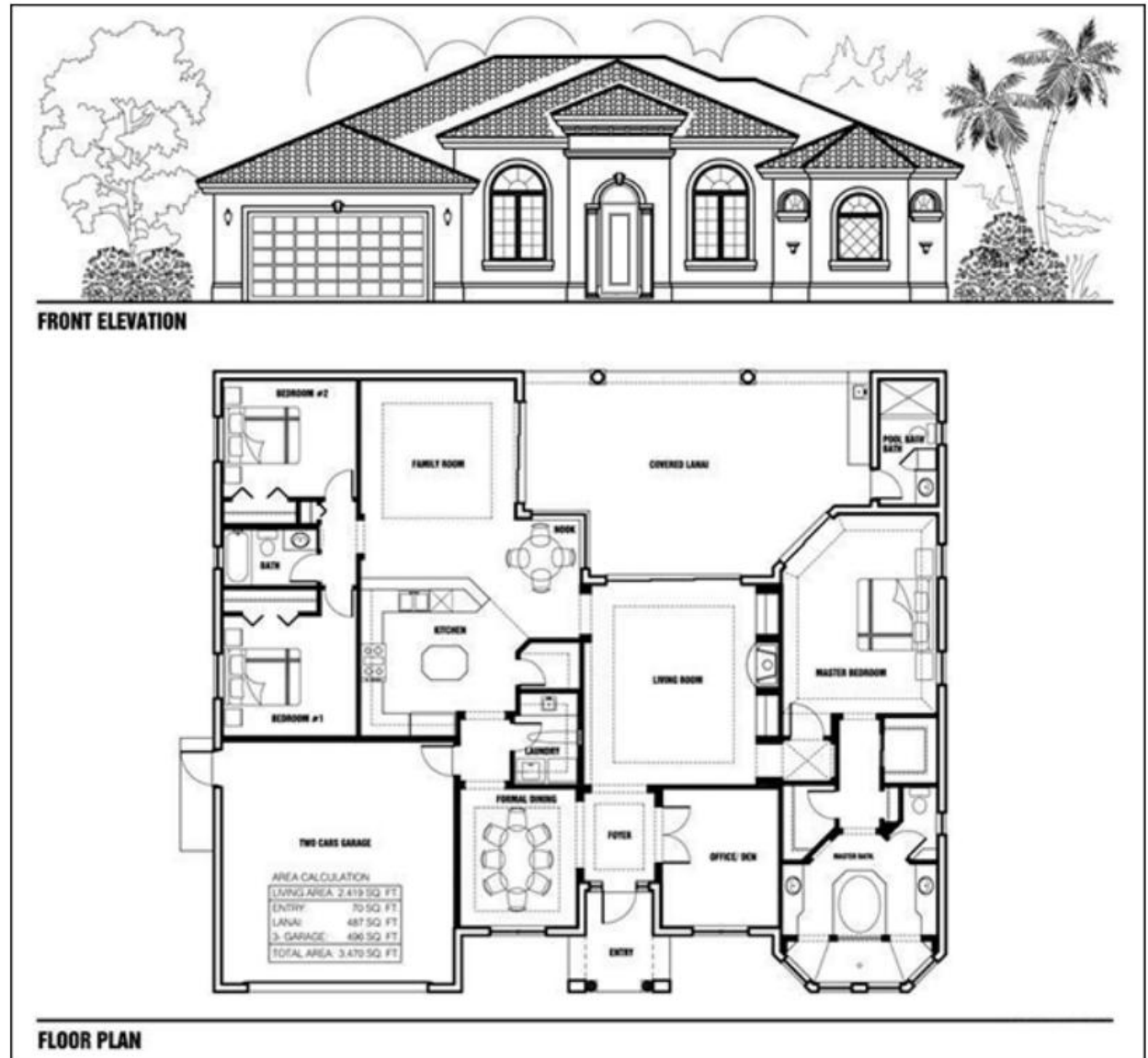
Oblique projection



Parallel projections are widely used in architecture

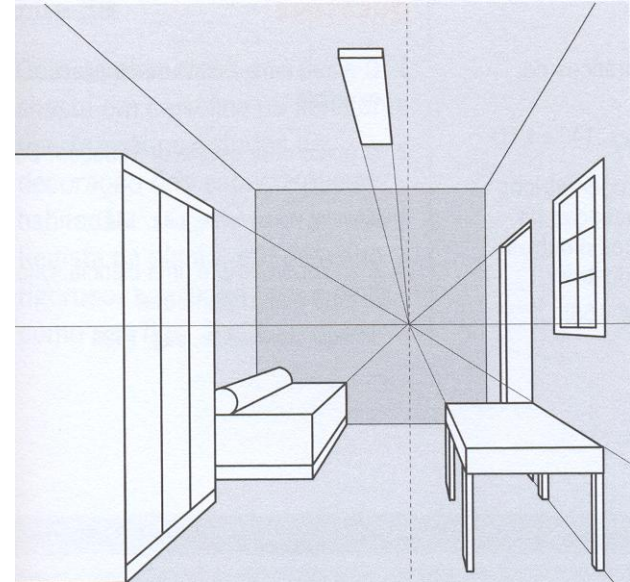
- Front elevation
- Floor plan

Allowing distance  
measures



# Perspective Projections

**Foreshortening** indicates a perspective projection



Approximate representation as a scene is seen by the eye

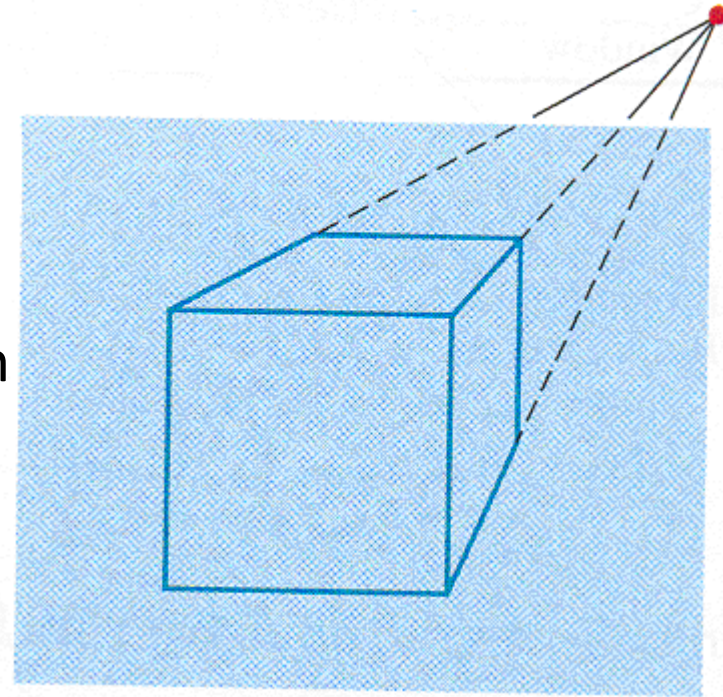
