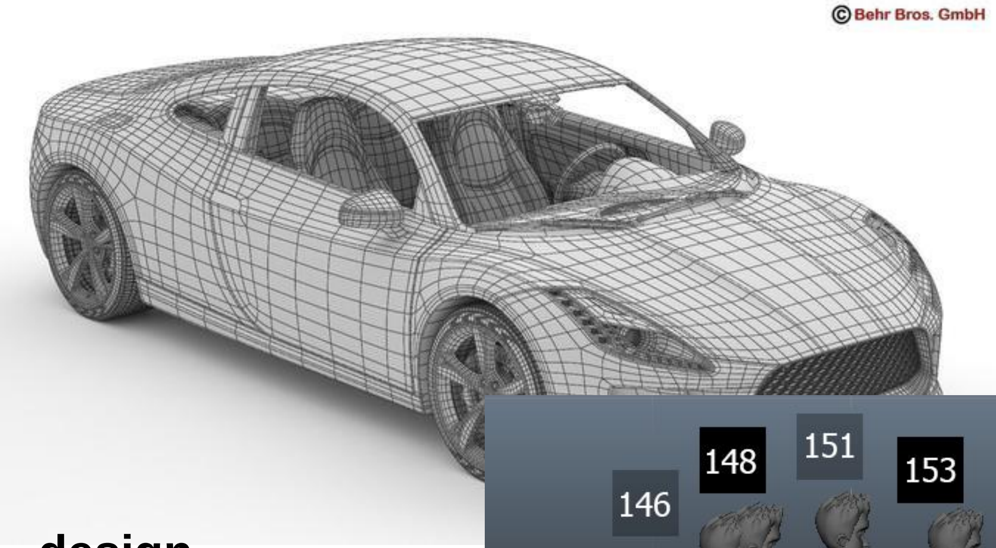
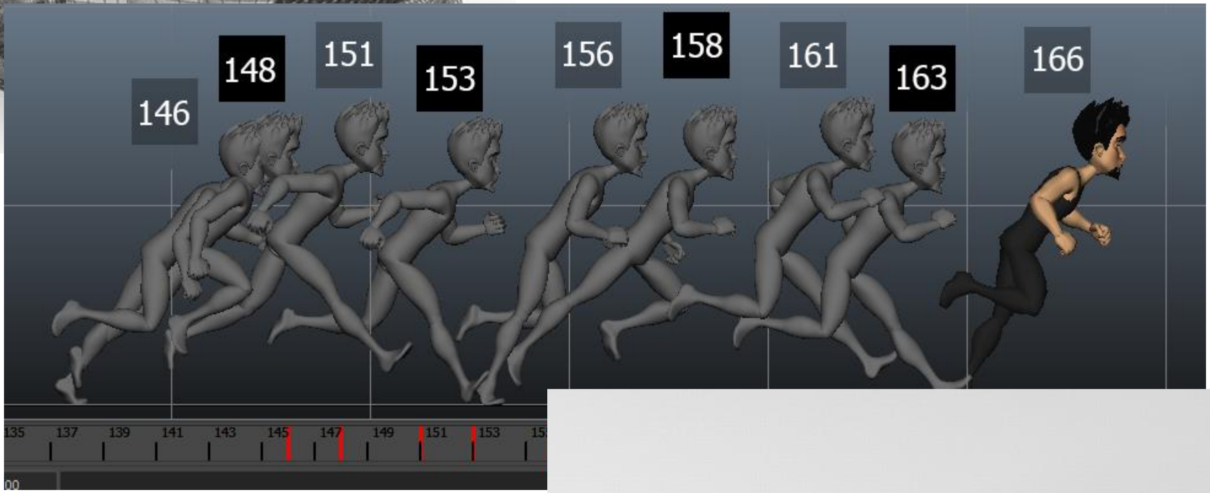
3D GRAPHICS



