Universidade de Aveiro
Departamento de Electrónica,
Telecomunicações e Informática

universidade de aveiro
1973 . 2018

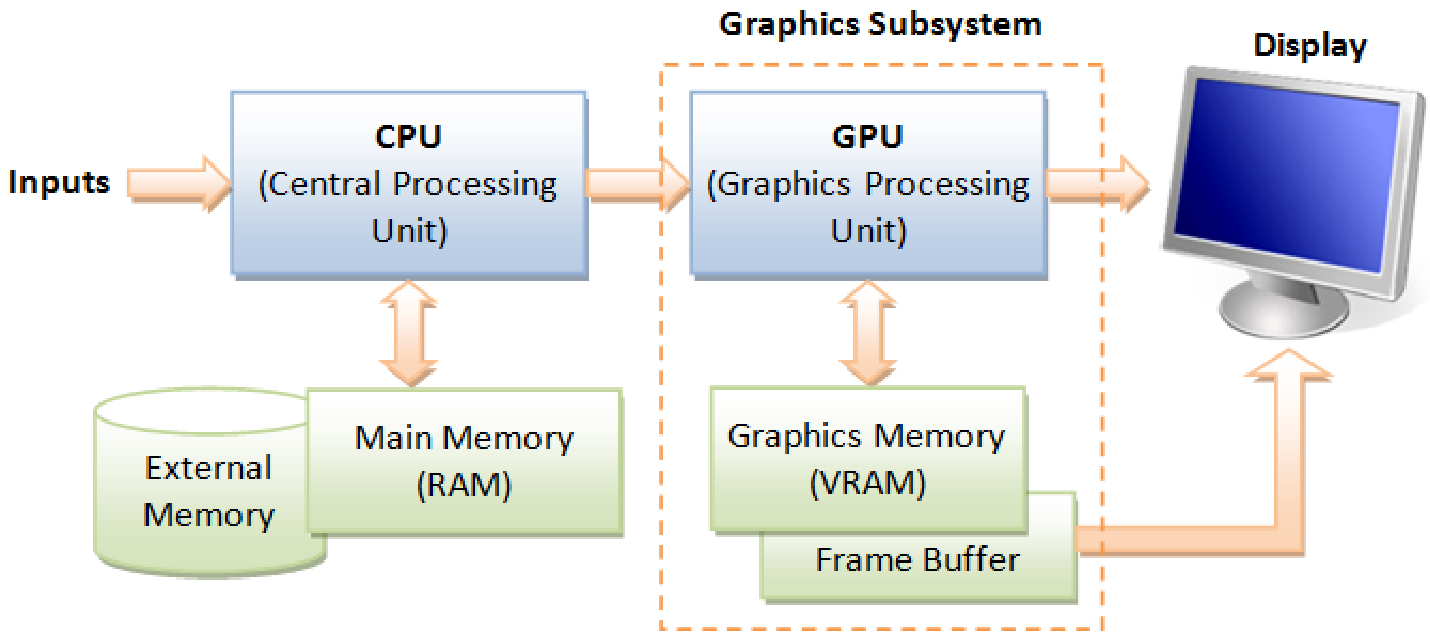
Introduction to Computer Graphics
main concepts and methods

(Wikipedia)

Beatriz Sousa Santos

University of Aveiro, 2019

Basic Graphics System



https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Topics

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

CG Main Tasks

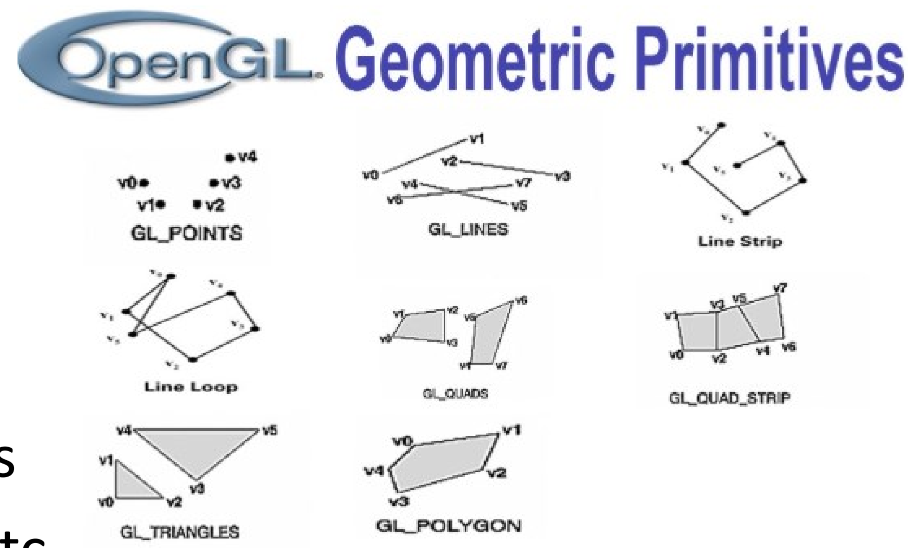
- Modeling
  - Construct individual models / objects
  - Assemble them into a 2D or 3D scene
- Animation
  - Static vs. dynamic scenes
  - Movement and / or deformation
- Rendering
  - Generate final images
  - Where is the observer?
  - How is he / she looking at the scene?

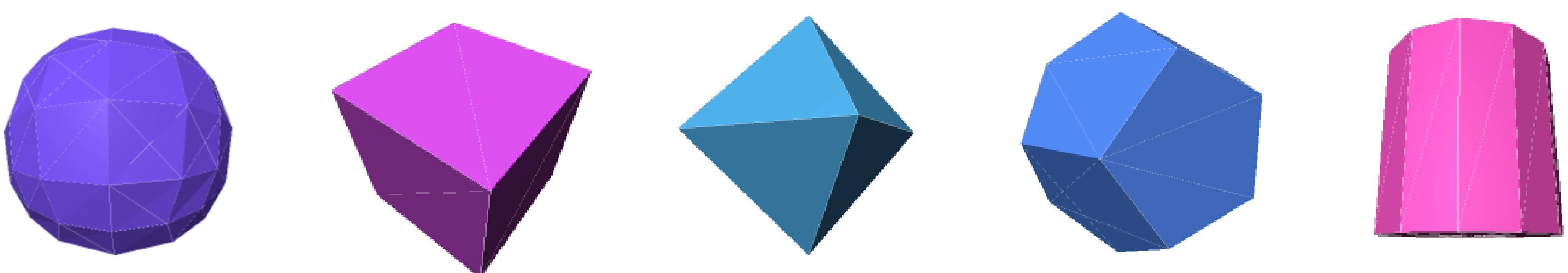
Geometric Primitives

- Simple primitives
  - Points
  - Line segments
  - Polygons

- Geometric primitives
  - Parametric curves / surfaces
  - Cubes, spheres, cylinders, etc.

Examples:





https://threejsfundamentals.org/threejs/lessons/threejs-primitives.html

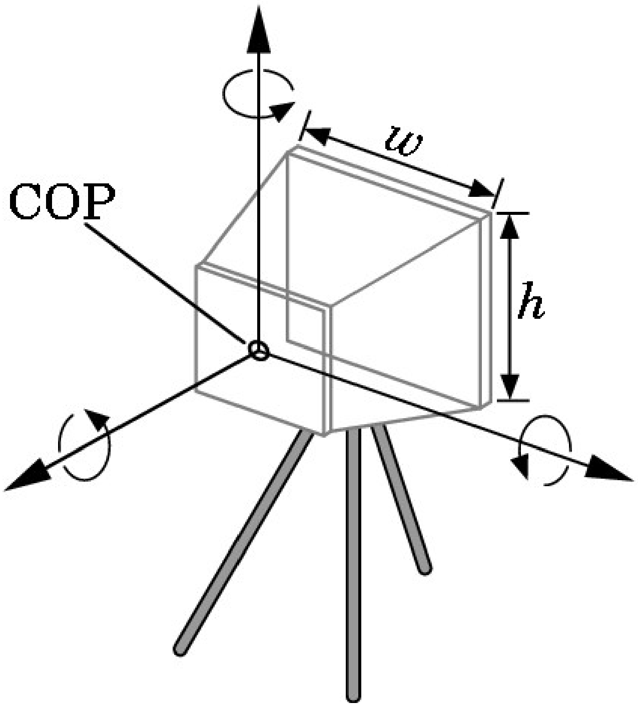
Lights and materials

- Types of light sources
  - Point vs distributed light sources
  - Spot lights
  - Near and far sources
  - Color properties

- Material properties
  - Absorption: color properties
  - Scattering: diffuse and specular
  - Transparency

Camera specification

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

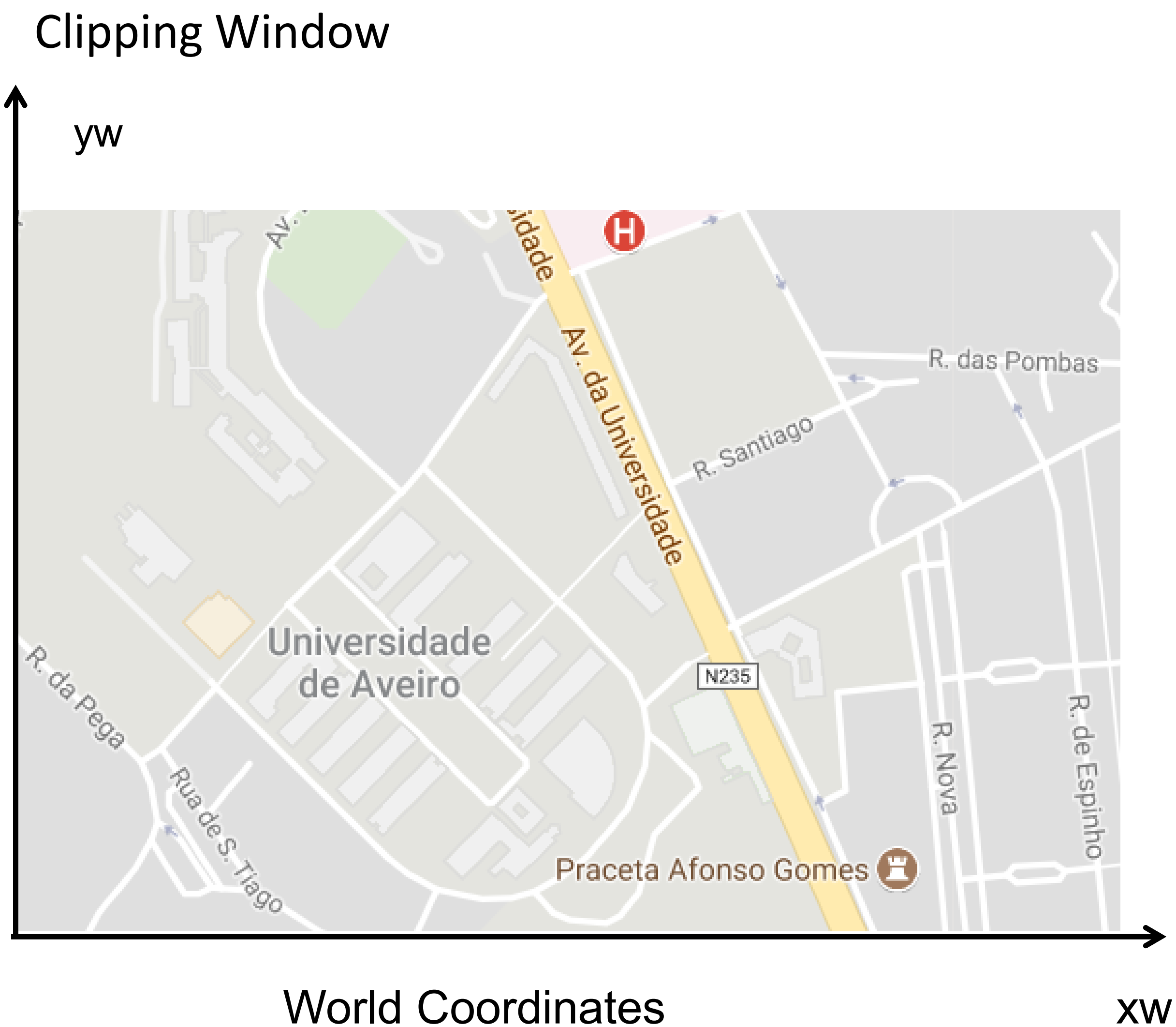


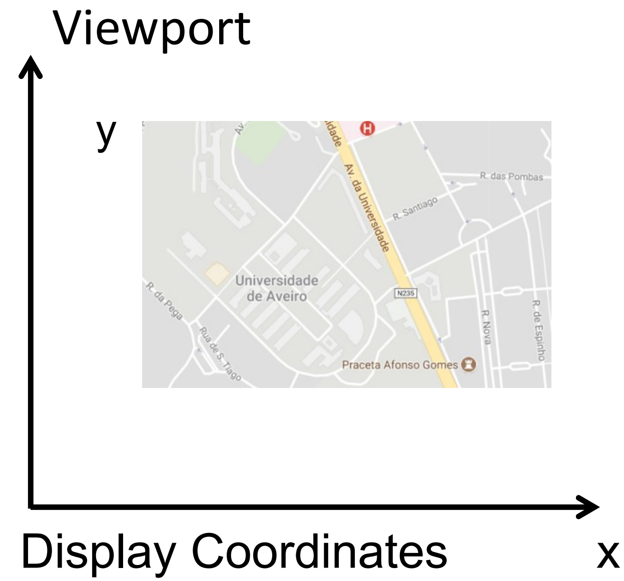
(Angel, 2012)