Object's dimensions along the line of sight appear shorter than its dimensions across the line of sight

[https://en.wikipedia.org/wiki/Perspective\\_\(graphical\)](https://en.wikipedia.org/wiki/Perspective_(graphical))

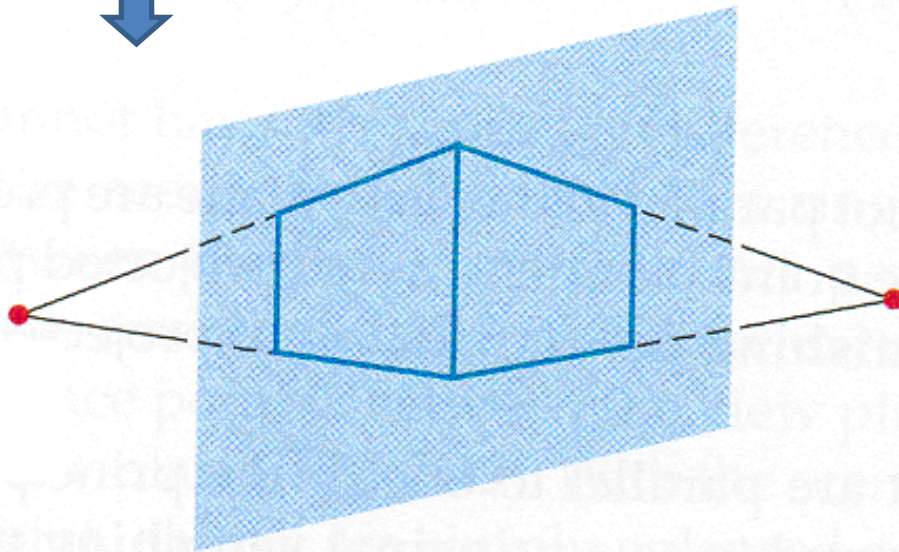


# Perspective Projections

One vanishing point perspective projection

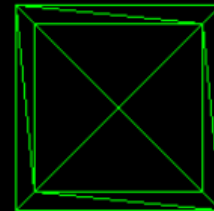
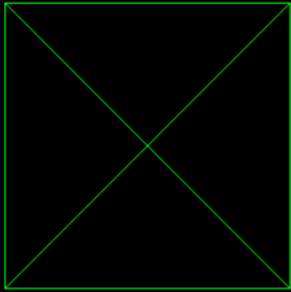