design



animate

render

Computer Graphics

- 3D animation movies

Computer Graphics

- Special effects

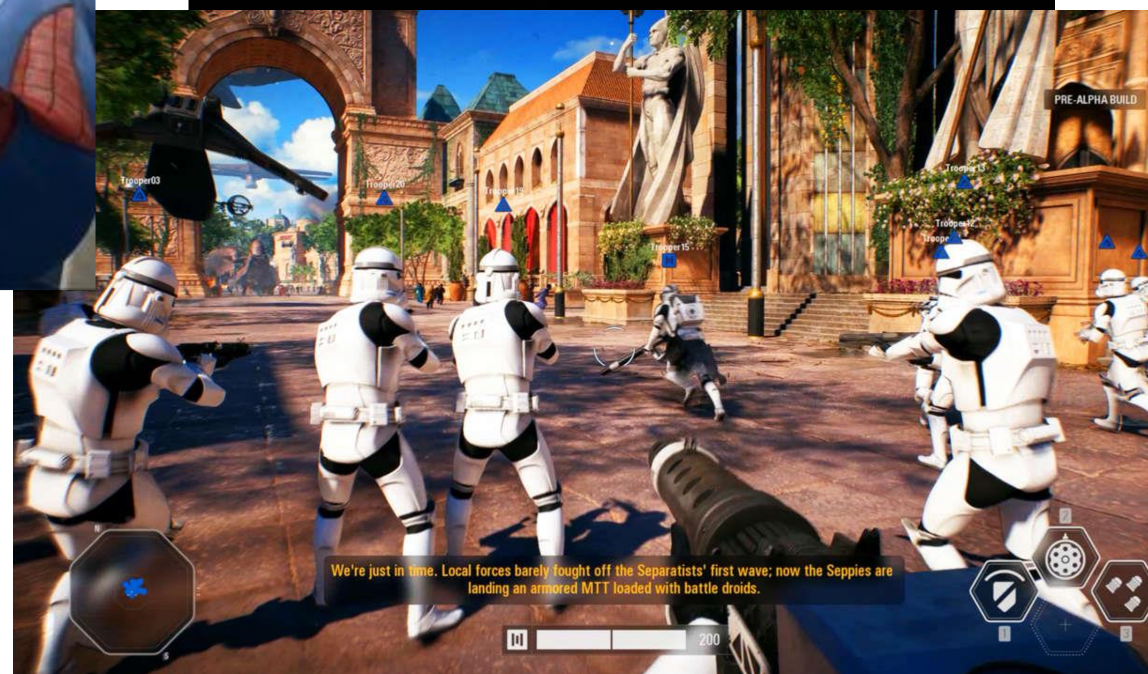



Computer Graphics

- Advertising

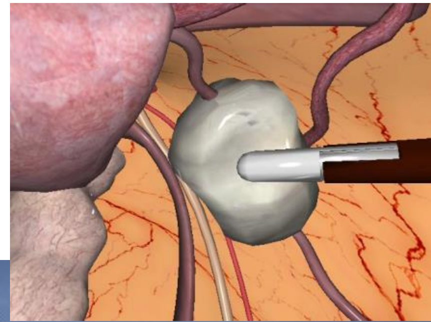
Computer Graphics

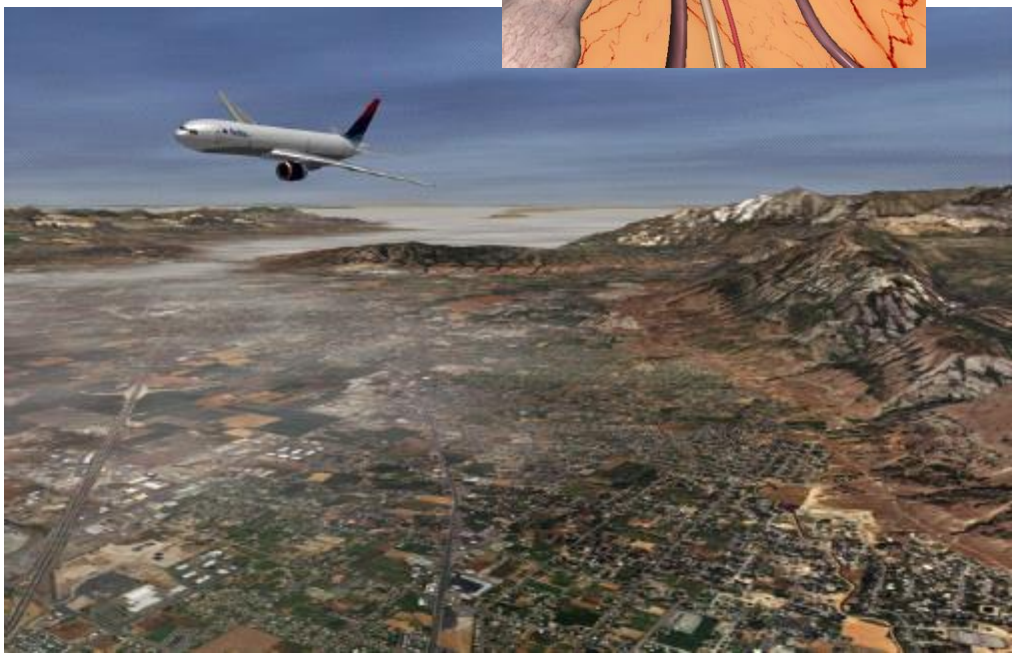
- Games



Computer Graphics

- Simulations & serious games





Computer Graphics

- Computer Aided Design (CAD)