2D Visualization

- 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

World -> display

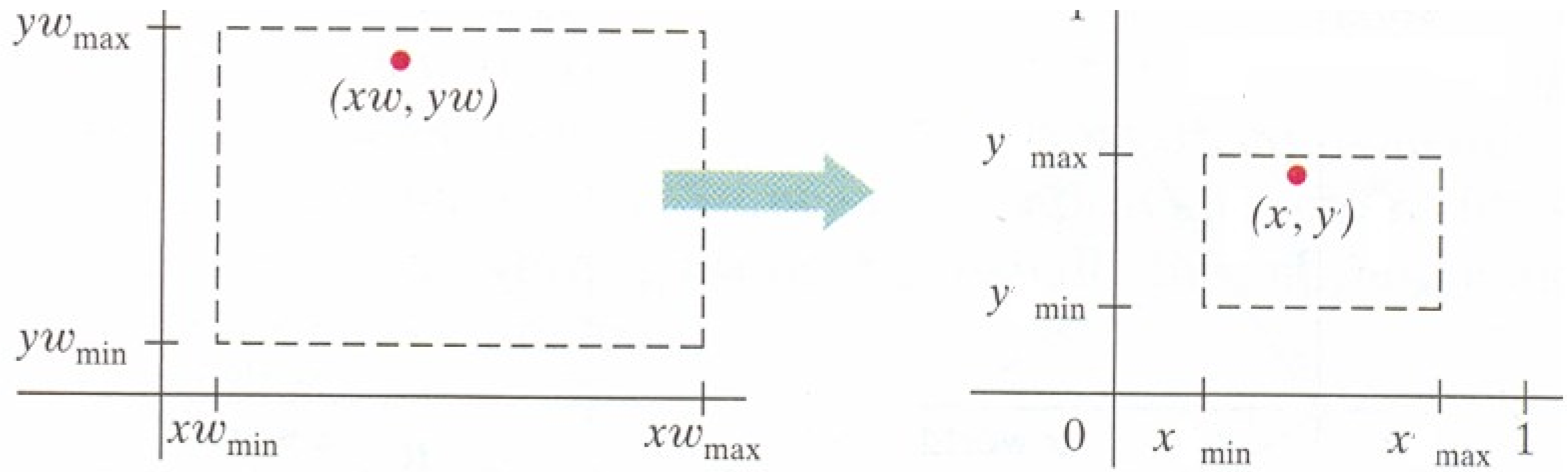




Coordinate mapping

Clipping Window

Viewport

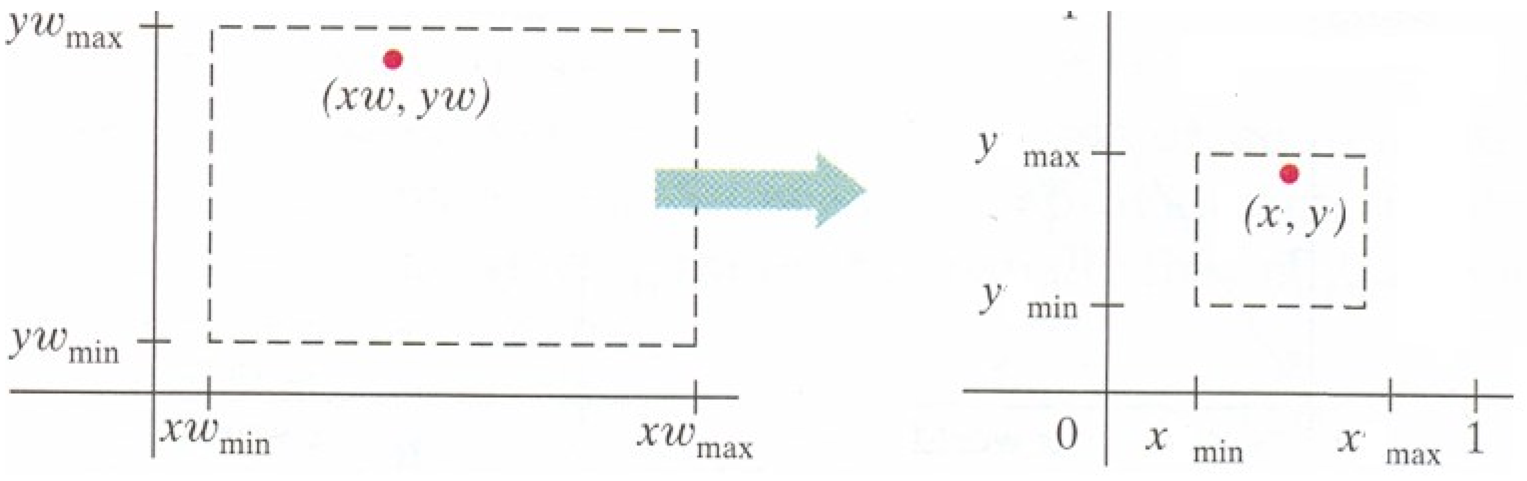


World Coordinates

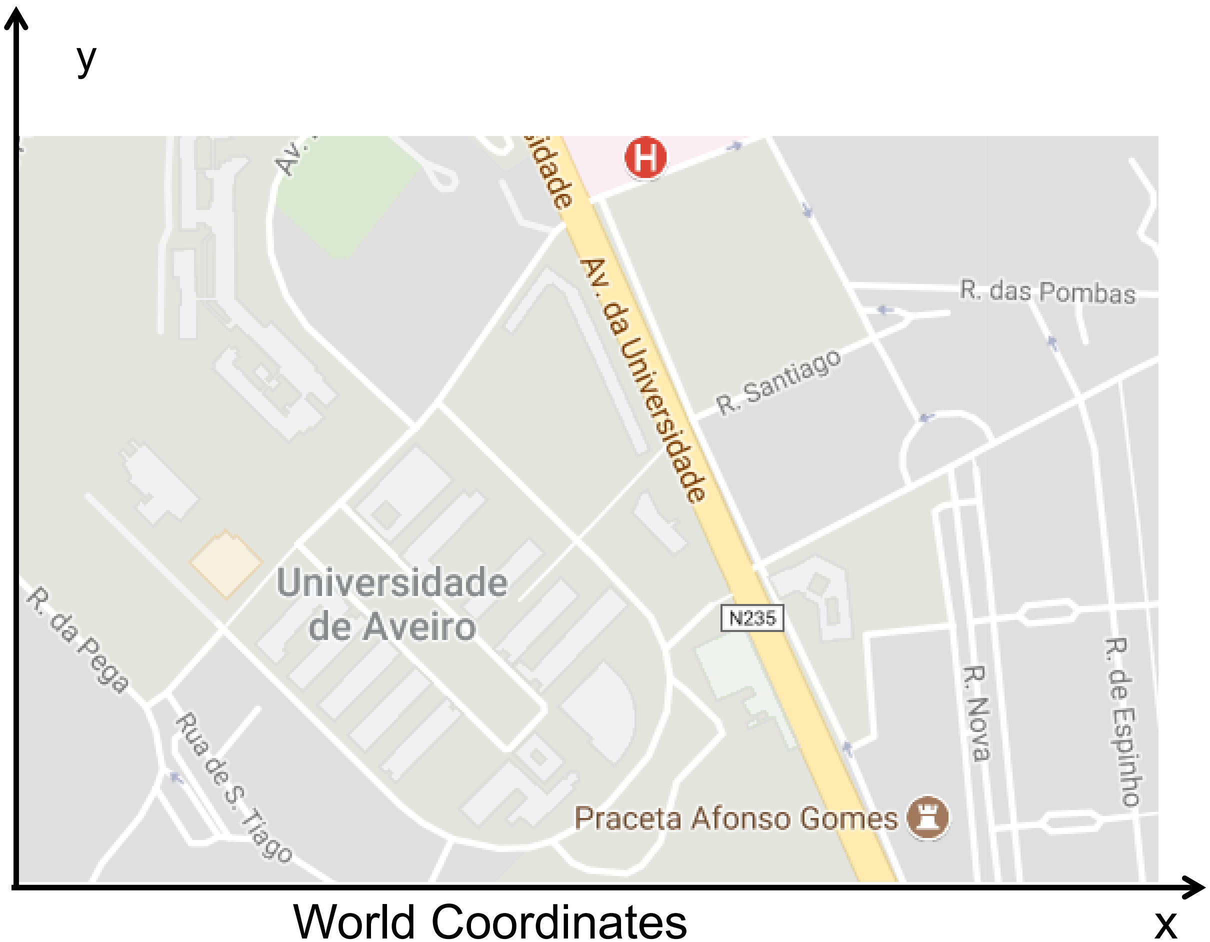
Screen coordinates

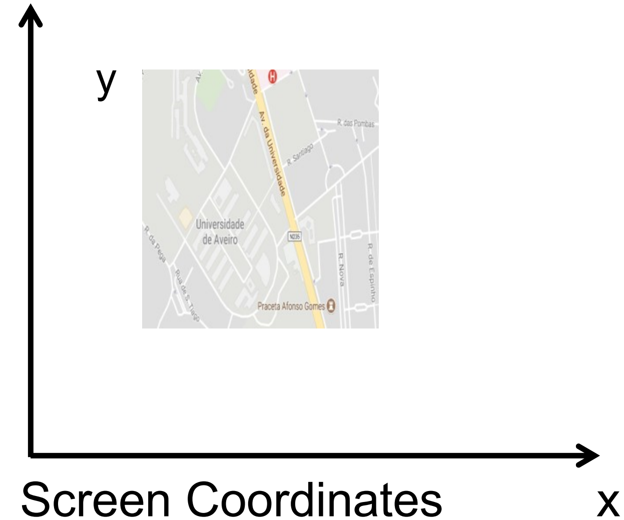
Coordinate mapping

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



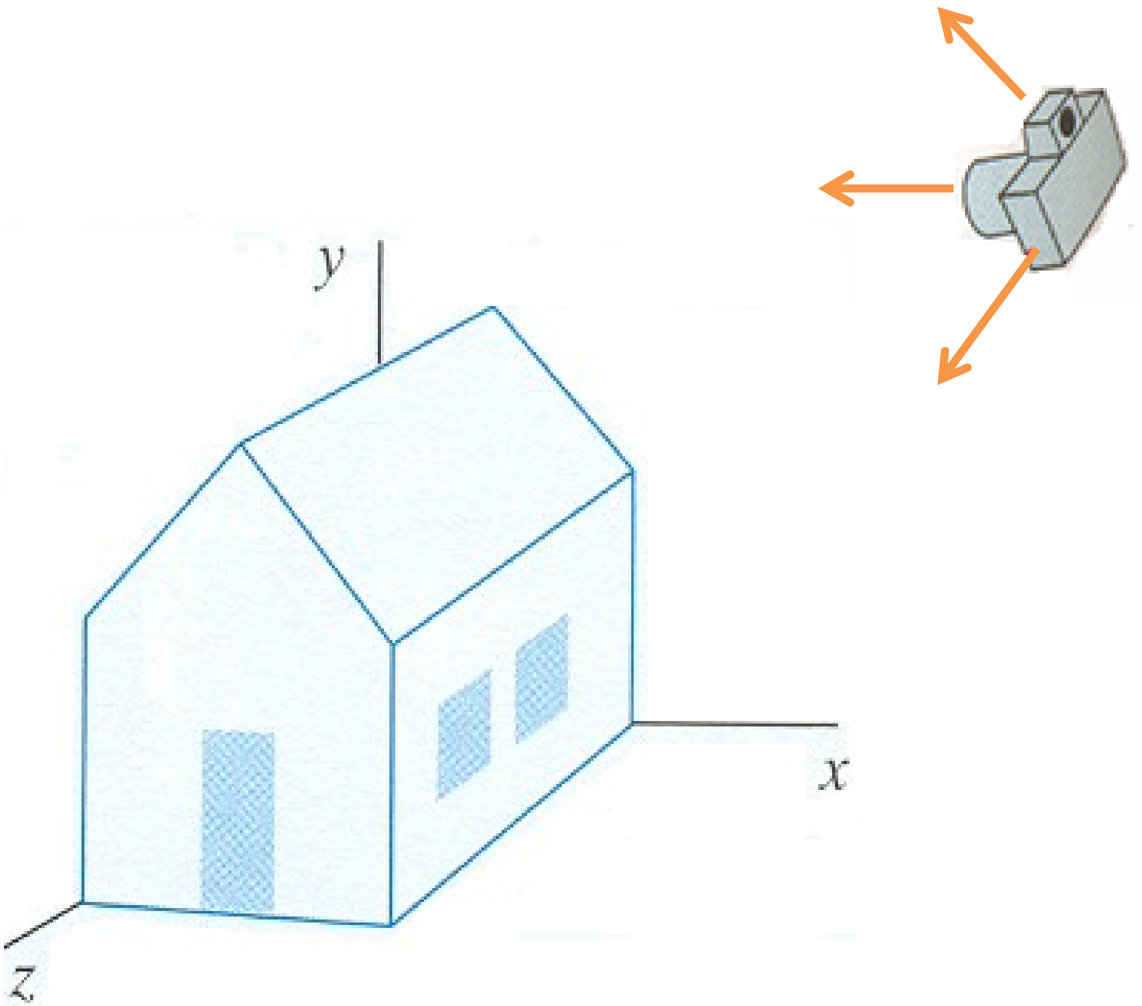
World -> screen





The **aspect ratio** is not the same in both situations: distortion!

3D Viewing

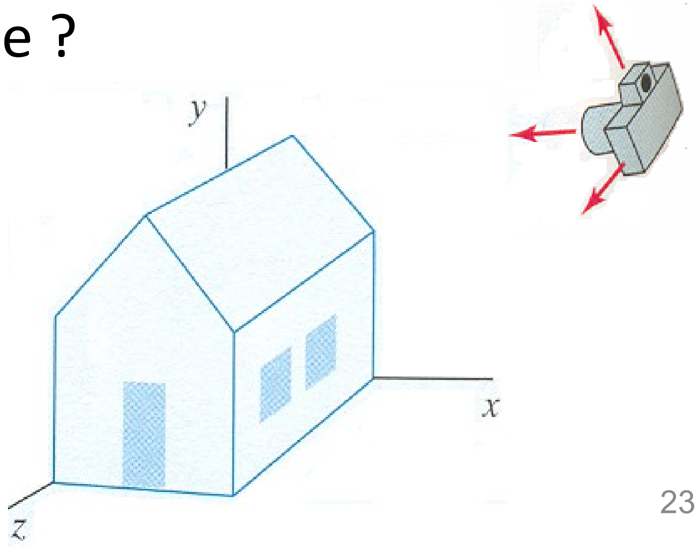


3D Viewing

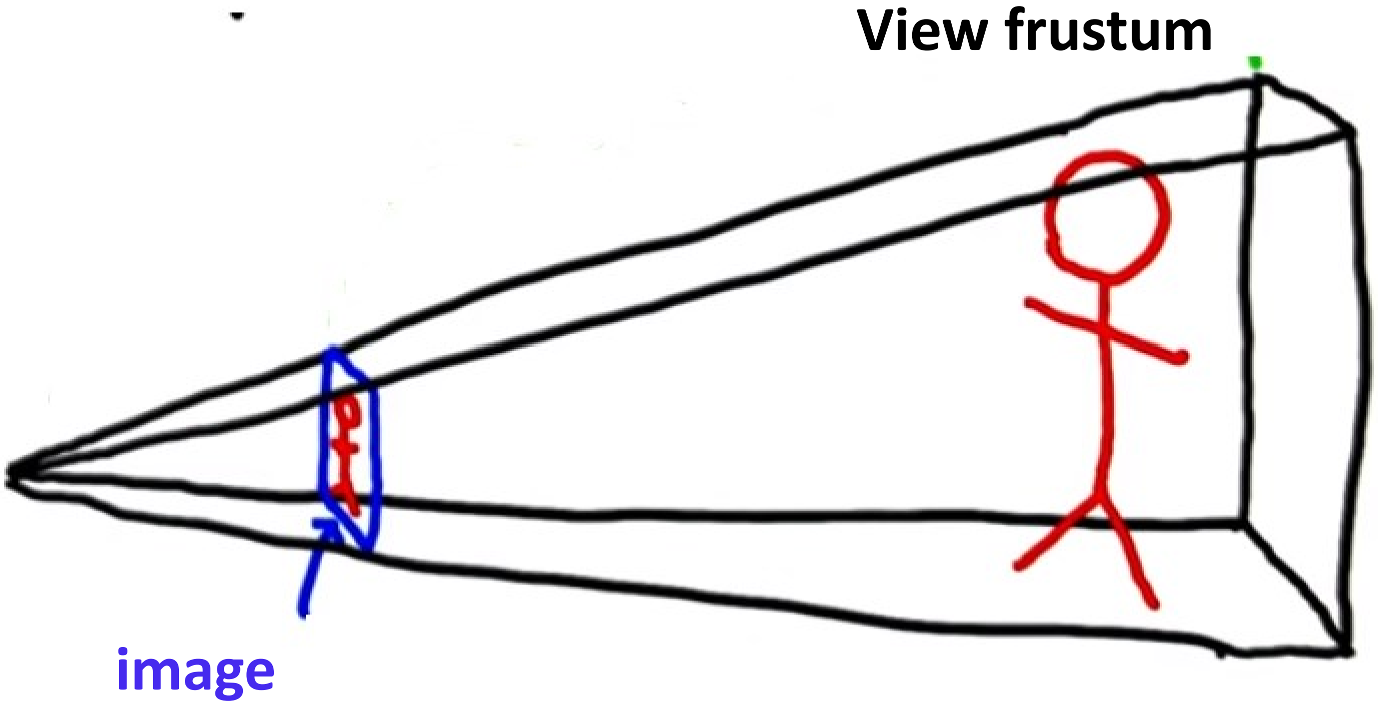
- Where is the observer / the camera ?
  - Position ?
  - Close to the 3D scene ?
  - Far away ?