Two vanishing points perspective projection



(Hearn & Baker, 2004)

# Orthographic vs perspective camera in Three.js



# How to represent/apply projections?

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



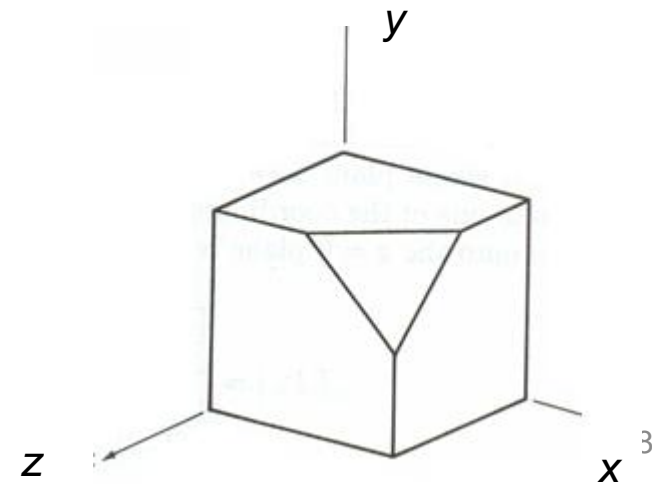
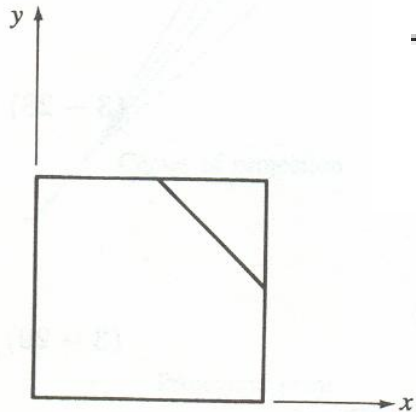
# How to apply Projections?

- Also by matrix multiplication

Example: Matrix of the orthographic projection on the  $xy$  plane in homogeneous coordinates:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$z$  coordinates are discarded



## Example of a more complex projection matrix

Compute the matrix to obtain a view allowing to make measures on the triangle face of the object.

The vertices of the face are:

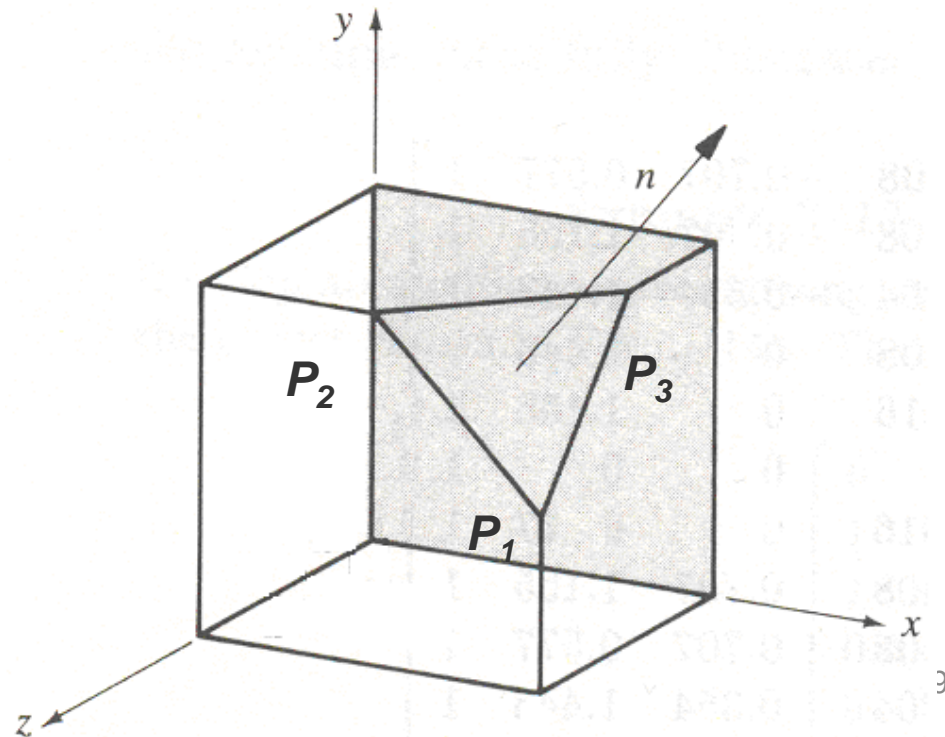
$$P_1 = (1, 0.5, 1)$$

$$P_2 = (0.5, 1, 1)$$

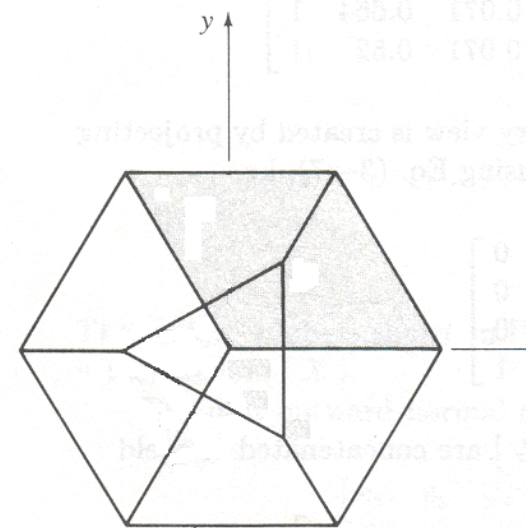
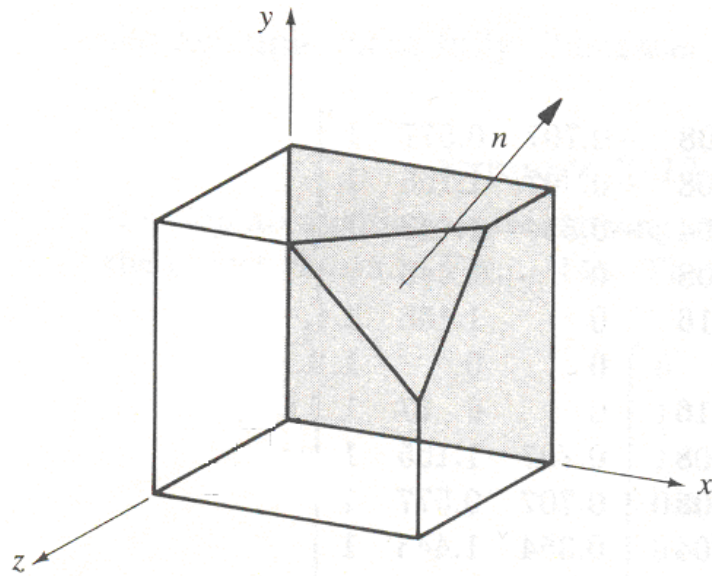
$$P_3 = (1, 1, 0.5)$$

Hint:

- What is the type of projection that allows to make measures?
- Find the rotations needed ...



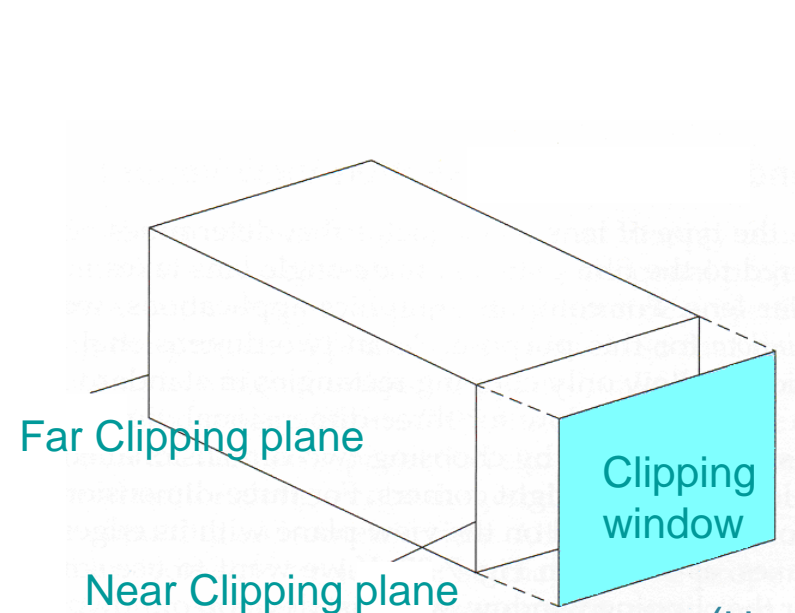
Resulting matrix:



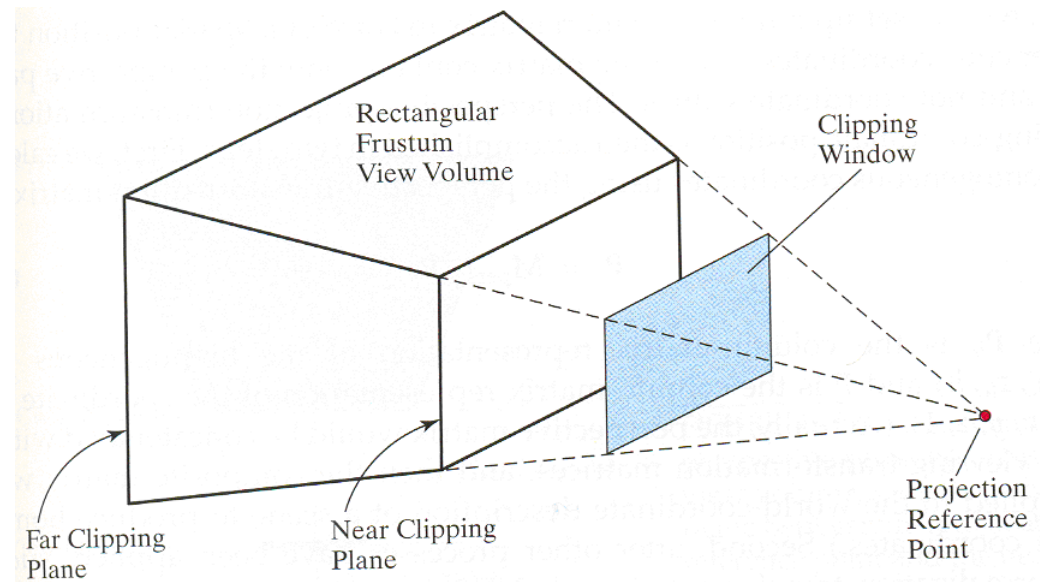
$$\mathbf{M}_{\text{final}} = \begin{bmatrix} 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# How to limit what is observed and represented ?

- Clipping window on the projection plane
- View volume (frustum) in 3D



Parallel projection



(Hearn & Baker, 2004)

Perspective projection

# Examples using Three.js

three.js <sup>r87</sup>

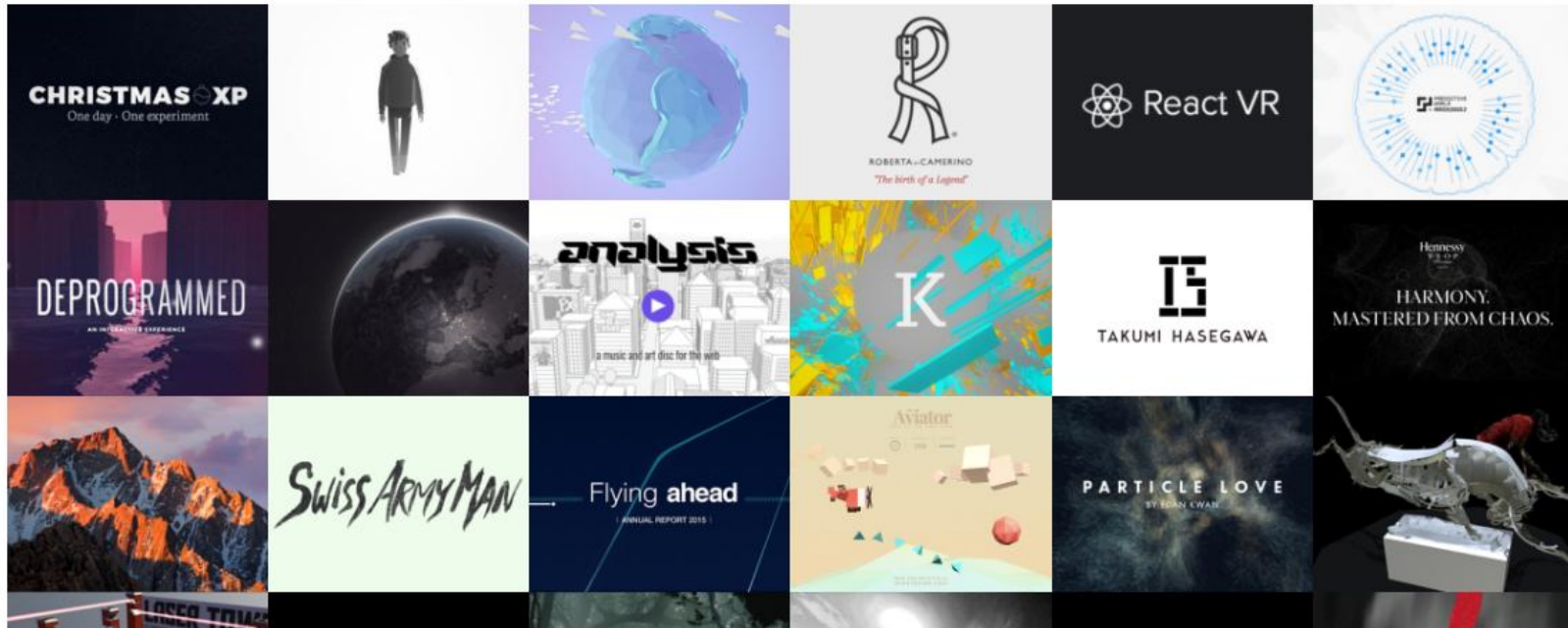
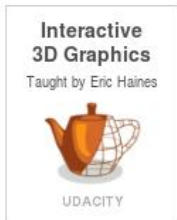
featured projects