Computer Graphics

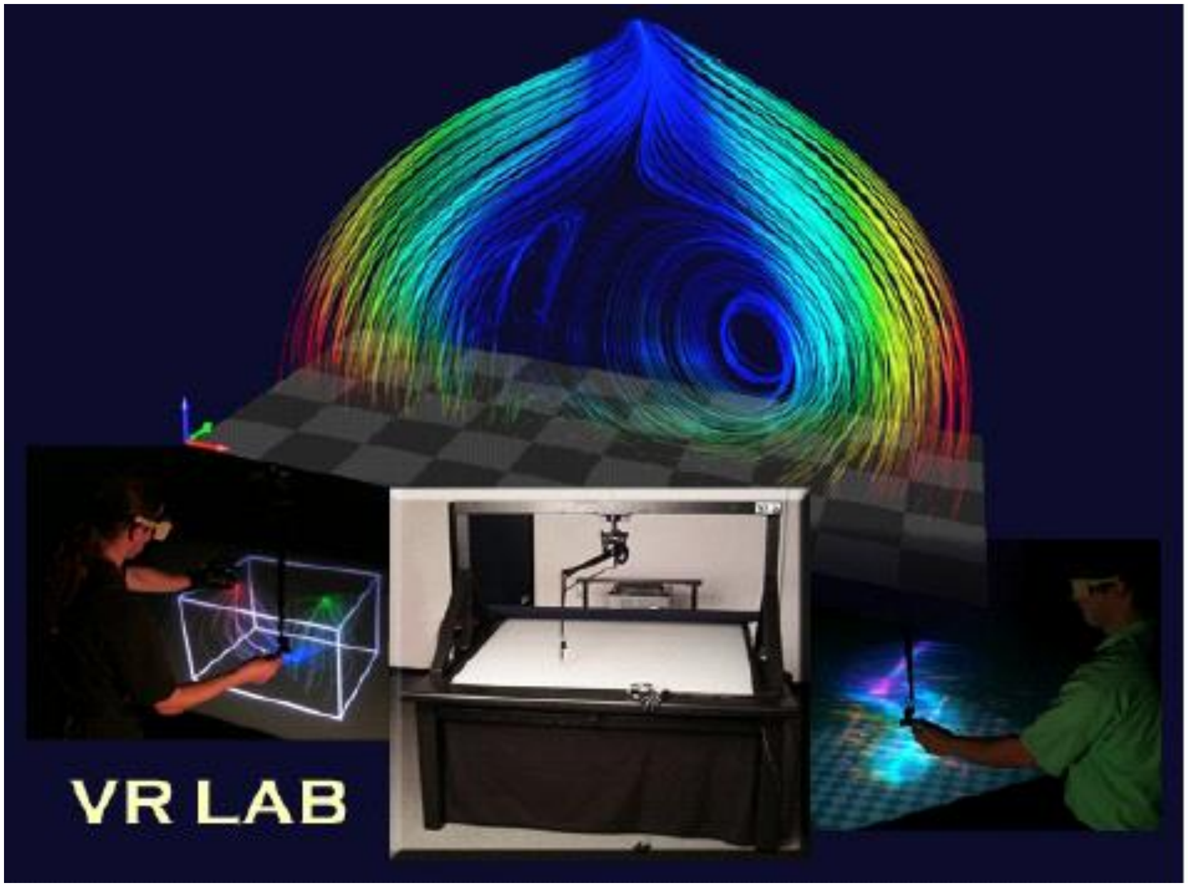
- Architecture

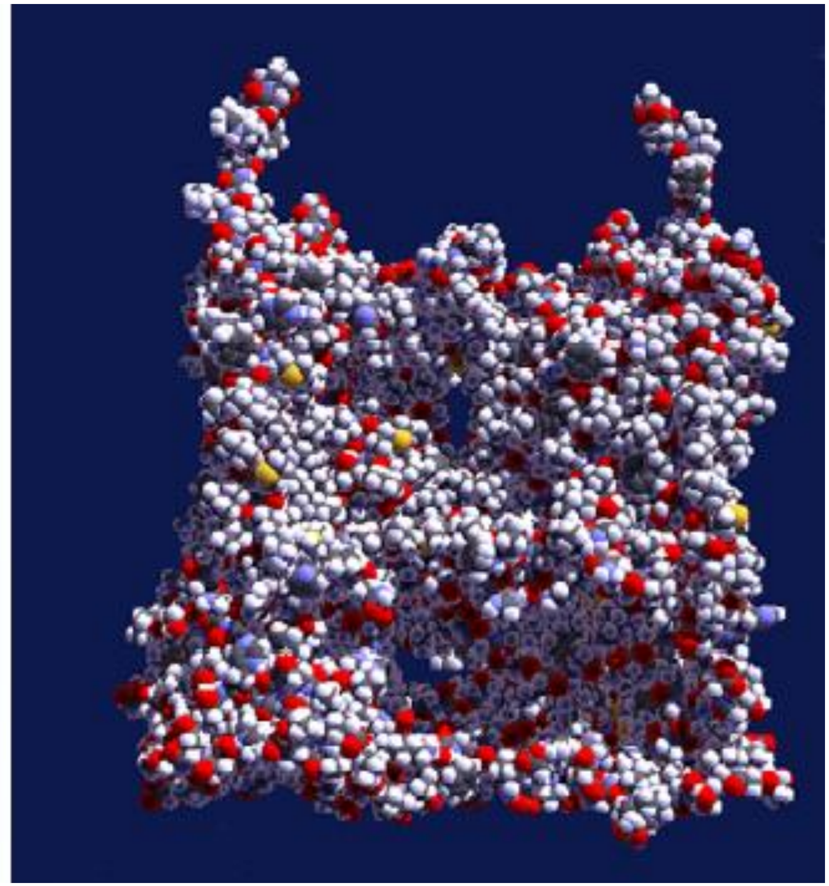
Computer Graphics

- Virtual/augmented reality

Computer Graphics

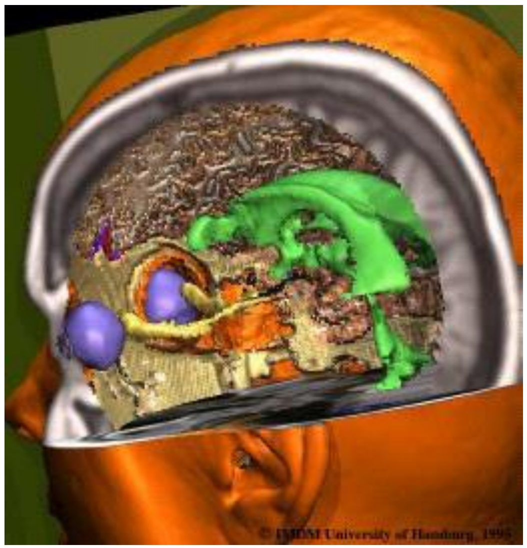
- Visualization

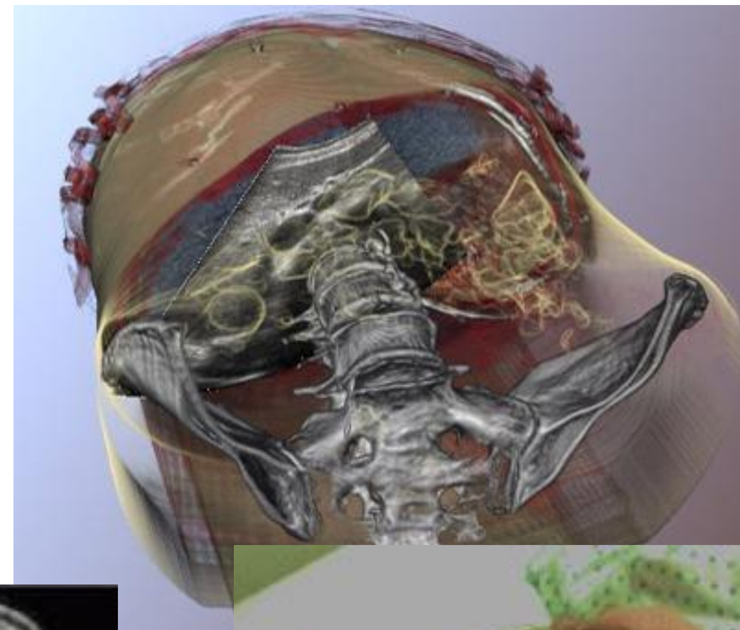


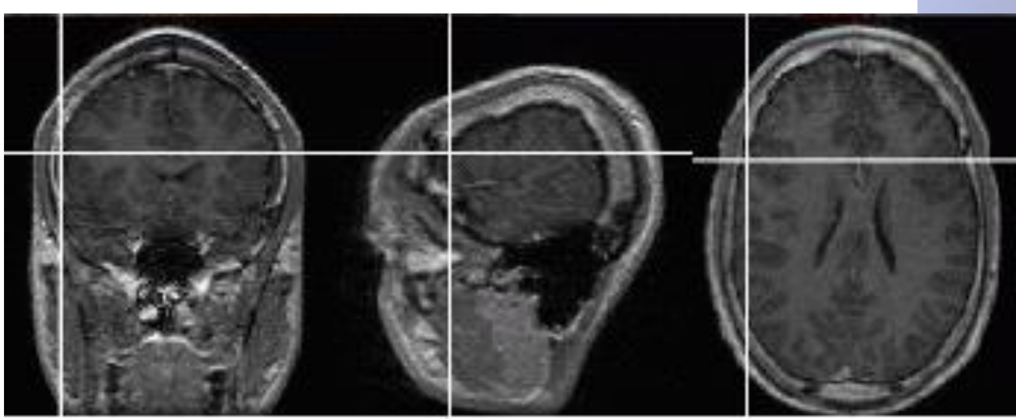


Computer Graphics

- Medical imaging

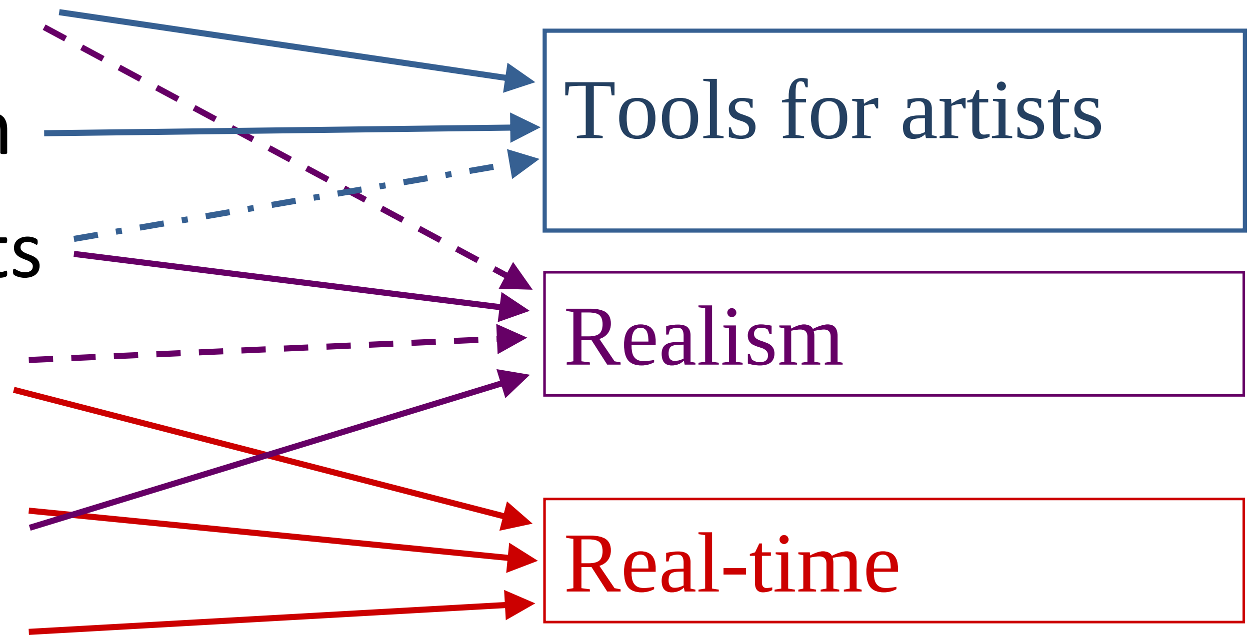






Computer Graphics

- Design
- 3D animation
- Special effects
- Games
- Simulators
- Visualization



What you will learn

1. Overview of Computer Graphics (for engineering & research)
   - Modeling : create 3D geometry
   - Animation : move & deform
   - Rendering : 3D scene → image

2. How the basic techniques work
3. Practice with OpenGL (+Python)
4. Case studies : Practical problems
   - How to choose & combine existing techniques (TD)