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

- How to represent as a 2D image ?
  - Projection ?



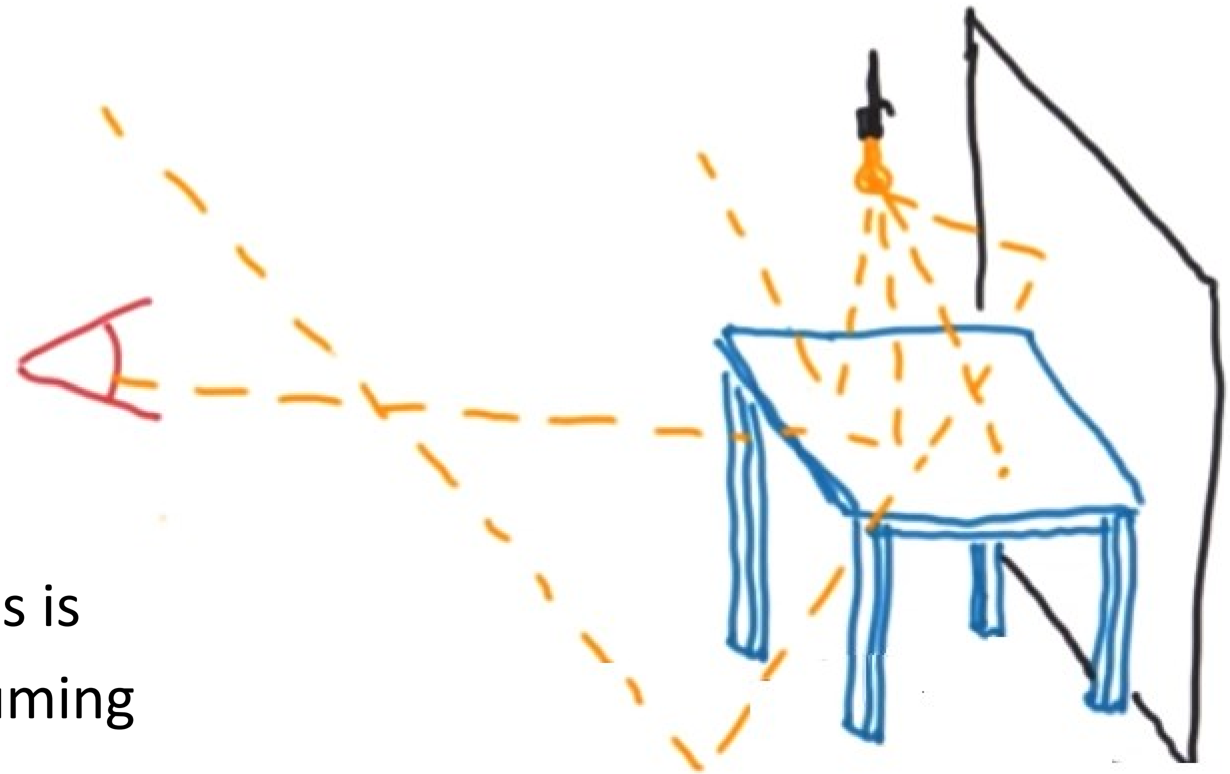
- Obtaining an image of the scene using perspective



(Interactive 3D Graphics, Udacity)

Light and Rendering

- In the real world the light emits rays that are reflected by objects and seen by the eye

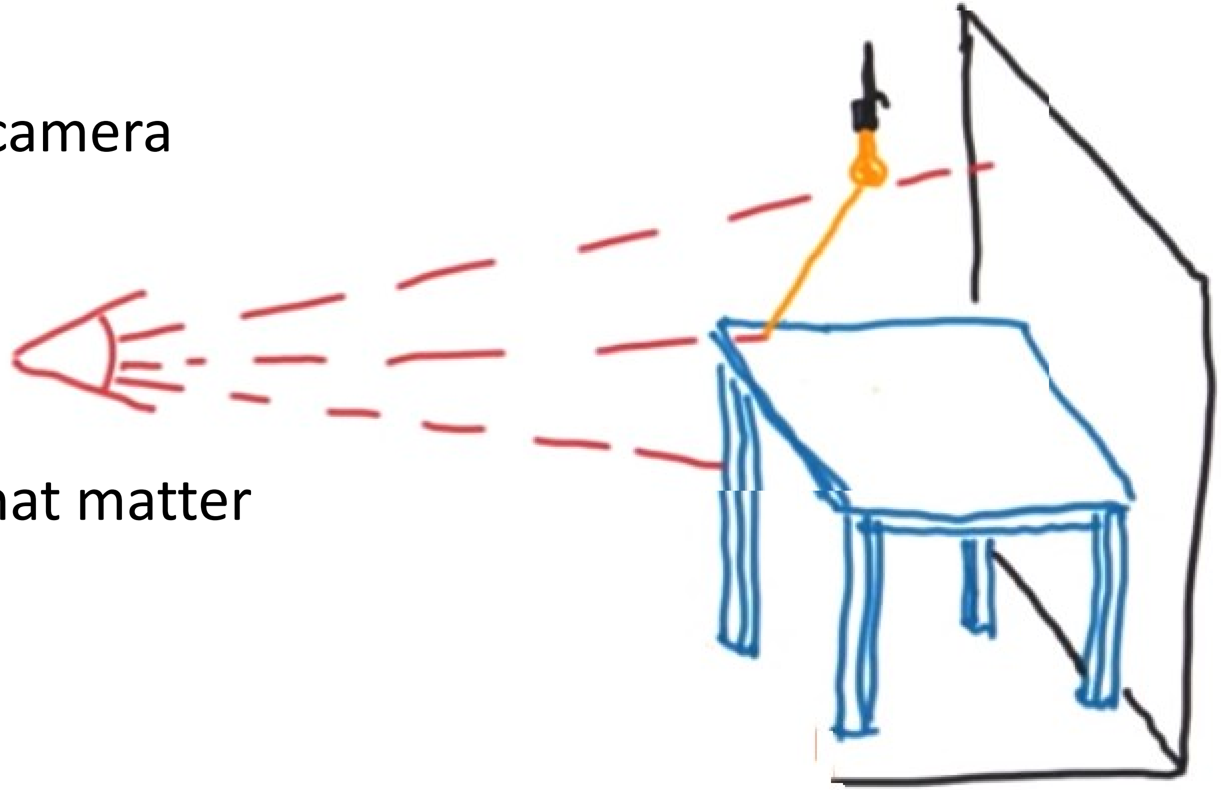


- Computing this is too time consuming

(Interactive 3D Graphics, Udacity)

Reversing the process in CG

- In CG simplifying assumptions may be made
- Start from the camera
- No shadows
- Only the rays that matter are processed



3D scene

Geometry

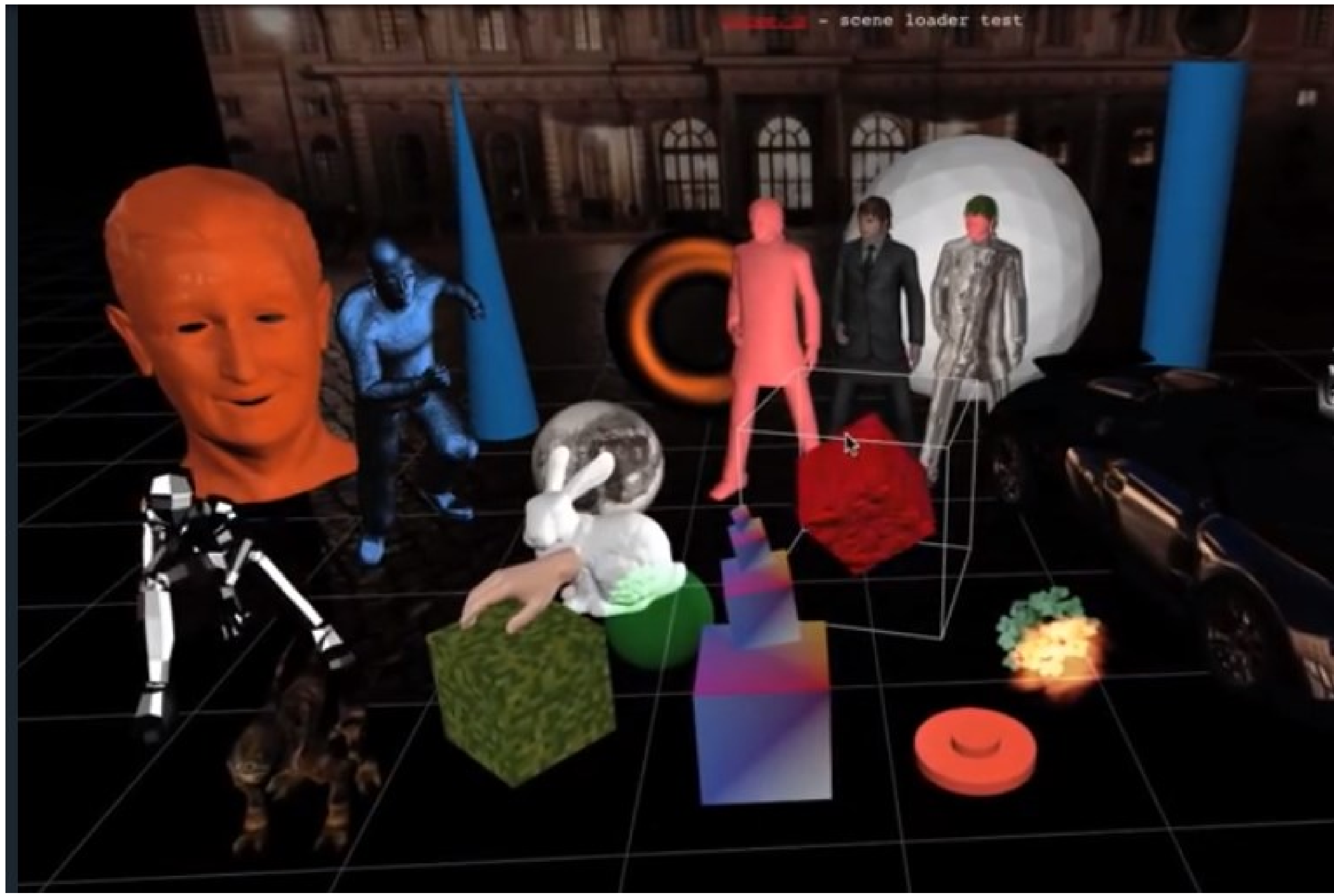
Material

Light

(animation)

+

Camera

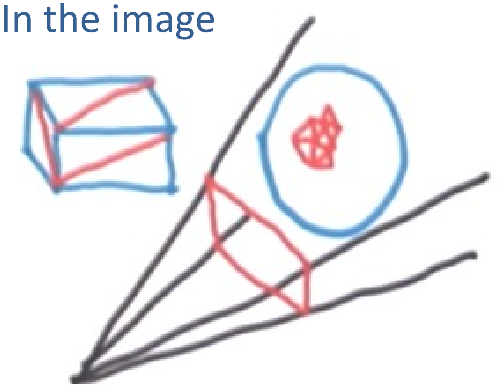


(Interactive 3D Graphics, Udacity)

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

3D visualization pipeline



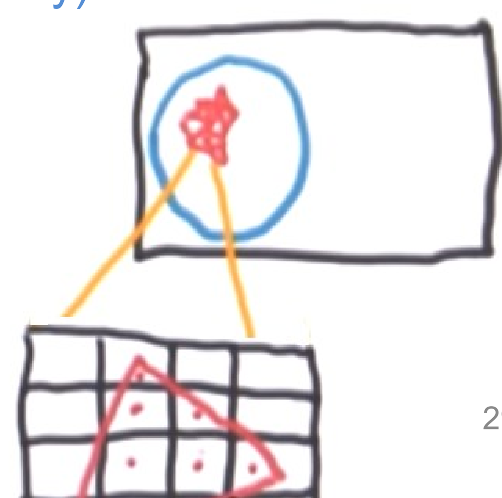
- Each object is processed separately
- Typically 3D triangles

(e.g. a cube or a sphere are made of triangles)

- Triangles are modified by the camera view of the world
- Compute the color of each pixel
- Is the object inside the view frustum?
  - (No -> next object!)
  - Yes -> project and compute location

  of each triangle on the screen (rasterization)

(Interactive 3D Graphics, Udacity)



Projection (from 3D to 2D)

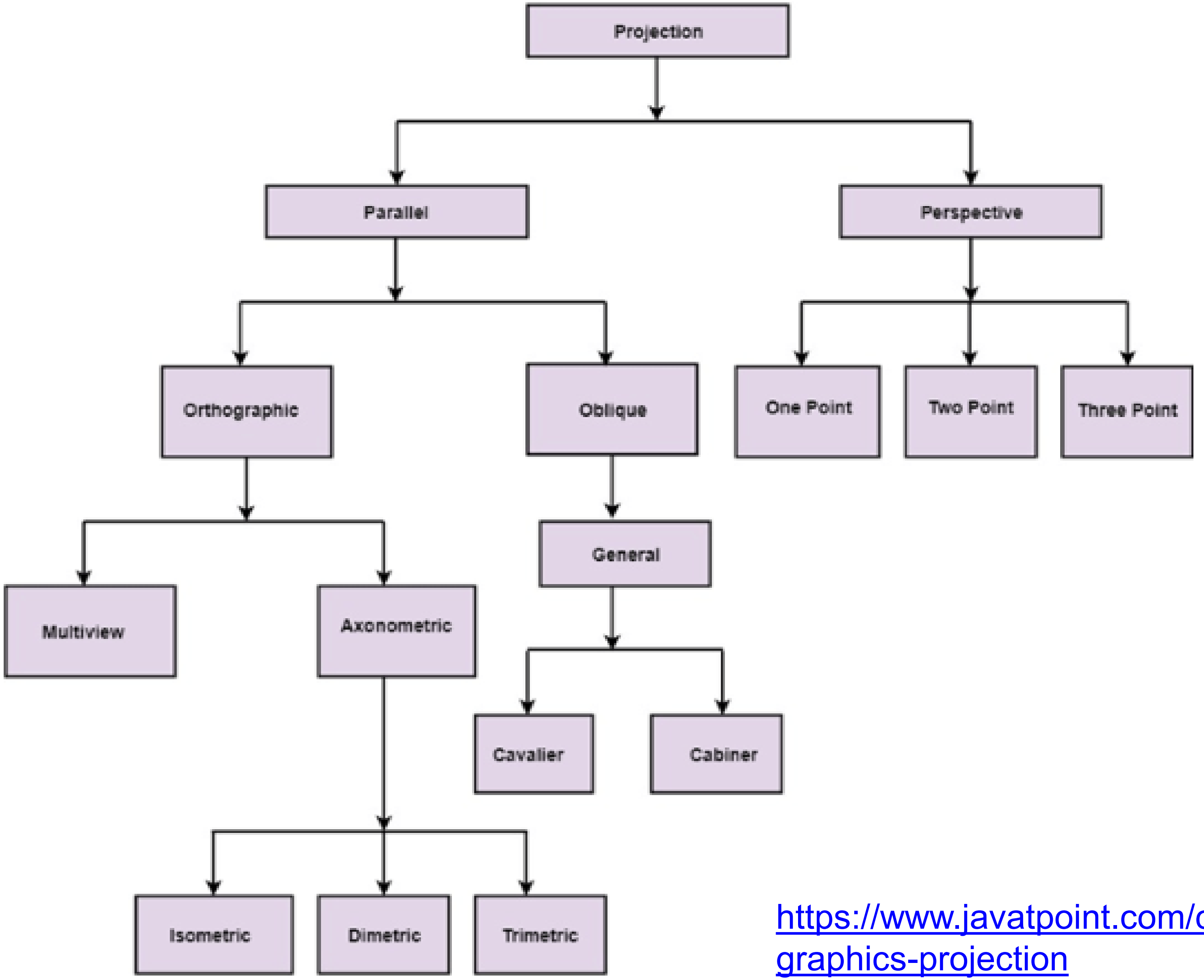
Parallel Projection

(allows measures)