[submit project](#)

[documentation](#)  
[examples](#)

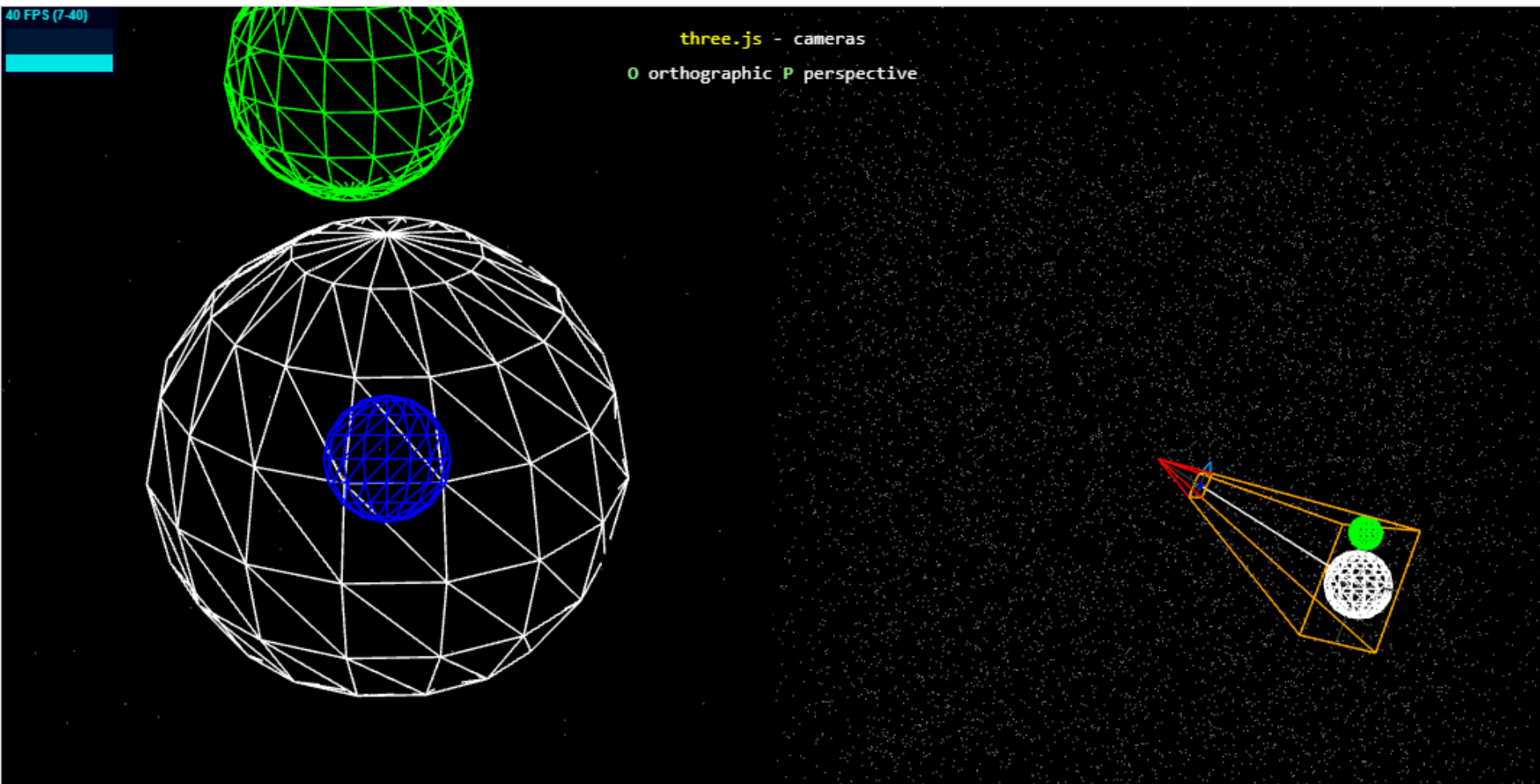
[download](#)

[source code](#)  
[questions](#)  
[forum](#)  
[irc](#)  
[slack](#)  
[google+](#)  
[editor](#)