*What you will **not** learn*

- Mathematical bases (algebra, geometry!)
- Advanced CG techniques in detail
- Advanced Programming the Graphics Hardware (GPU)
- Artistic skills or Game design
- Use of existing software

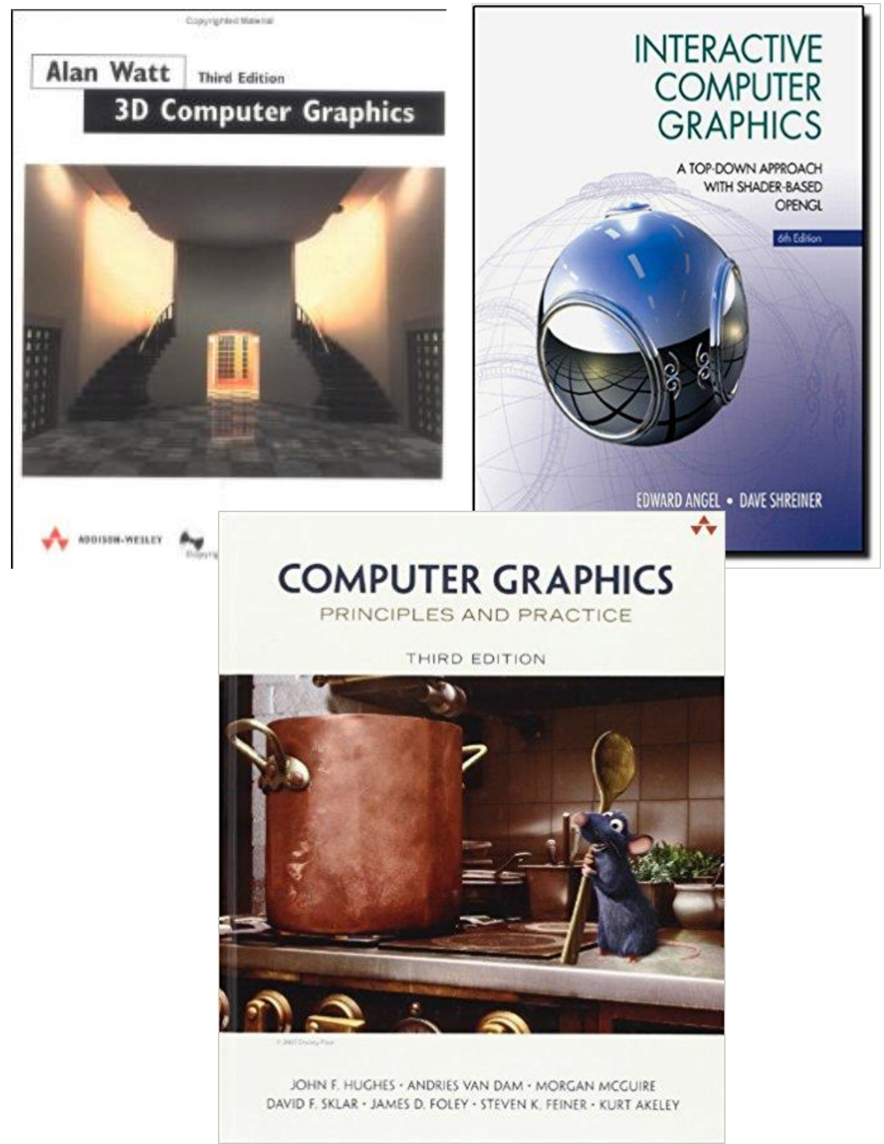
  (CAD-CAM, 3D Studio Max, Maya, Photoshop, etc)

Text books

No book required

References

- *3D Computer Graphics (Watt, 2000)*
- *Interactive Computer Graphics (Angel & Shreiner, 20).*
- *Computer Graphics: Principles and Practices (Hughes et al. 2013)*



Organization & overview

1.5h CTD (course & exercises) + 1.5h TP (lab)

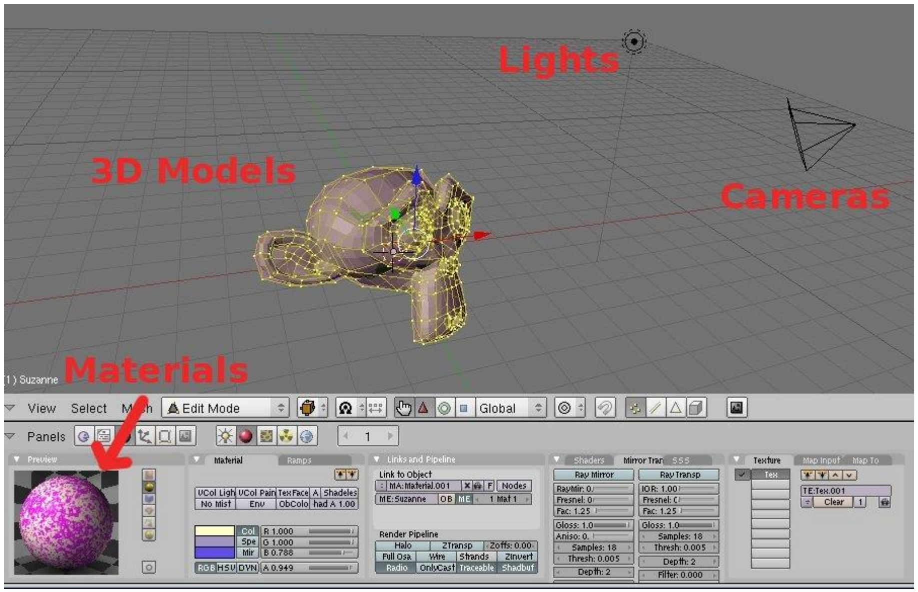
- *Part 1: Basic techniques*
  1. Modeling: geometric representations, hierarchical modeling
  2. Rendering: illumination, shading, textures
  3. Animation: Keyframing, skinning, collisions
- *Part 2: Introduction to advanced methods*

  3 advanced courses on modeling/rendering/animation

Evaluation: [0,5 Exam + 0,5 OpenGL project]

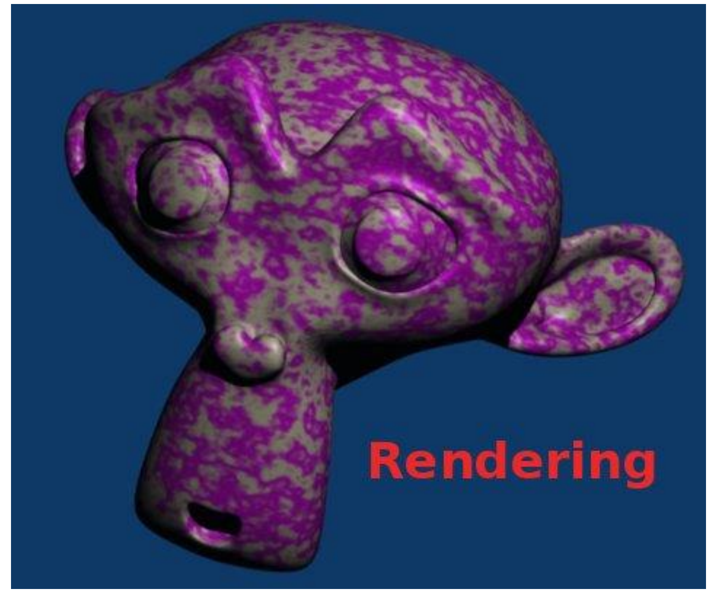
1: graphics (projective) pipeline

Inputs



?