Perspective Projection

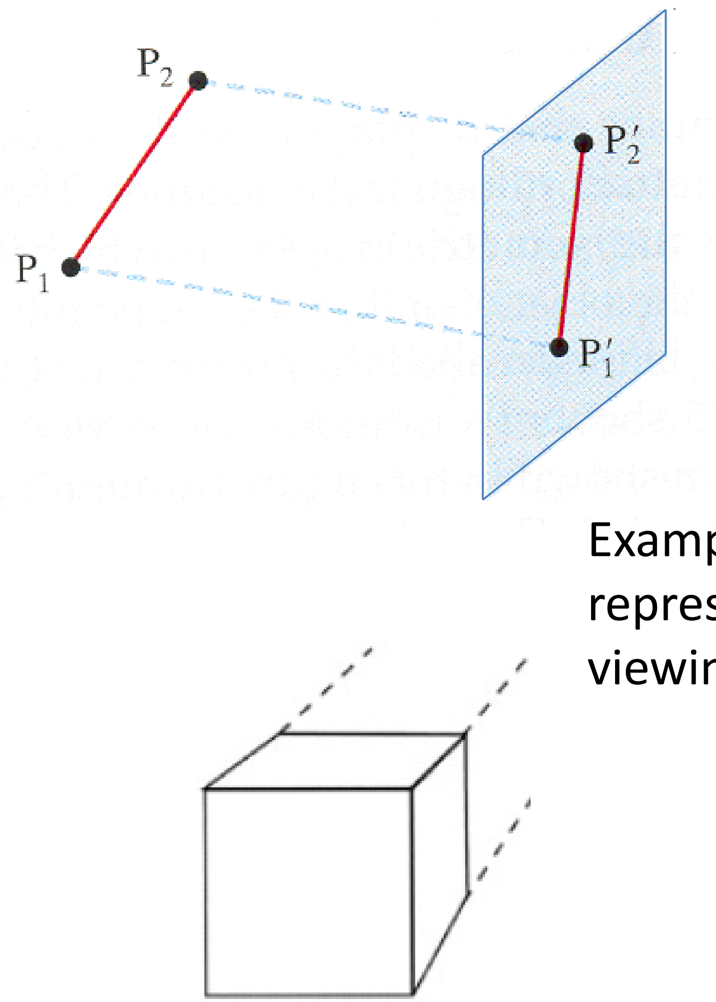
(more realistic images)

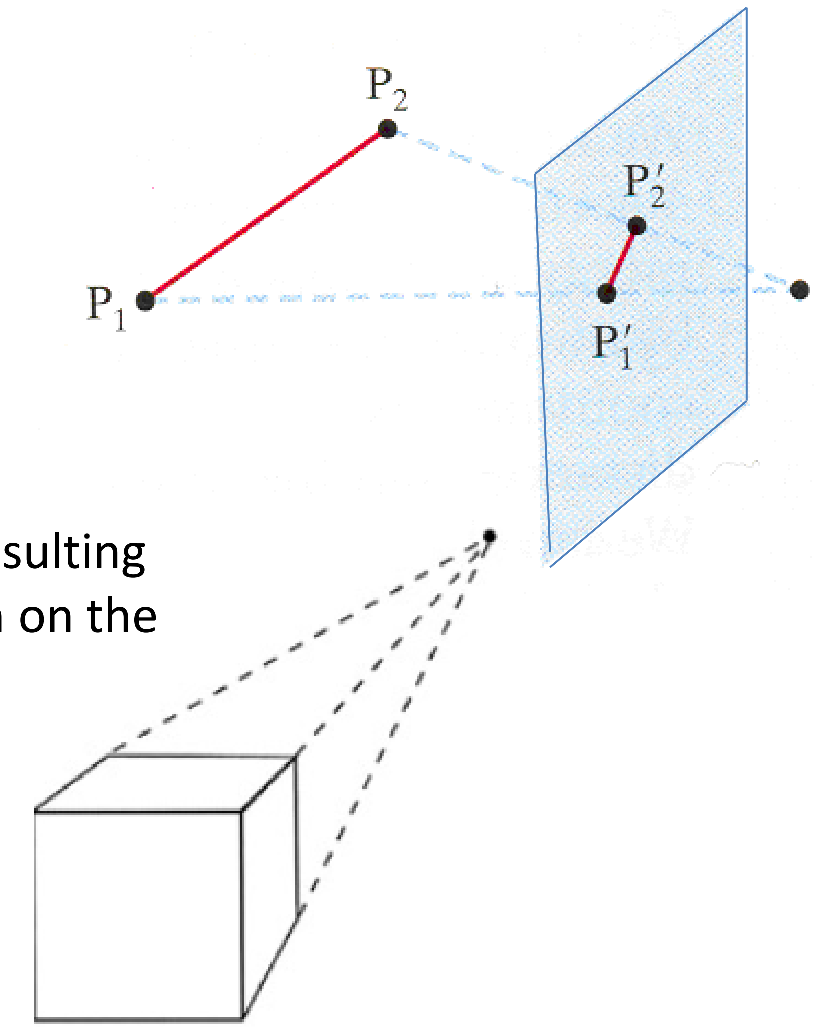
Projections



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

Projections





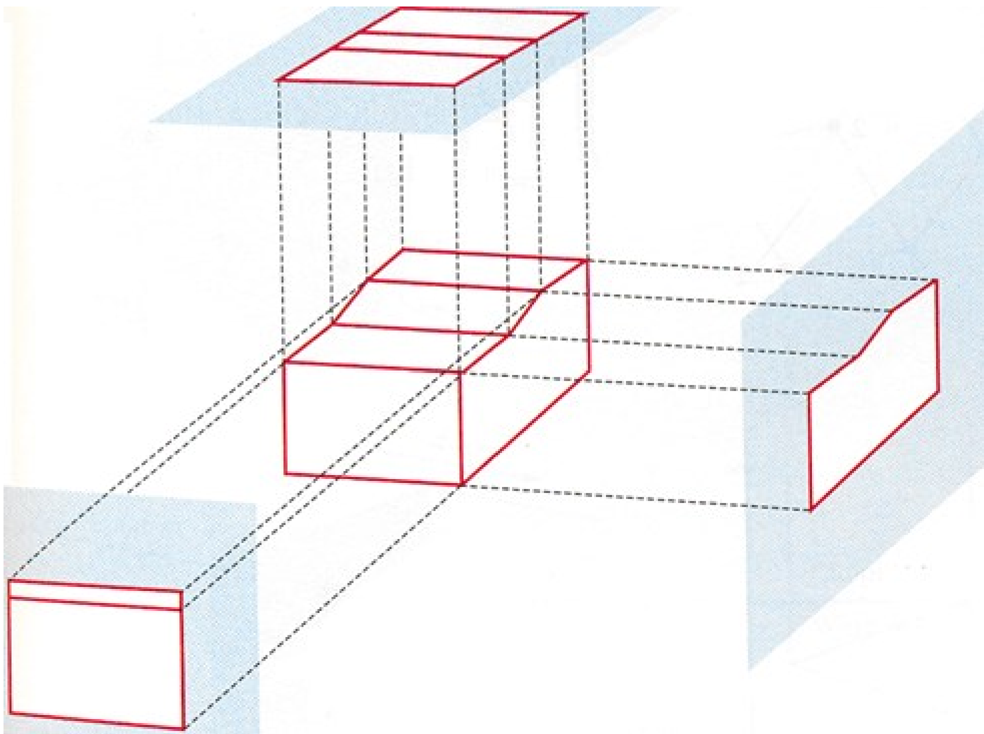
Examples of resulting representation on the viewing plane

Parallel Projection

Perspective Projection

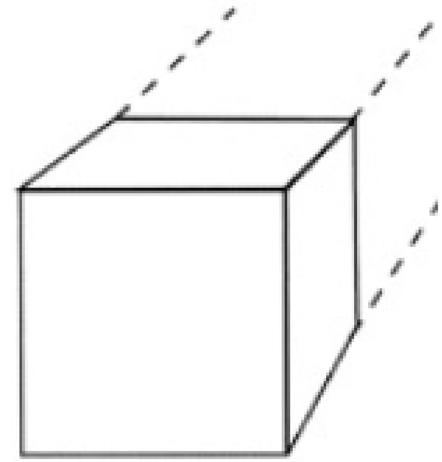
(Hearn & Baker, 2004)

Parallel Projections



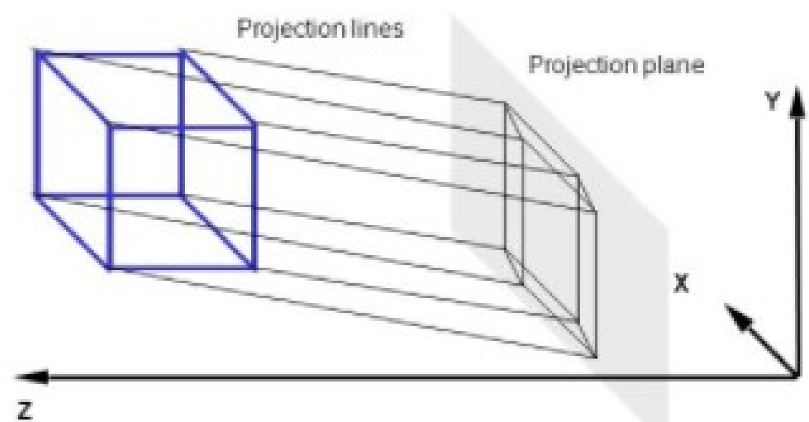
Orthographic/ Multiview projection

(Hearn & Baker, 2004)



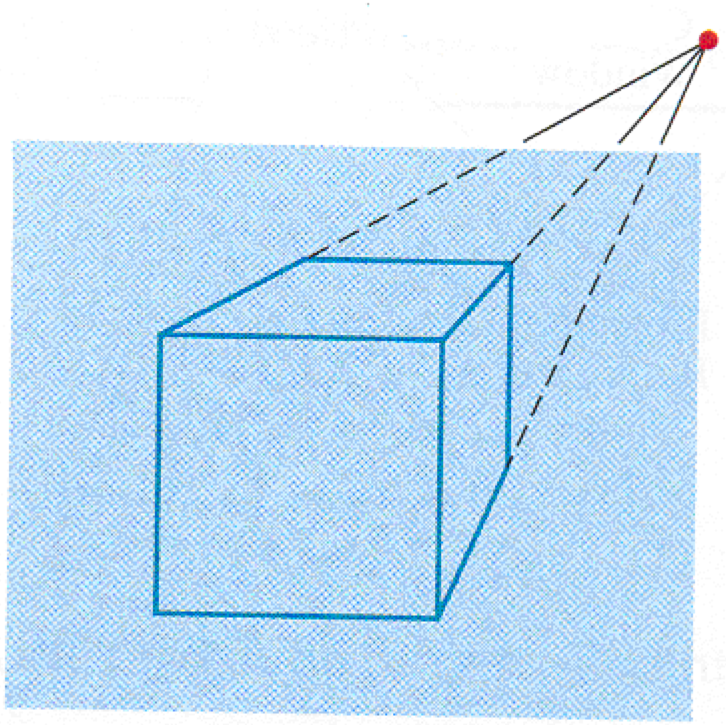
Orthographic / Axonometric projection

Oblique projection

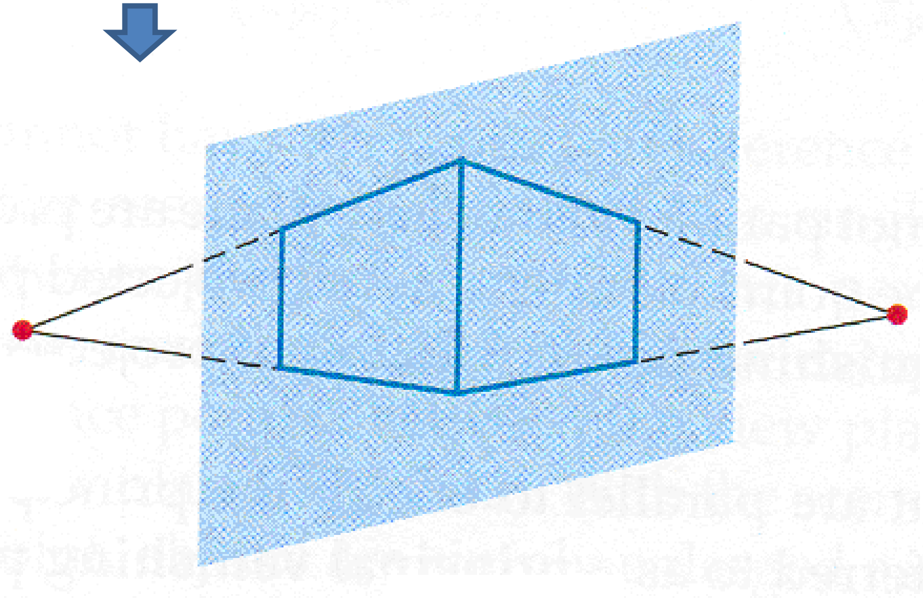


Perspective Projections

One vanishing point perspective projection



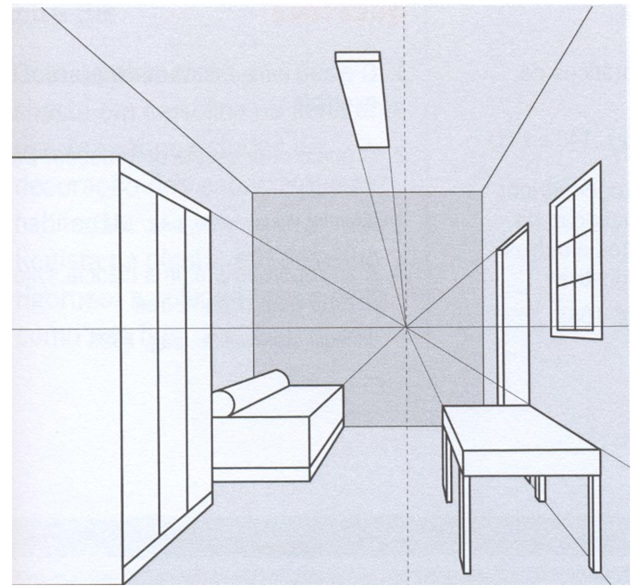
Two vanishing points perspective projection



(Hearn & Baker, 2004)