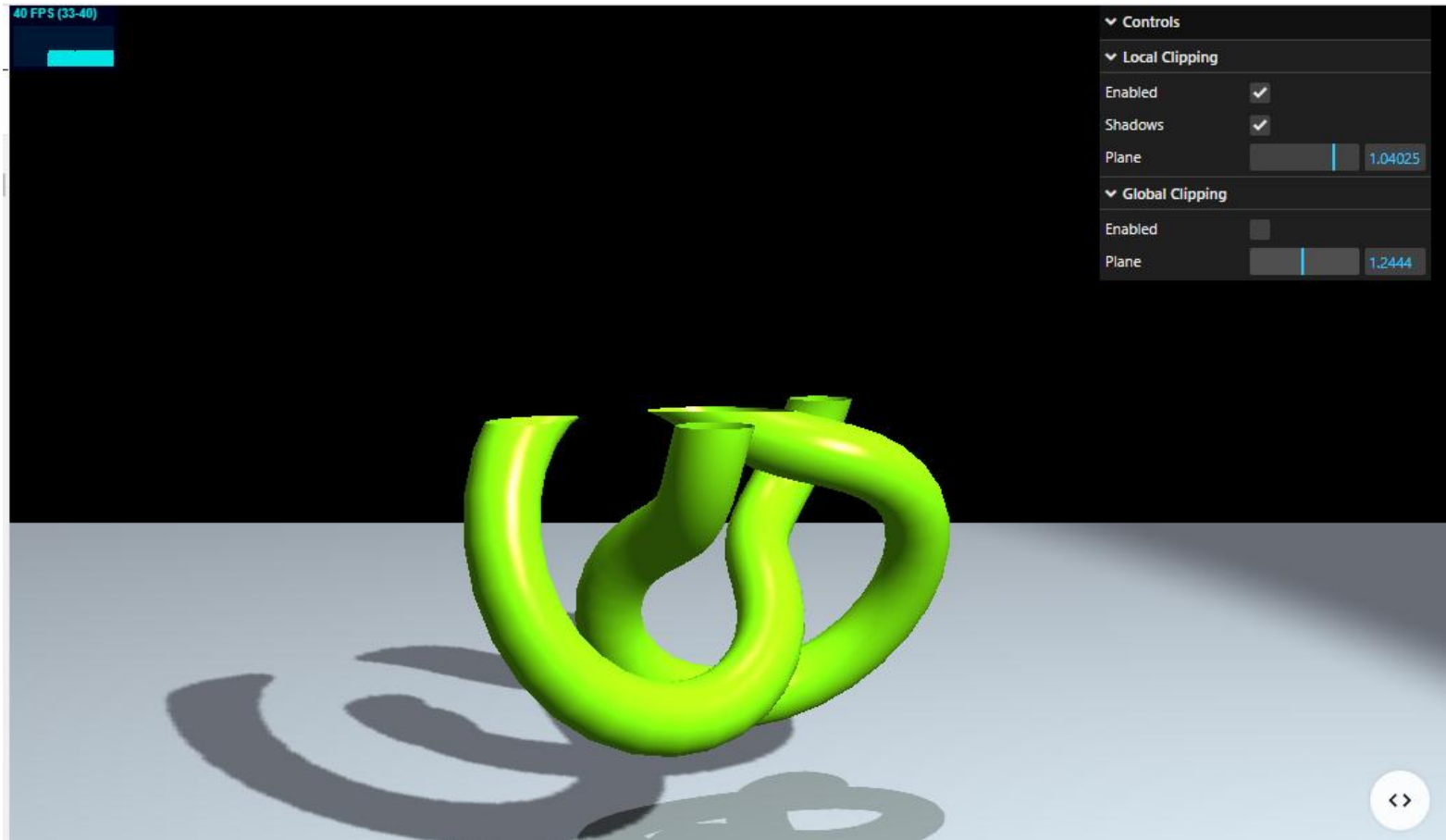
<https://threejs.org/>

# Projections



[https://threejs.org/examples/#webgl\\_camera](https://threejs.org/examples/#webgl_camera)

# Clipping (and shadows)

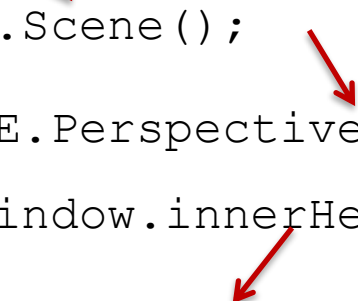


[https://threejs.org/examples/#webgl\\_clipping](https://threejs.org/examples/#webgl_clipping)