Output



Implemented in the Graphics Hardware (real time)

Used by OpenGL

1: graphics pipeline

Required: transformations using 4x4 matrices

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$V' = M \quad V$$

1: graphics pipeline

Required: transformations using 4x4 matrices

- "Homogeneous coordinates"
- Needed for projective transformations
- Cartesian to projective coordinates: w=1
- Projective to Cartesian coordinates: divide by w

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$V' = M \; V$$

1: graphics pipeline

sub-case: affine transformations

- Start with cartesian coordinates: w=1
- Keep the last line to 1
- The last matrix column enables to express translations!

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$V' = M \quad V$$

Typical affine transformations

Translation

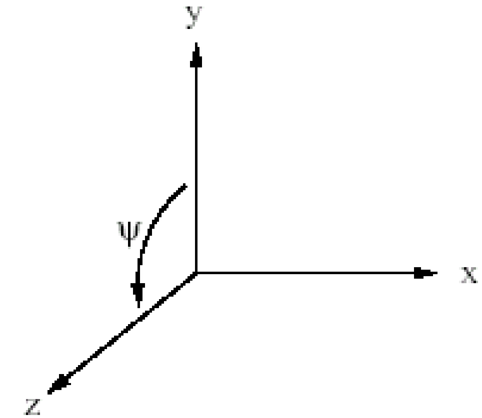
$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale

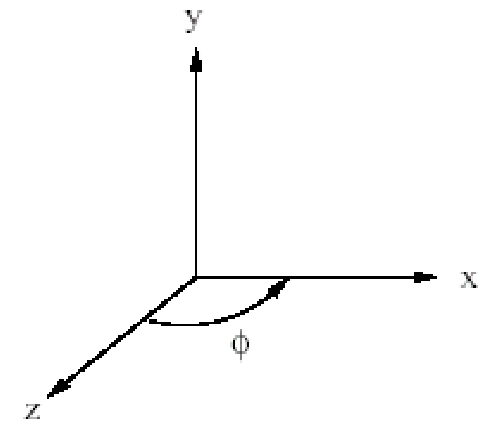
$$T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotations (Euler angles)

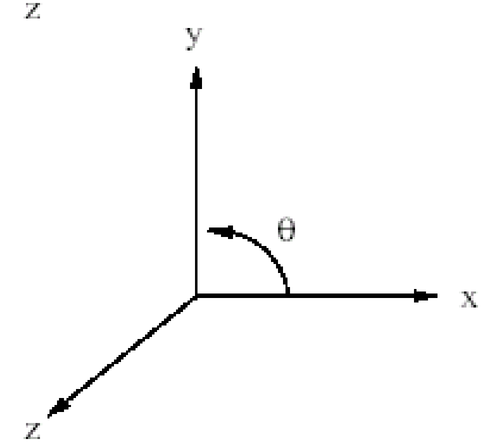
$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi & 0 \\ 0 & \sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$R_y = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

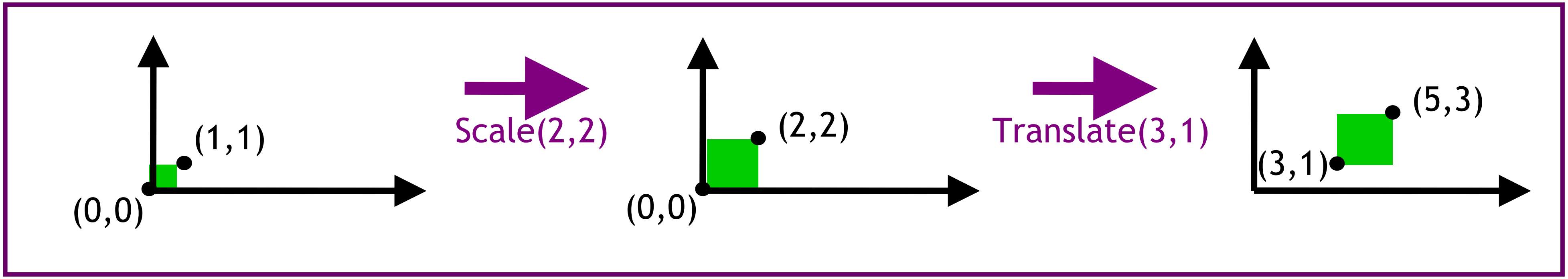


$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$R = R_z \cdot R_y \cdot R_x ,$$

2D example
Composition of scale and translate

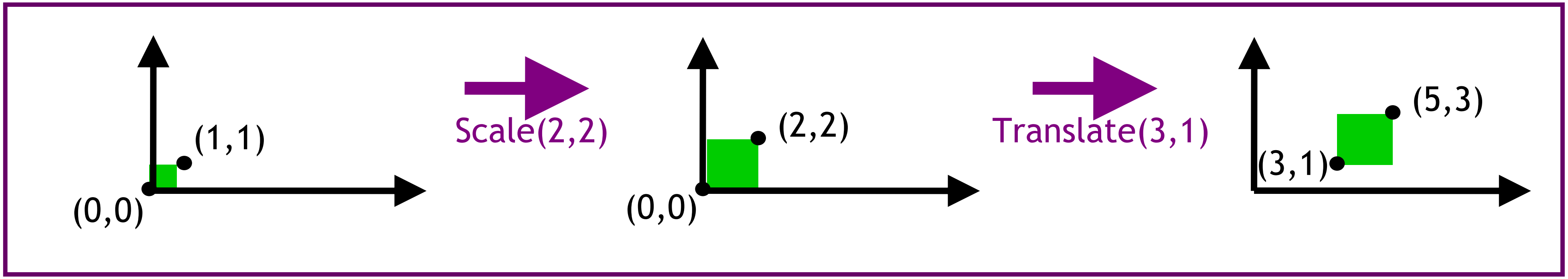


Multiplication of matrices : p' = T (S p) = TS p

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

TD part

1. *Is the order of transformation important?*



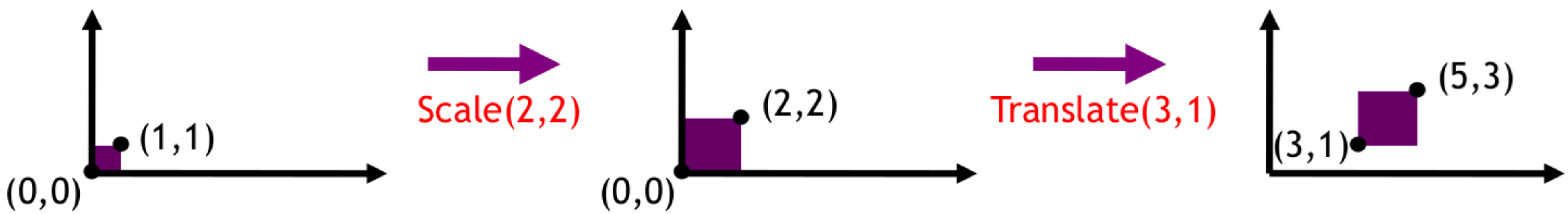
Exercise : (T S) p =? (S T) p

$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} & & \\ & ? & \\ & & \end{pmatrix}$$