Perspective Projections

Foreshortening indicates a perspective projection



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

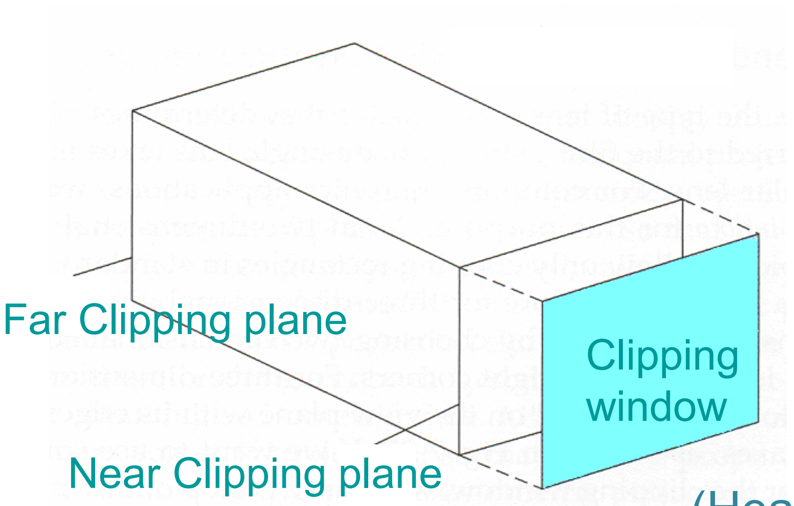
(Wijipedia)

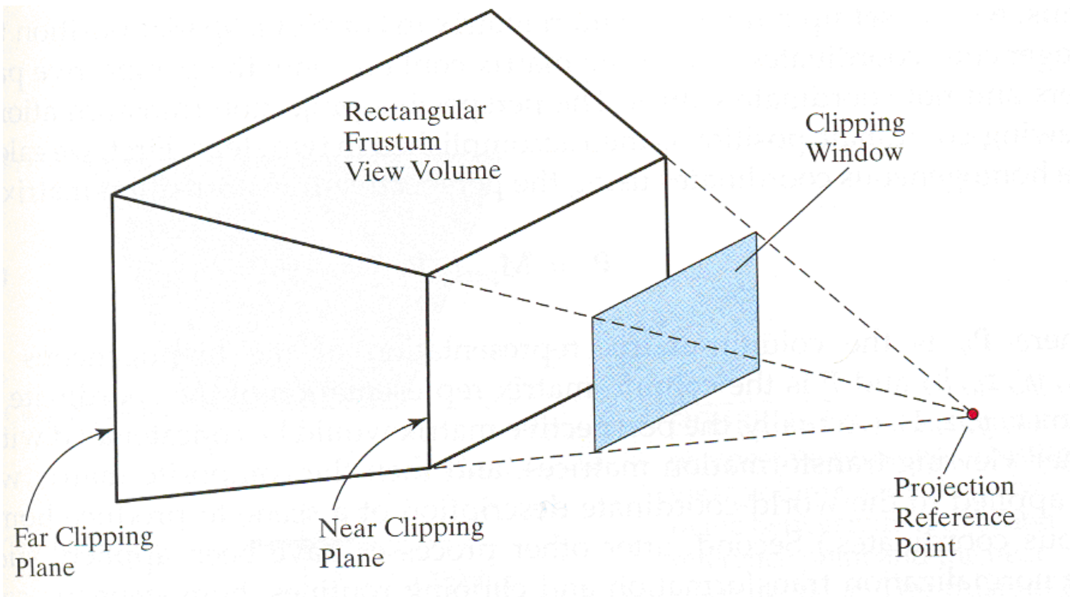
How to represent ?

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

How to limit what is observed and represented ?

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



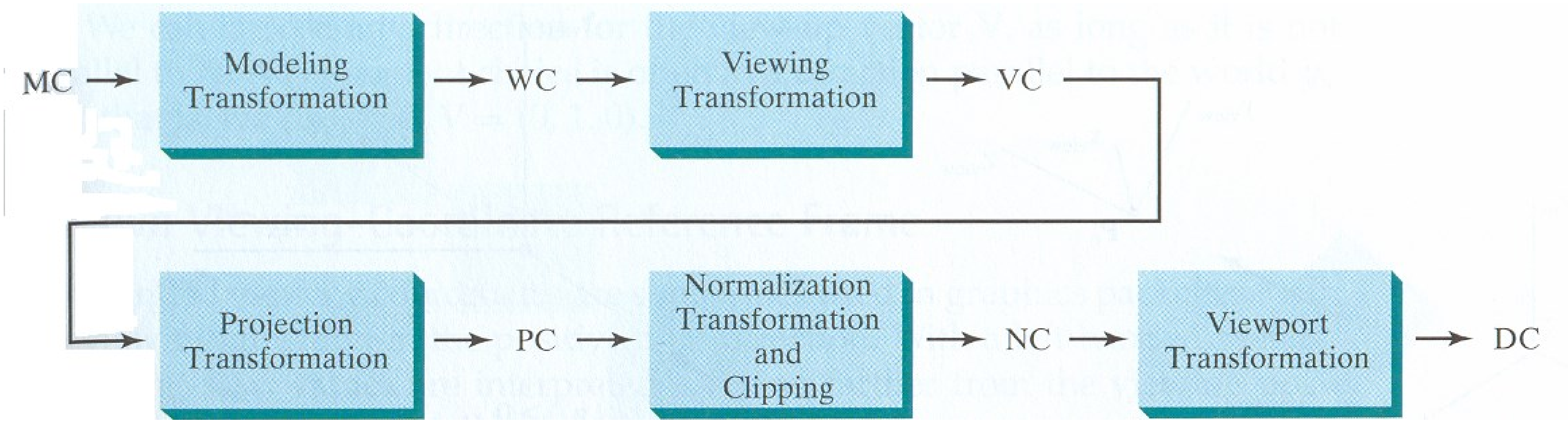


(Hearn & Baker, 2004)

Parallel projection

Perspective projection

3D visualization pipeline
(coordinate transformations)



(Hearn & Baker, 2004)

3D visualization pipeline

- Main operations represented as point transformations
  - Homogeneous coordinates
  - Transformation matrices
  - Matrix multiplication

Basic 2D Transformations

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

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

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

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

Vector notation

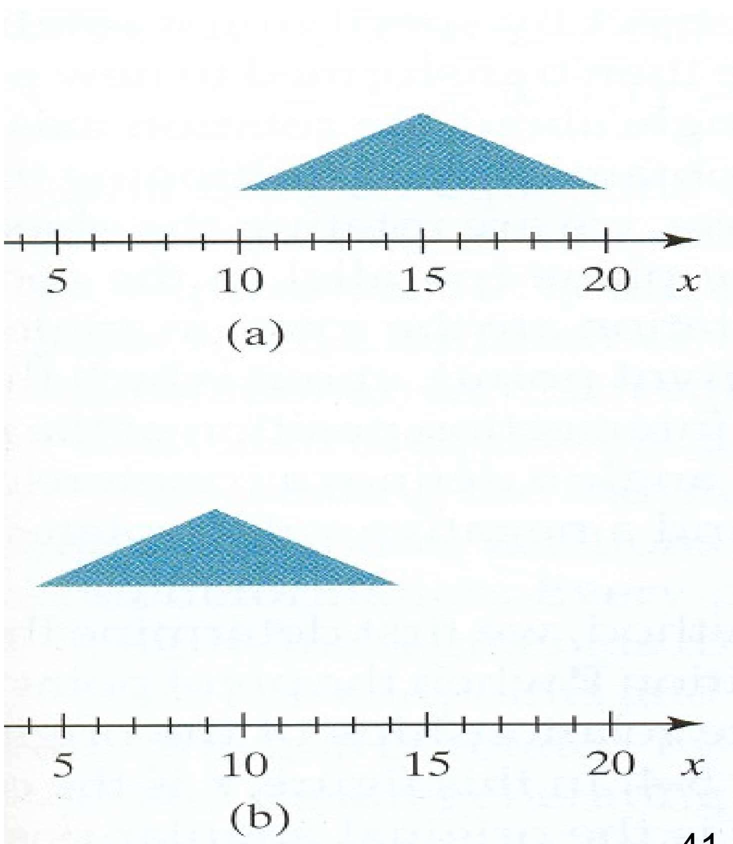
- Basic transformations:
  - Translation
  - Scaling
  - Rotation

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

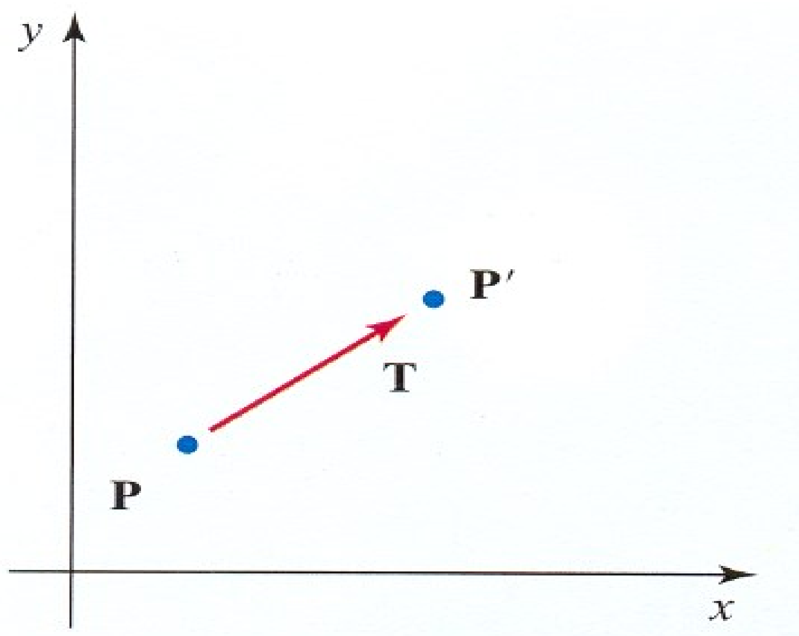
  - To apply a translation to a polygon we need only to transform the vertices



Translation

- It is necessary to specify translations in x and y

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



$$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad T = \begin{bmatrix} tx \\ ty \end{bmatrix}$$

$$P' = P + T$$

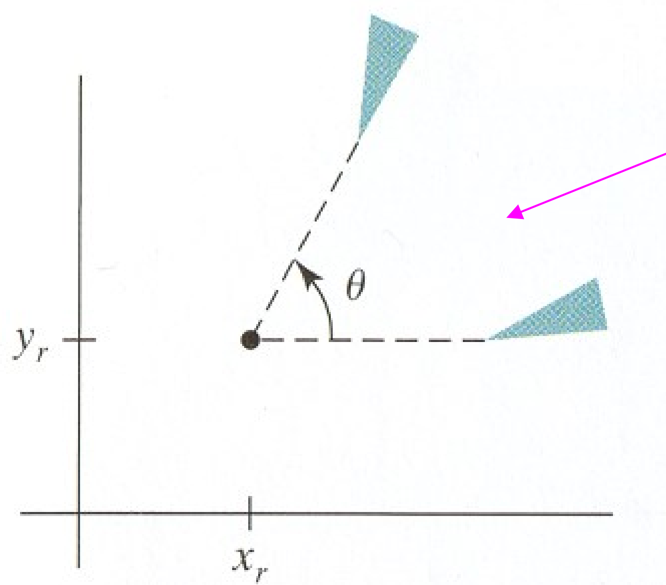
transformation matrix

Rotation

- To apply a rotation we need to specify:

  - a point (center of rotation) (x_r, y_r)

  - A rotation angle ϑ (positive - counter-clockwise)



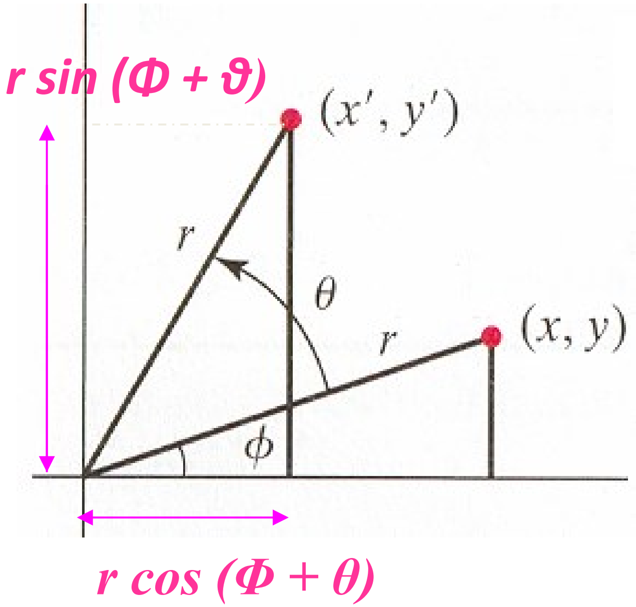
Positive rotation

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 coordenates of the original point:

$$x = r\cos\Phi$$
$$y = r\sin\Phi$$

Replacing:

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

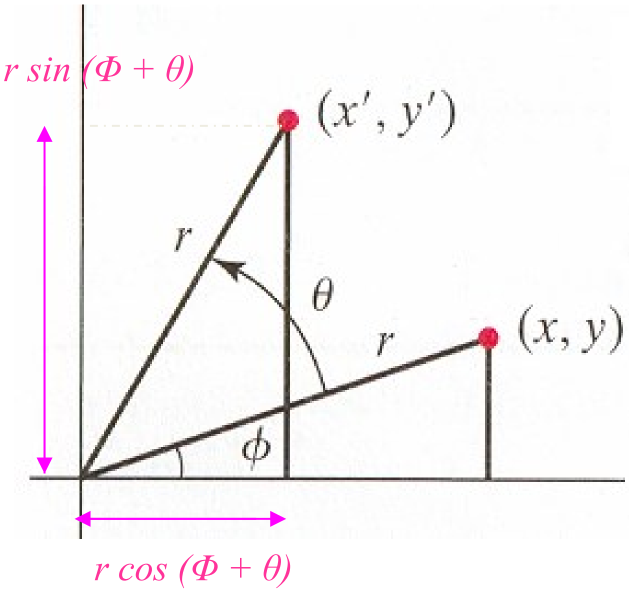
2D Rotation in matrix notation

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

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

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

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



Scaling

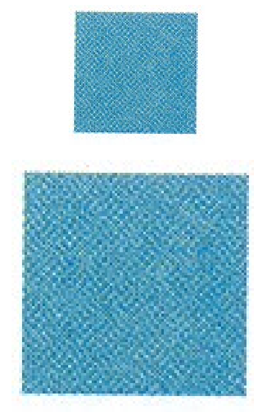
- Modifies the size of an object; we need to specify scaling factors: s_x and s_y

$$x' = x \cdot s_x$$
$$y' = y \cdot s_y$$

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

Trasformation 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

- 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

- Each point is now represented by 3 coordinates

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

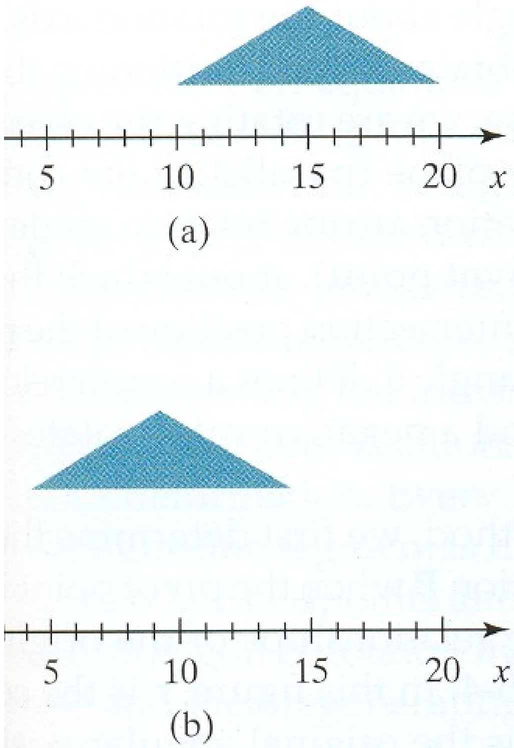
$$x = x_h / h \qquad y = y_h / h$$

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

2D Translation

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

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



(Hearn & Baker, 2004)