# Thee.js first example

## 1. Defining the scene, the camera and where the scene is rendered

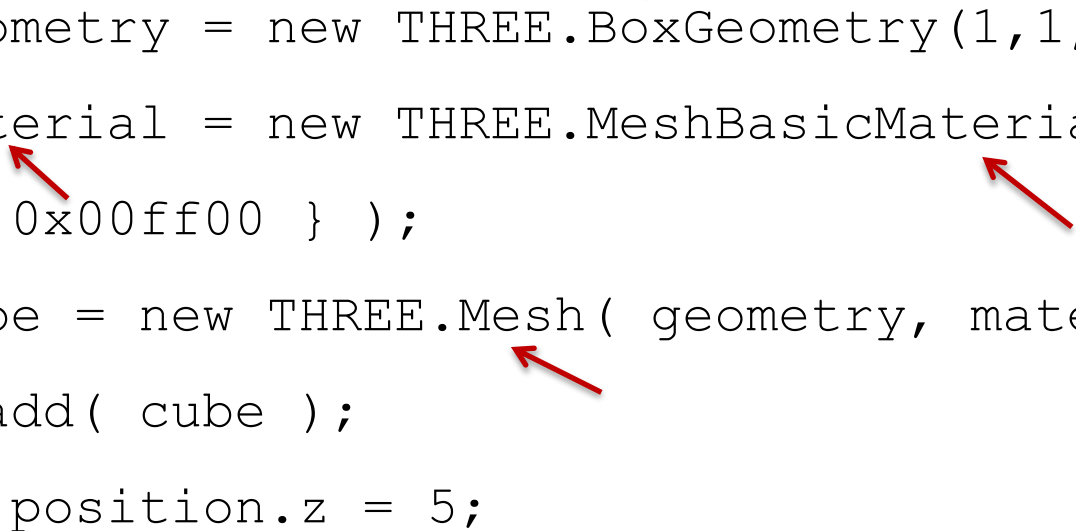
```
var scene = new THREE.Scene();  
var camera = new THREE.PerspectiveCamera( 75,  
window.innerWidth / window.innerHeight, 0.1, 1000 );  
var renderer = new THREE.WebGLRenderer();  
renderer.setSize( window.innerWidth, window.innerHeight );  
document.body.appendChild( renderer.domElement );
```






## 2.Creating an object and camera position

```
var geometry = new THREE.BoxGeometry(1,1,1);  
var material = new THREE.MeshBasicMaterial( {  
color: 0x00ff00 } );  
var cube = new THREE.Mesh( geometry, material );  
scene.add( cube );  
camera.position.z = 5;
```




### 3. Scene rendering

```
function render() {  
  requestAnimationFrame(render);  
  renderer.render(scene, camera);  
}
```



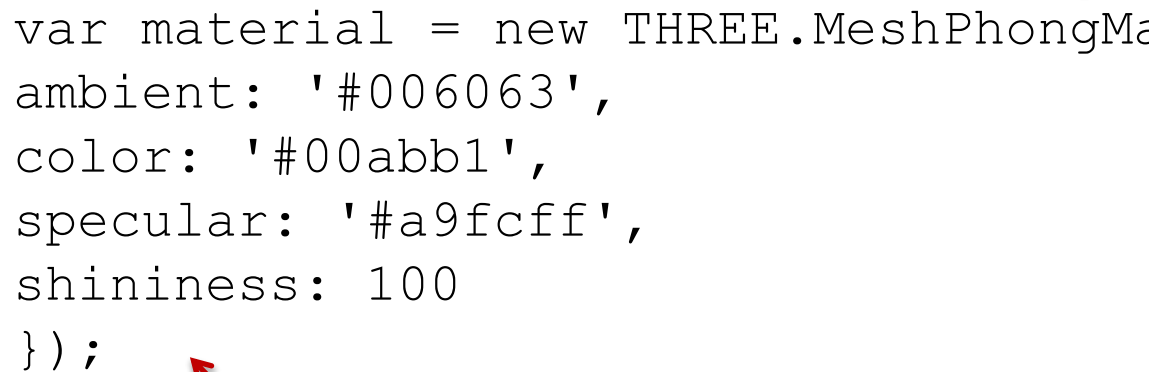
### 4. Scene animation

```
render();  
cube.rotation.x += 0.1;  
cube.rotation.y += 0.1;
```



## Adding lights and shading

```
var material = new THREE.MeshPhongMaterial({  
  ambient: '#006063',  
  color: '#00abb1',  
  specular: '#a9fcff',  
  shininess: 100  
});
```



## Some reference books

- S. Marschner, P. Shirley, *Fundamentals of Computer Graphics*, 5<sup>th</sup> ed., A K Peters/CRC Press, 2021  
[Fundamentals of Computer Graphics, 5th Edition \(oreilly.com\)](https://oreil.ly/fundamentals-of-computer-graphics-5th-edition)
- D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3<sup>rd</sup> Ed., Addison-Wesley, 2004
- E. Angel and D. Shreiner, *Introduction to Computer Graphics*, 6<sup>th</sup> Ed., Pearson Education, 2012
- Hughes, J., A. Van Dam, et al., *Computer Graphics, Principles and Practice*, 3rd Ed., Addison Wesley, 2013  
[Hughes/Computer Graphics, 3/E \(oreilly.com\)](https://oreil.ly/computer-graphics-3e)