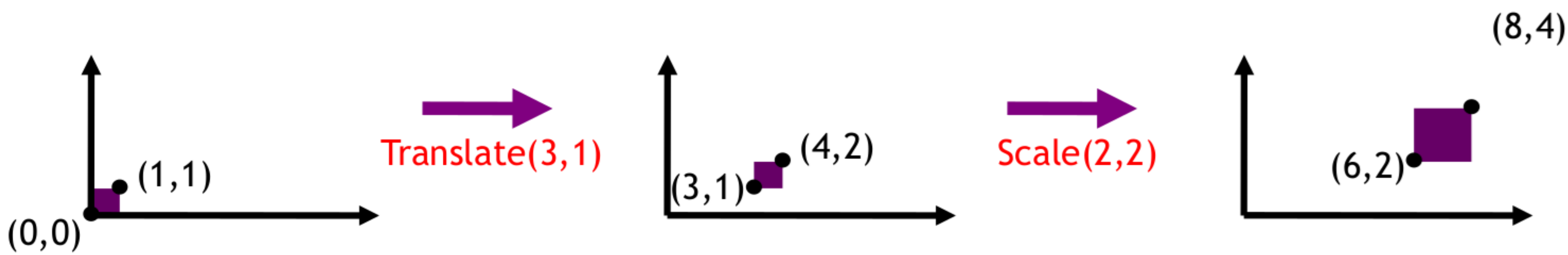
Draw the transformed polygon(s) using ST and TS

1. Solution: transformations not commutative!!

Scale then translate : p' = T (S p) = TS p

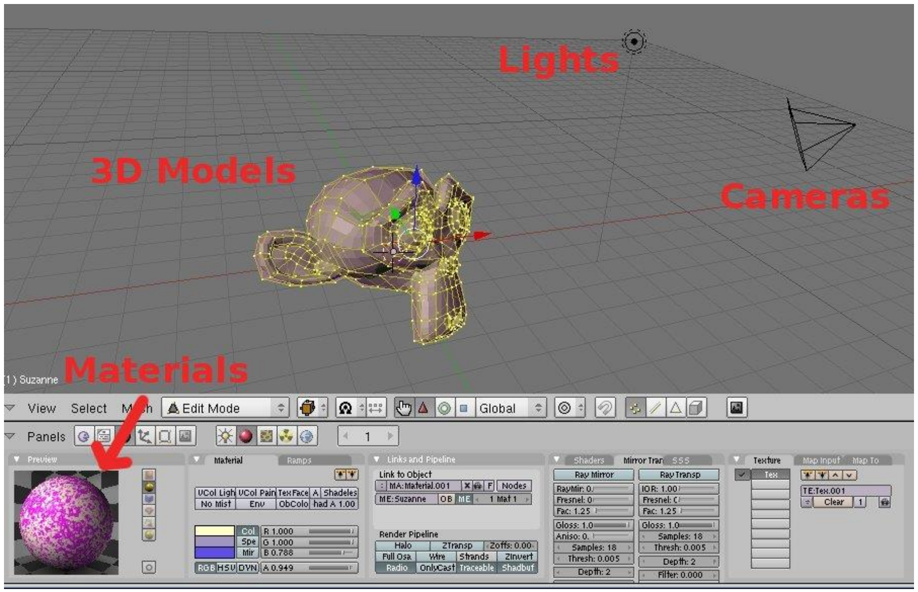


Translate, then scale : p' = S (T p) = ST p



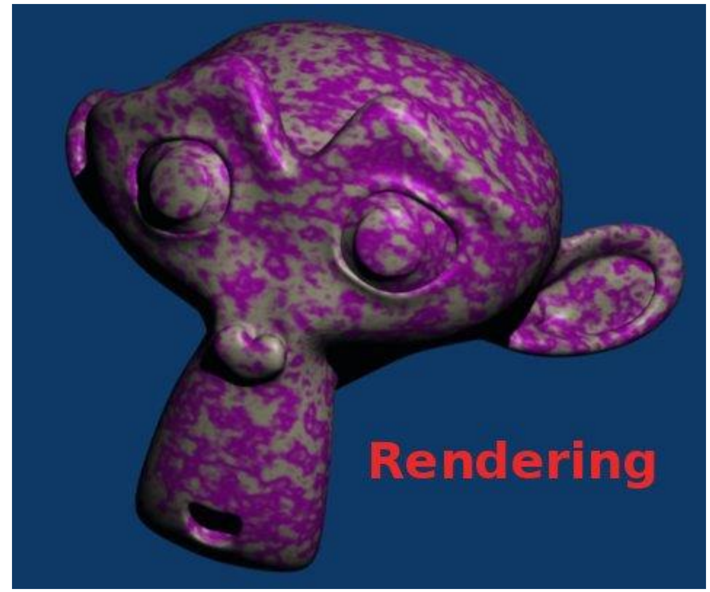
1: graphics pipeline

Inputs



?

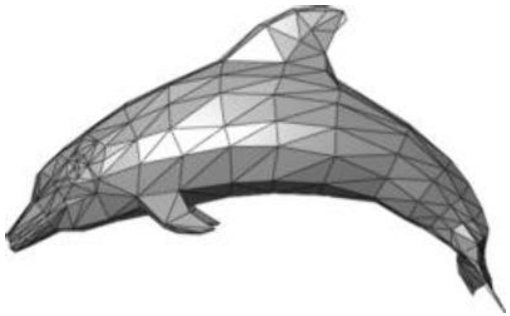
Output



Ok, back to our main question!

1: graphics pipeline

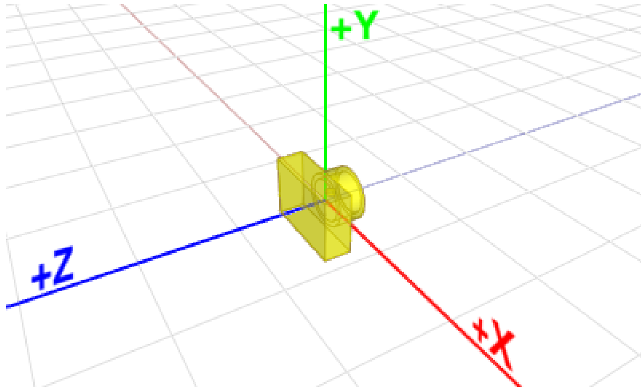
Lets consider that we already have the input data (ignore materials and lights for now)



Mesh, composed of triangle faces (v1,v2,v3)
Each vertex contains 3 coords (x,y,z)
defined in the local/model frame

v1 = (x1,y1,z1)
v2 = (x2,y2,z2)
…

(more in next lecture)

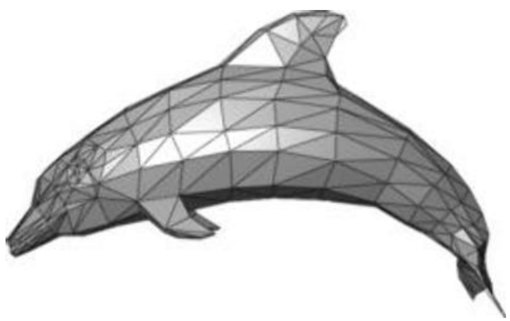


Camera, composed of 4x4 matrices

(more in a few minutes)