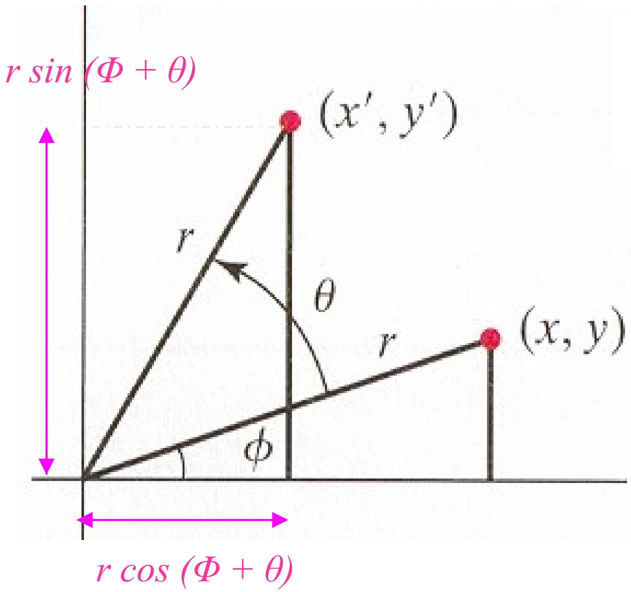
2D Rotation

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

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

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

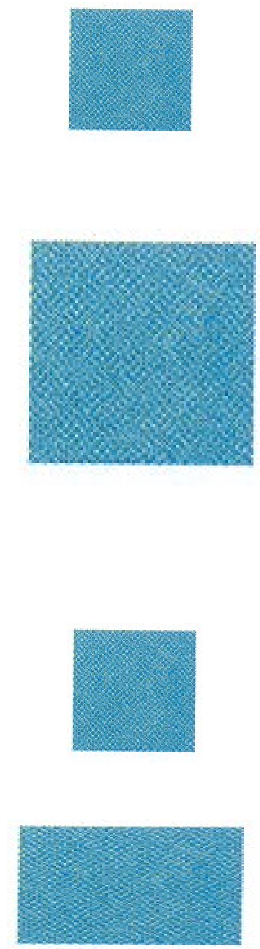
$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$



2D Scaling

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

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



(Hearn & Baker, 2004)

Concatenation of two translations

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

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

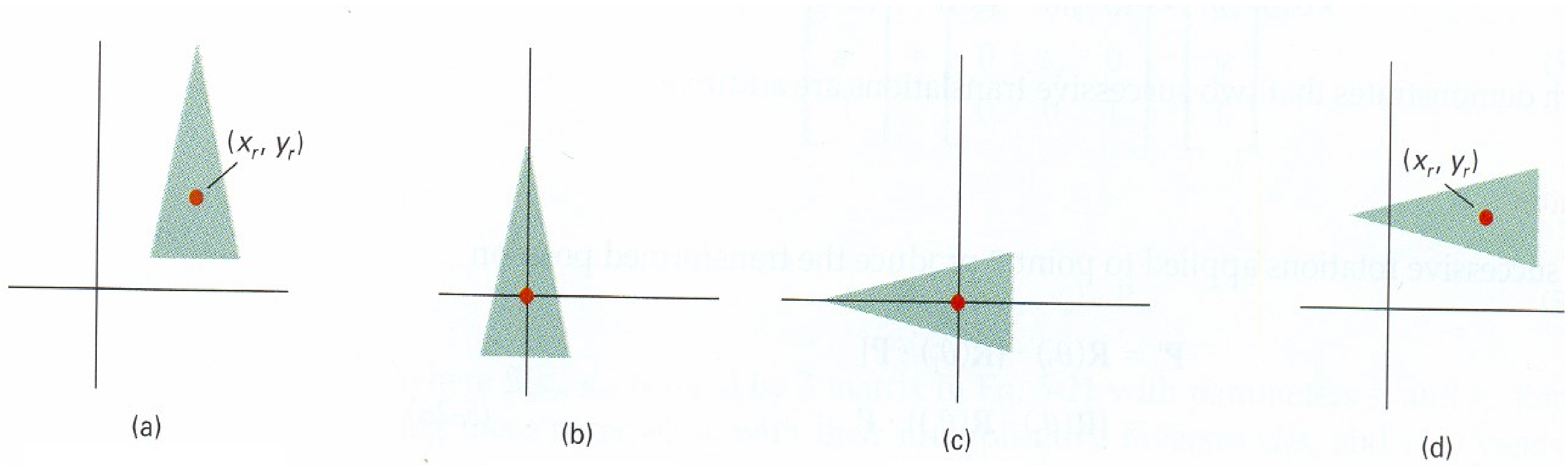
$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

Concatenation of two scaling transformations

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

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

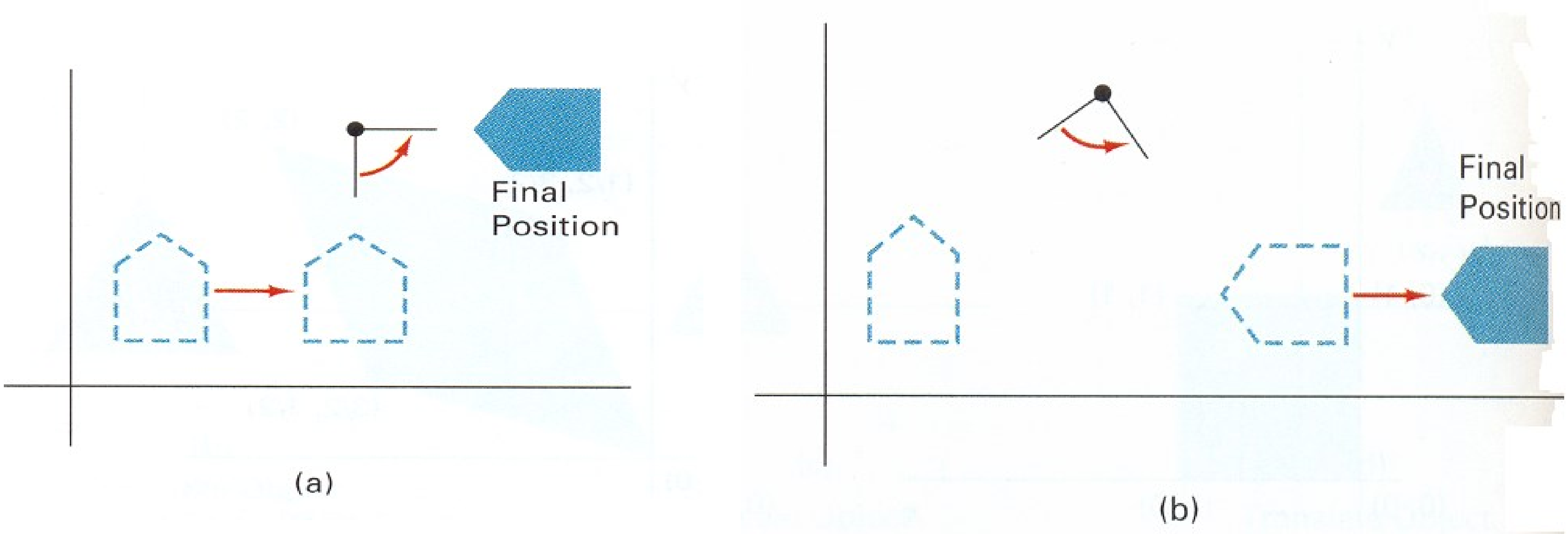
Arbitrary Rotation



(Hearn & Baker, 2004)

Translation + Rotation + Inverse Translation

Order is important !



(Hearn & Baker, 2004)

3D Transformations

- Translation

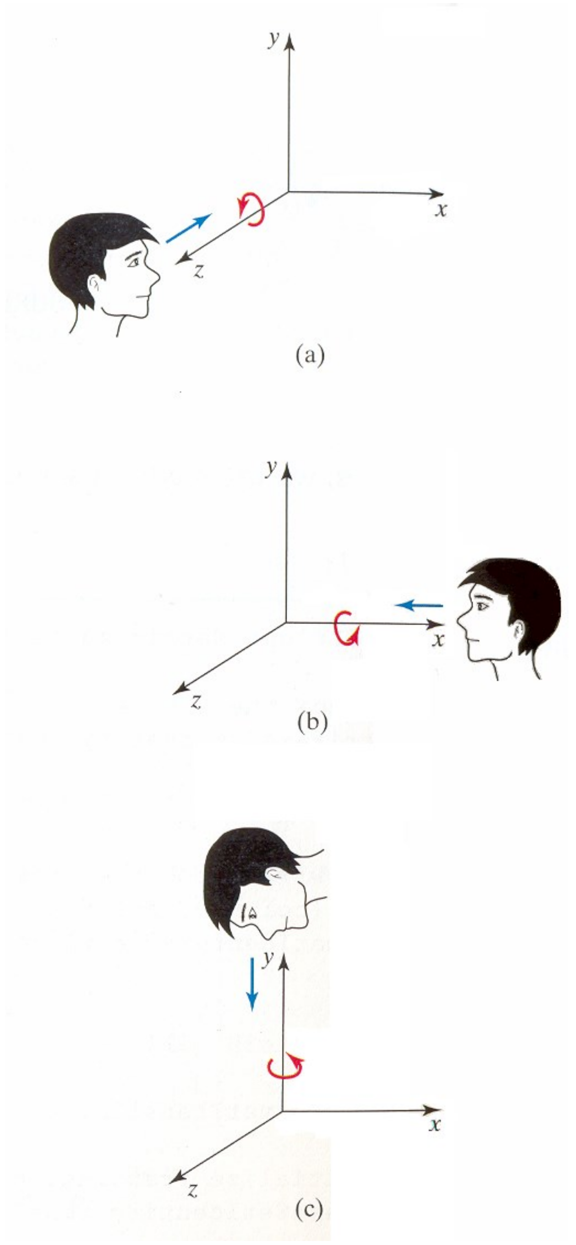
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Scaling

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

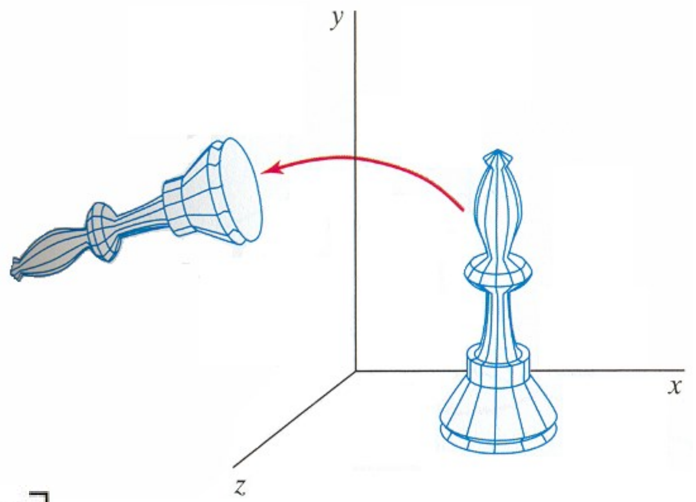
3D Rotation

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



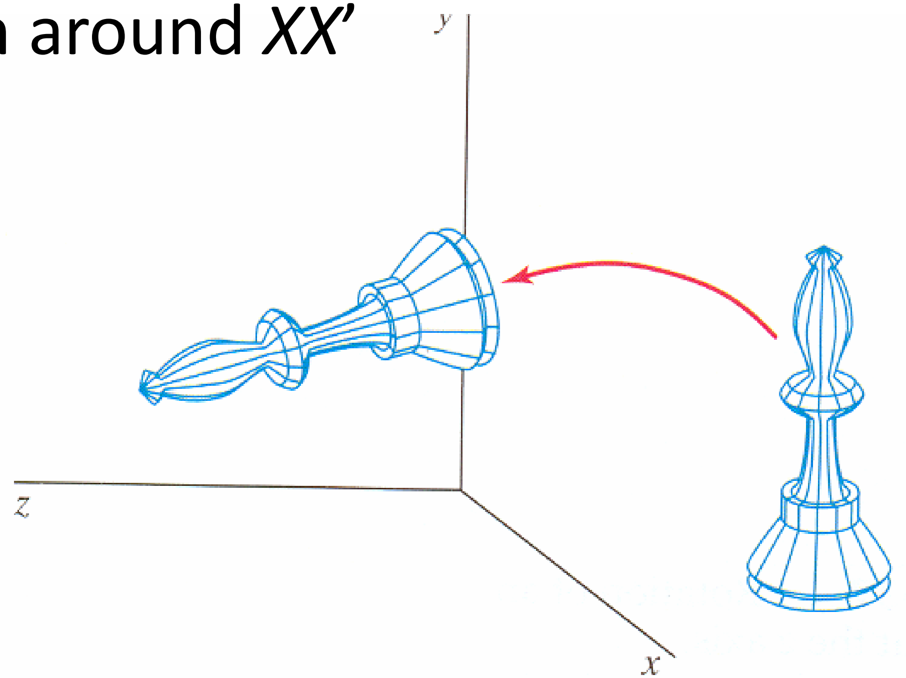
(Hearn & Baker, 2004)

Rotation around ZZ’



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

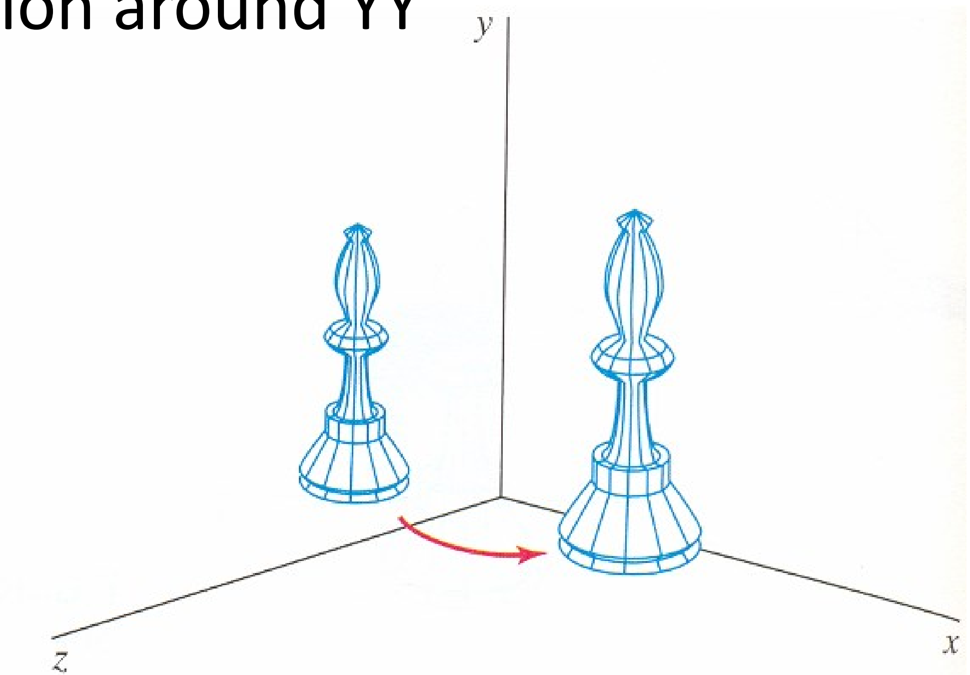
Rotation around *XX'*



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

(Hearn & Baker, 2004)

Rotation around YY'



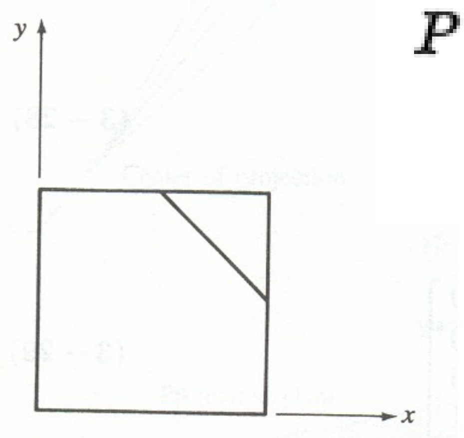
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(Hearn & Baker, 2004)

How to apply Projections?

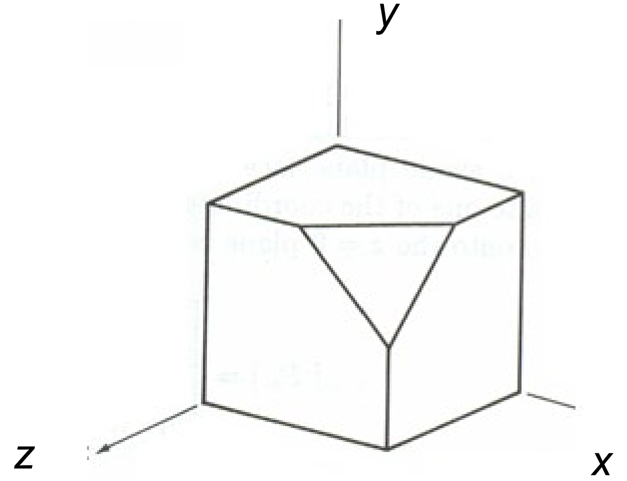
-
    - Also by matrix multiplication

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



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

***zz* coordinates are discarded**



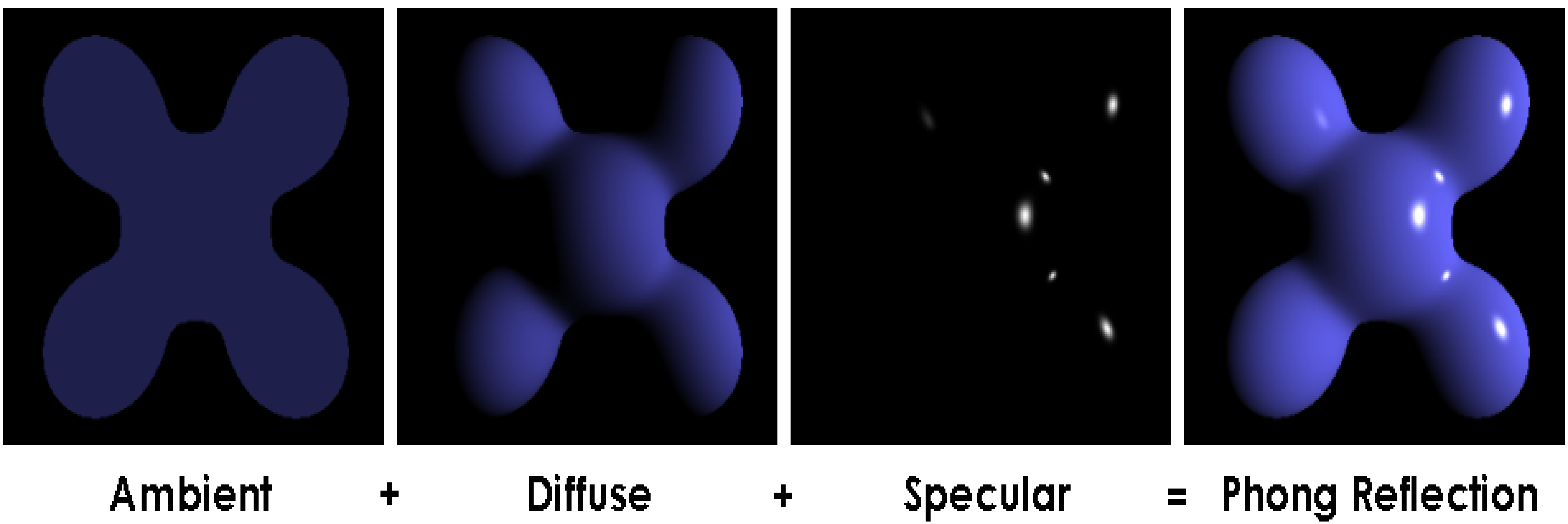
Lighting

- Compute surface color based on
  - Type and number of light sources
  - Illumination model
    - Phong: ambient + diffuse + specular components
  - Reflective surface properties
  - Atmospheric effects
    - Fog, smoke
- Polygons making up a model surface are shaded
  - Realistic representation

Phong reflection model

Empirical model of the local illumination of points on a surface

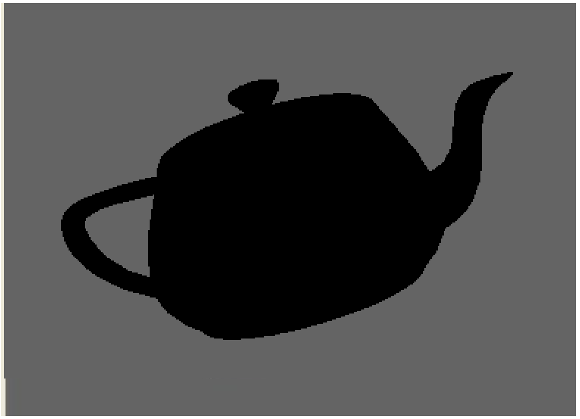
It describes the way a surface reflects light as a combination of the **diffuse reflection** of rough surfaces with the **specular reflection** of shiny surfaces and a component of **ambient light**



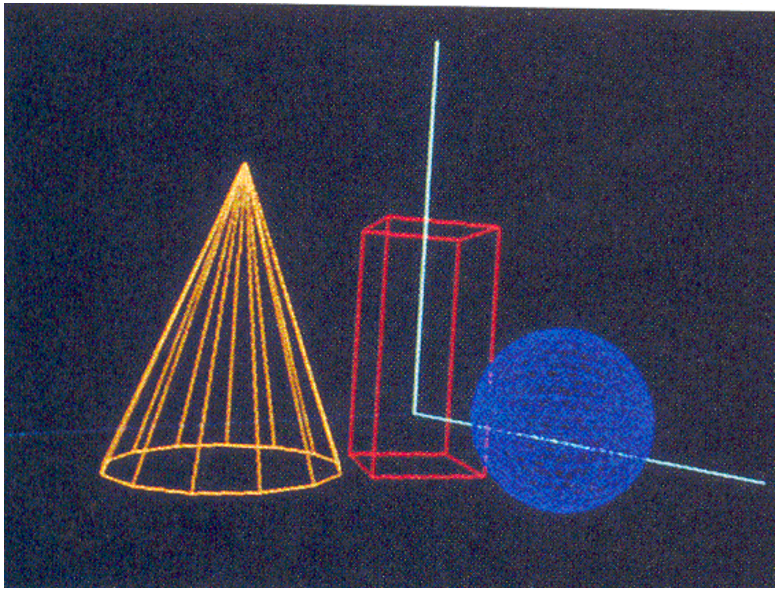
(Wikipedia)

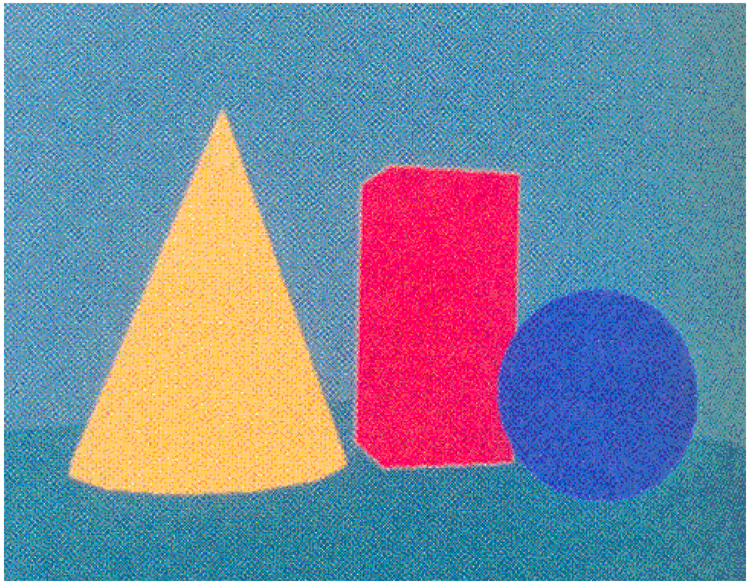
Phong Model – Ambient illumination

- Constant illumination component for each model
- Independent from viewer position or object orientation !
- Take only material properties into account !



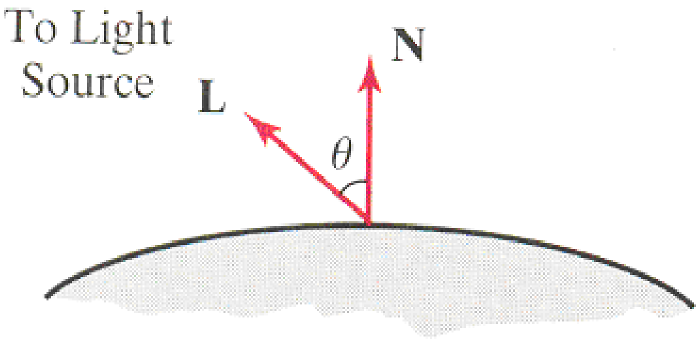
Phong Model – Ambient illumination





Phong Model – Diffuse reflection

$$I_{l,\text{diff}} = \begin{cases} k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$



- Model surface is an ideal diffuse reflector
  - What does that mean ?

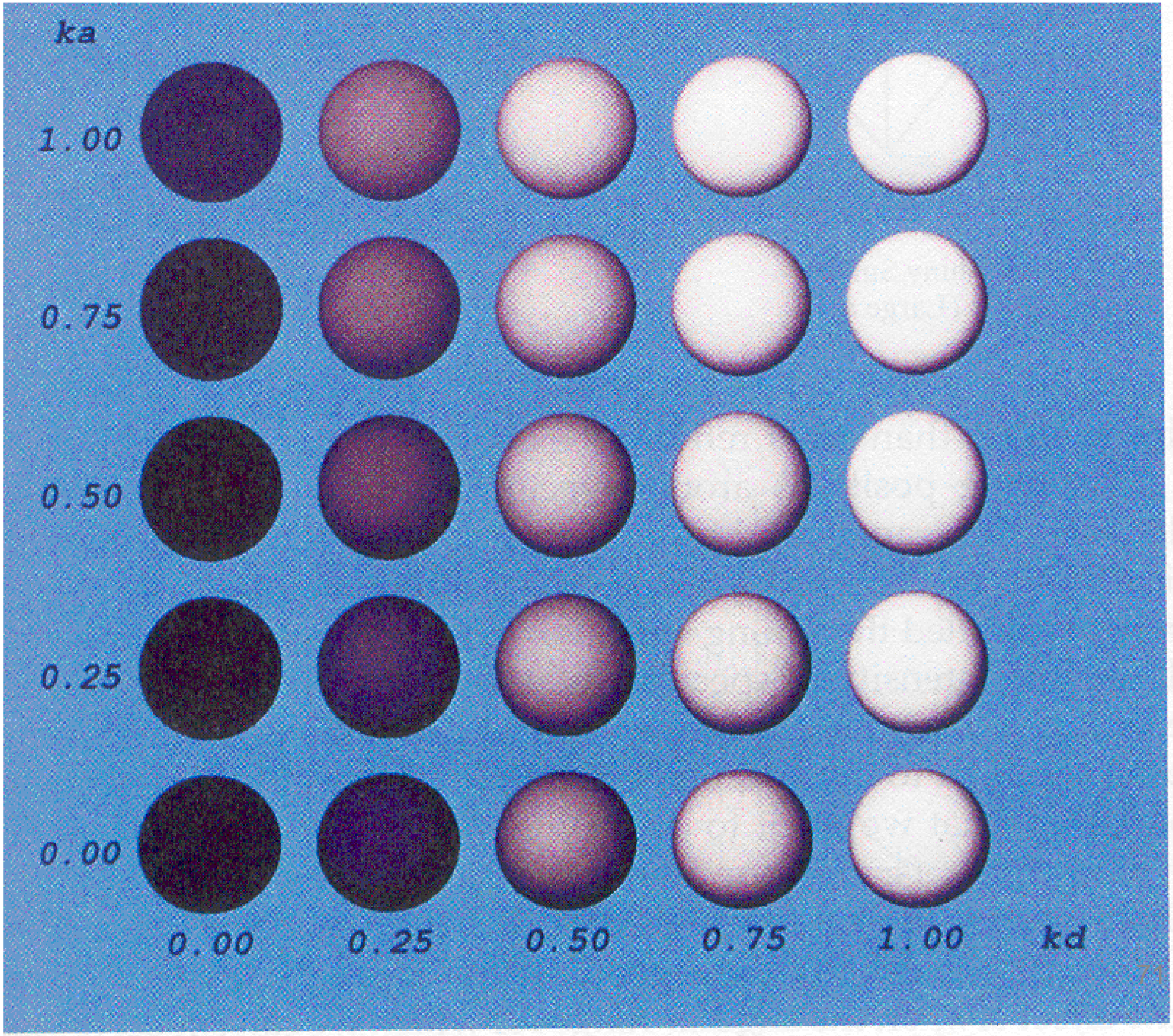
- Independence from viewer position !

- Unit vectors !!