1: graphics pipeline

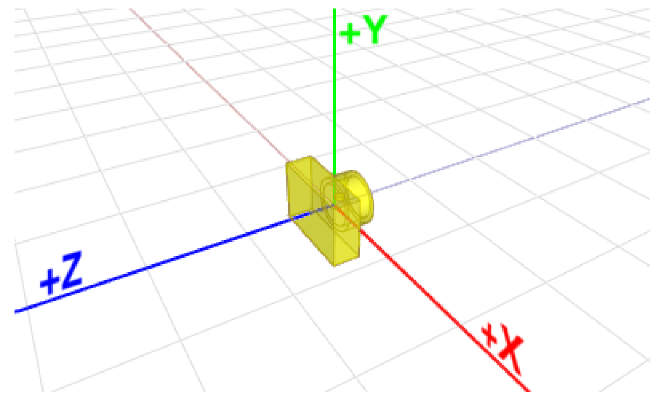
Lets consider that we already have the input data (ignore materials and lights for now)



Mesh, composed of triangle faces (v1,v2,v3)
Each vertex contains 3 coords (x,y,z)
defined in the local/model frame

v1 = (x1,y1,z1)
v2 = (x2,y2,z2)
…

(more in next lecture)



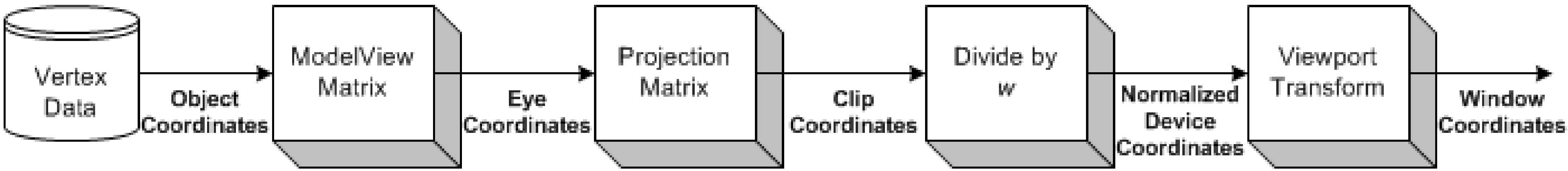
Camera, composed of 4x4 matrices

(more in a few minutes)

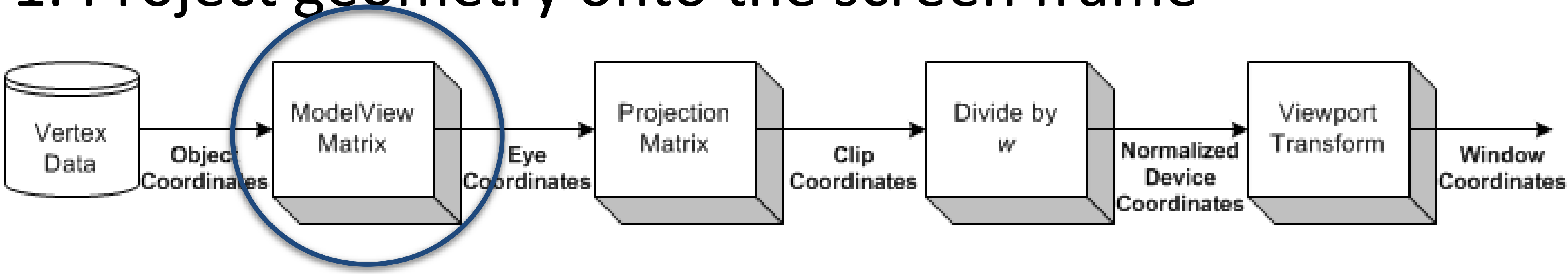
Creating an image from these data can be done in 4 steps!

1: graphics pipeline

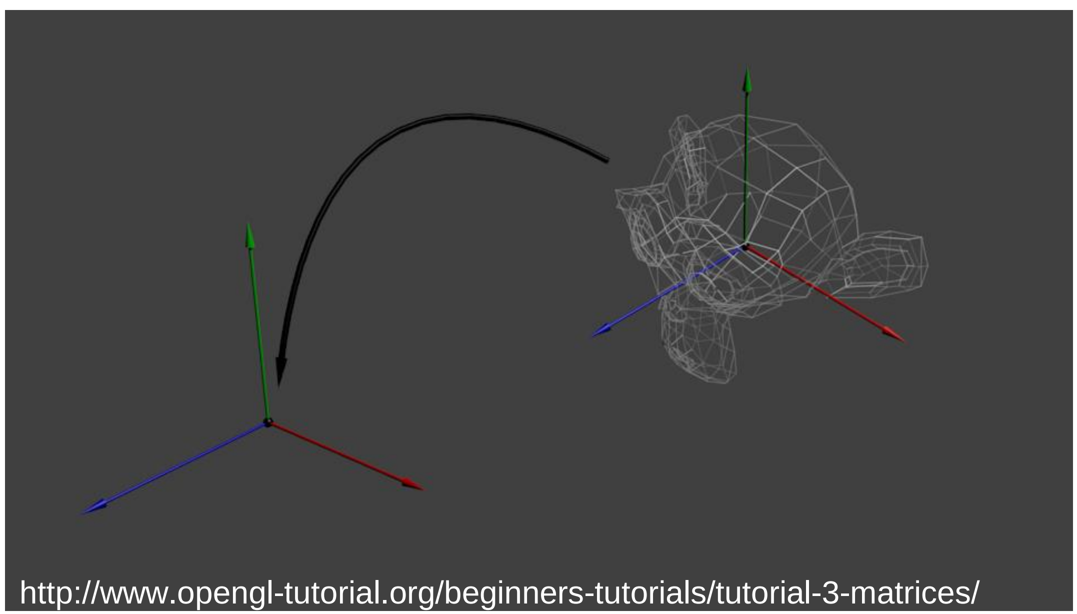
1. Project geometry onto the screen frame



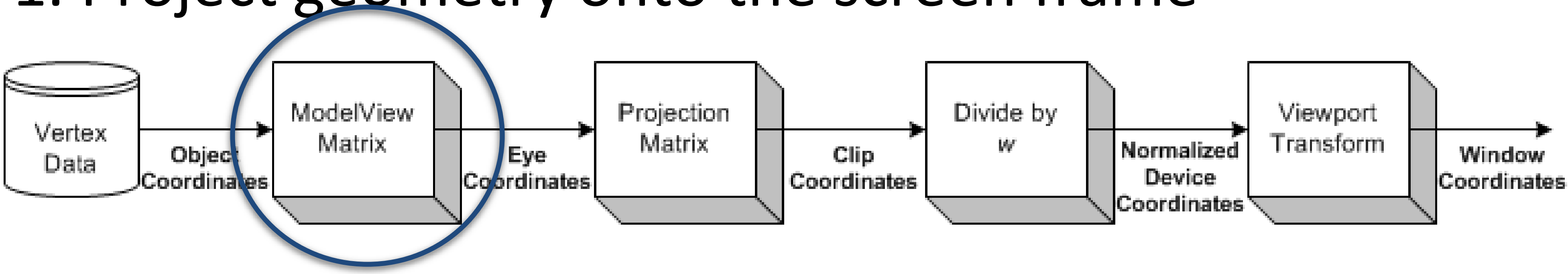
1. Project geometry onto the screen frame