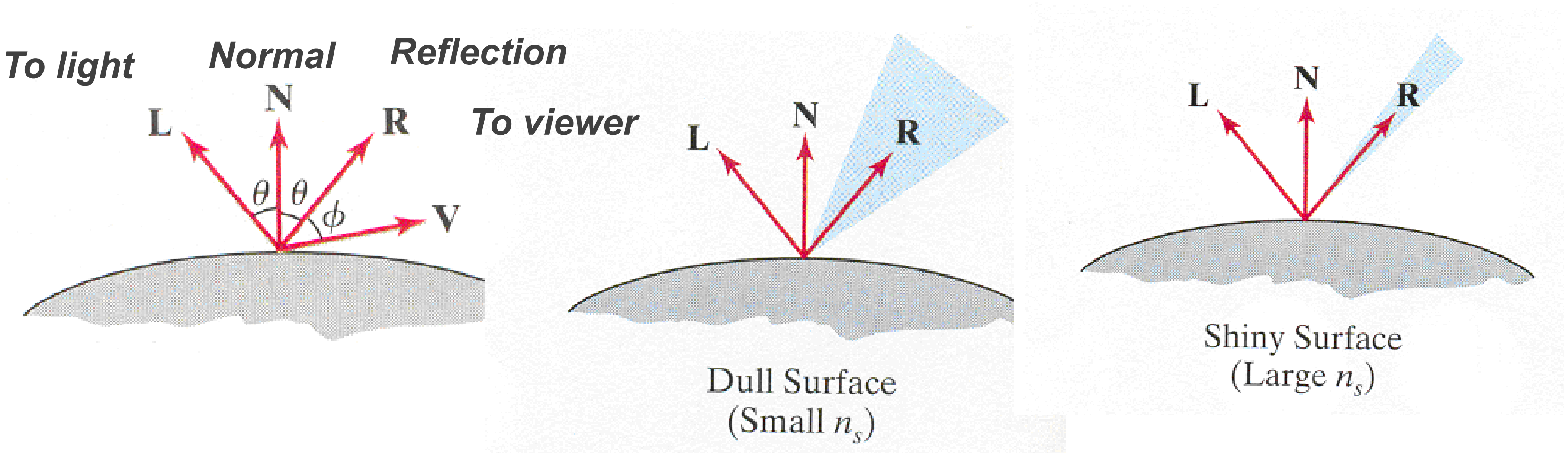
Phong Model

ka – ambient

Kd - diffuse



Phong Model – Specular reflection

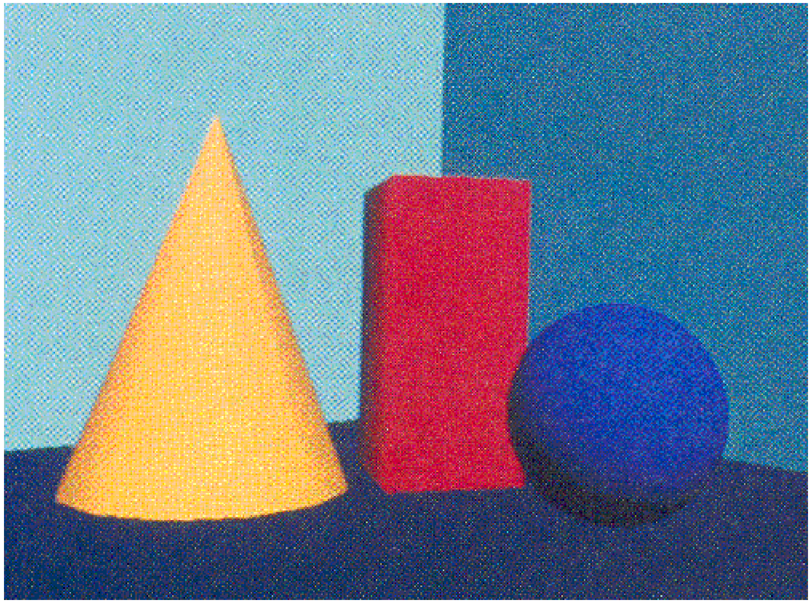


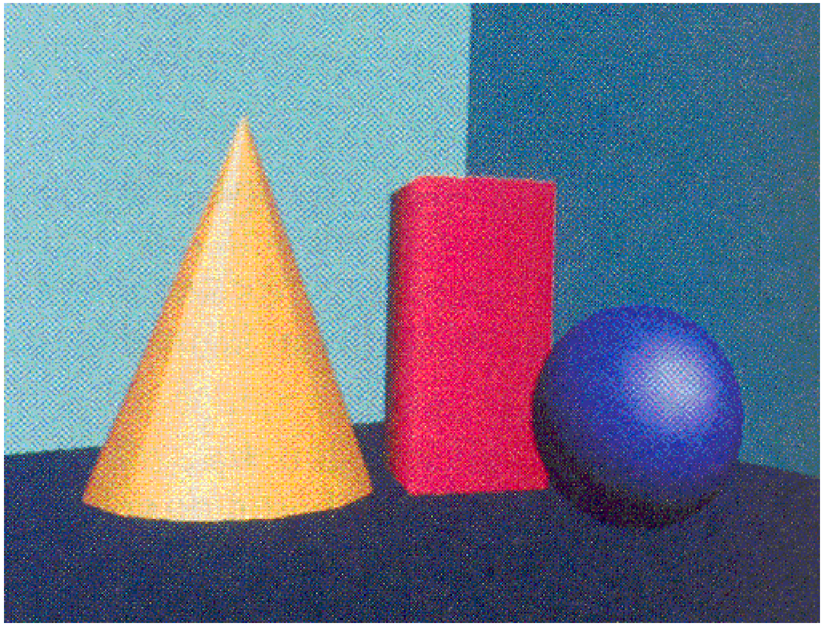
- Important for shiny model surfaces
  - How to model shininess ?

- Take into account viewer position !

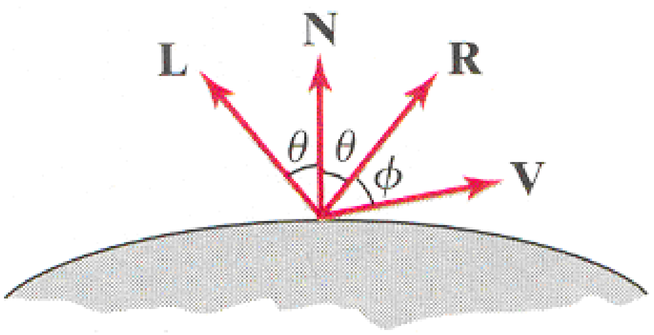
- Unit vectors !

Phong Model – Specular reflection



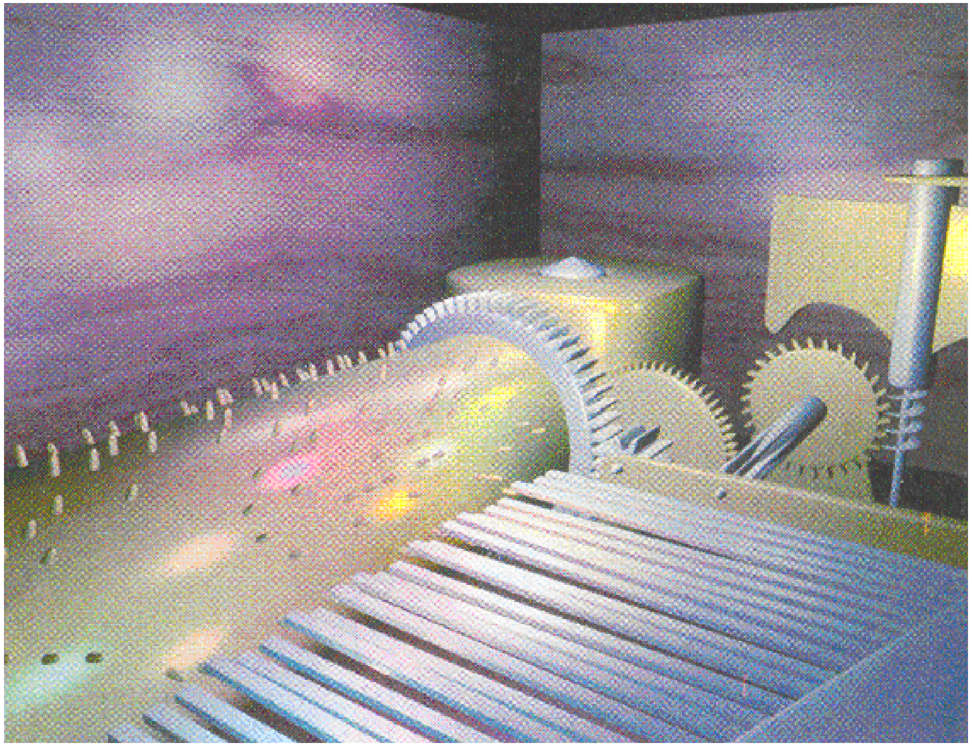


Phong Model – Specular reflection



$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{V} \cdot \mathbf{R})^{n_s}, & \text{if } \mathbf{V} \cdot \mathbf{R} > 0 \quad \text{and} \quad \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \text{if } \mathbf{V} \cdot \mathbf{R} < 0 \quad \text{or} \quad \mathbf{N} \cdot \mathbf{L} \le 0 \end{cases}$$

More than one light source



$$I = k_a I_a + \sum_{l=1}^{n} I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

Illumination and shading

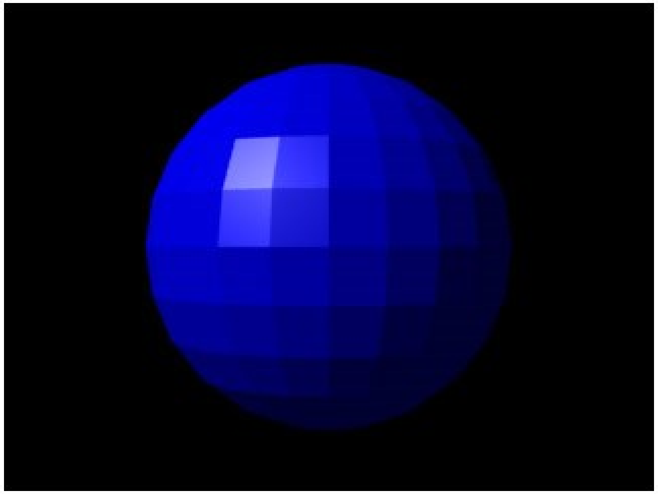
- How to optimize?
  - Fewer light sources
  - Simple shading method

- BUT, less computations mean less realism
  - Wireframe representation
  - Flat-shading
  - Gouraud shading
  - Phong shading

Flat shading

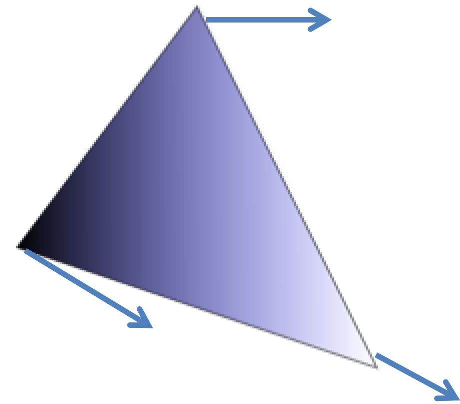
- For each polygon:
- Applies the illumination model just once
- All pixels have the same color

- smooth objects seem “blocky”
- It is fast



FLAT SHADING

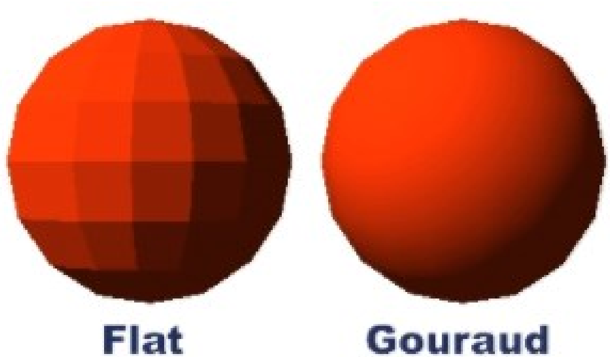
Gouraud shading



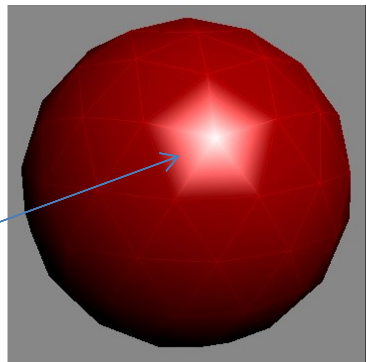
Apply the illumination model at vertices

- For each triangle:
- Applies the illumination model at each vertex
- Interpolates color to shade each pixel

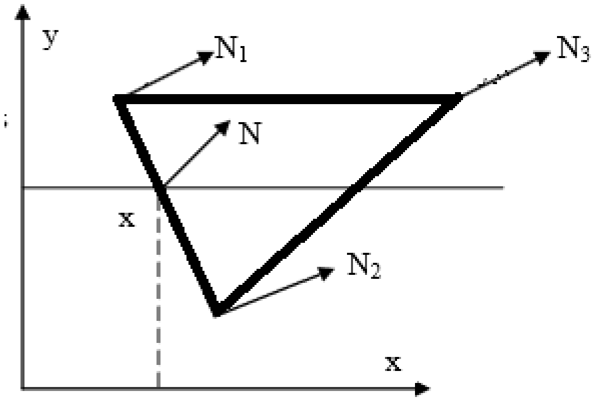
- It provides better results than flat shading
- But highlights are not rendered correctly



highlight

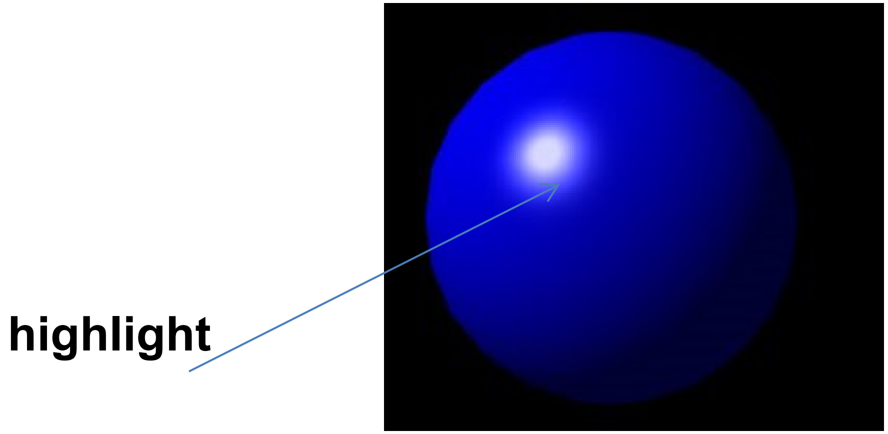


Phong shading

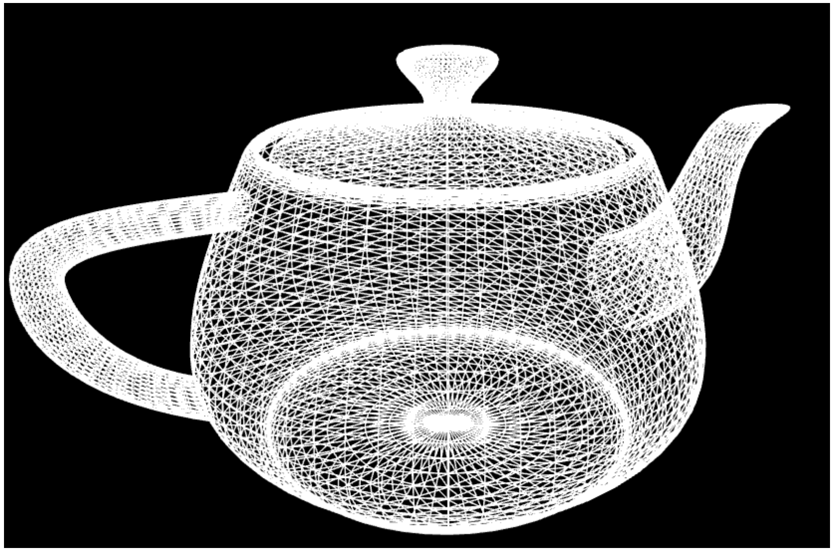


- Interpolates normals across rasterized polygons
- computes pixel colors based on the interpolated normals

- It provides better results than Gouraud shading
- But is more time consuming



Wire frame



Flat shading

Gouraud shading

Phong shading

https://threejs.org/examples/#webgl_geometry_teapot

Some reference books

- D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3rd Ed., Addison-Wesley, 2004
- E. Angel and D. Shreiner, *Introduction to Computer Graphics,* 6th Ed., Pearson Education, 2012
- J. Foley et al., *Introduction to Computer Graphics*, Addison-Wesley, 1993
- Hughes, J., A. Van Dam, et al., *Computer Graphics, Principles and Practice*, 3rd Ed., Addison Wesley, 2013