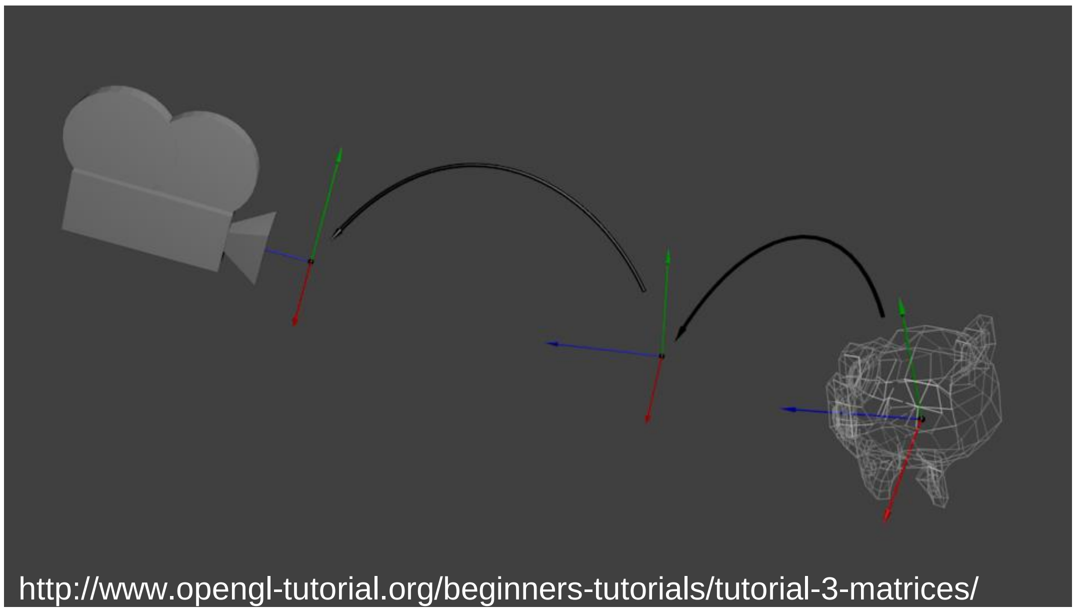
- From model to world space (4x4 matrix)



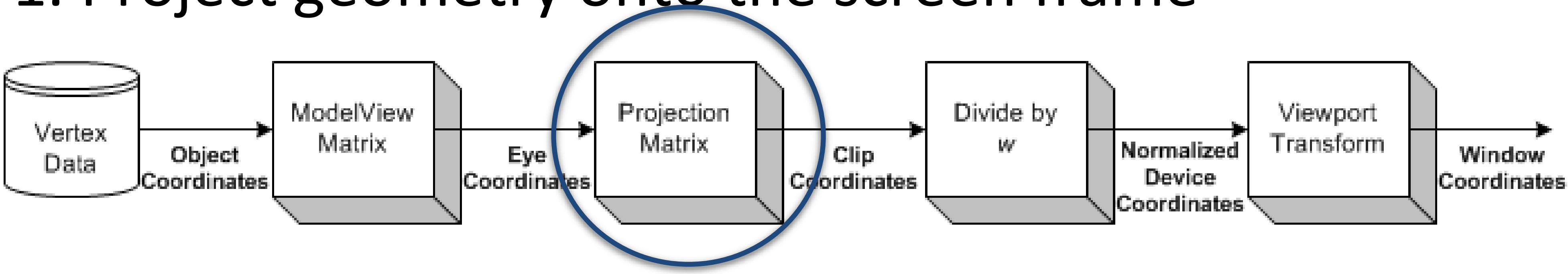
1. Project geometry onto the screen frame



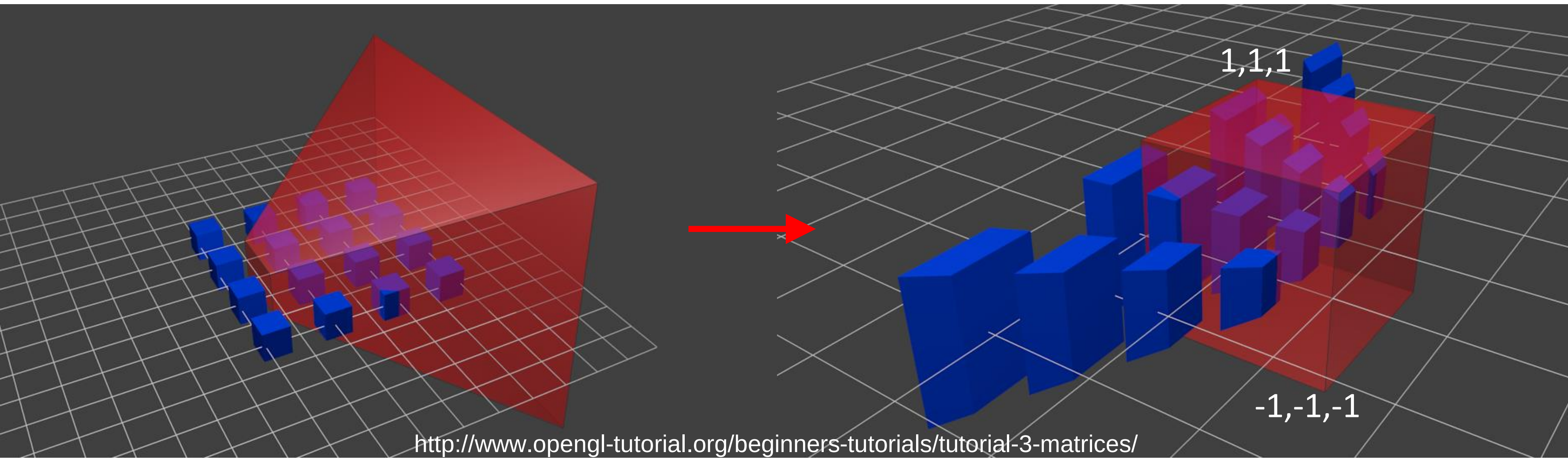
- From model to world space (4x4 matrix)
- From world to camera space (4x4 matrix)



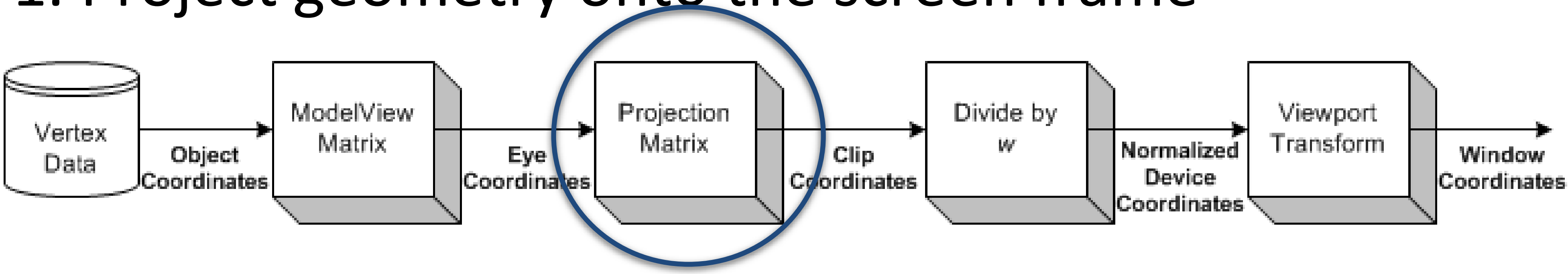
1. Project geometry onto the screen frame



- From model to world space (4x4 matrix)
- From world to camera space (4x4 matrix)
- From camera to “screen” space (4x4 matrix)

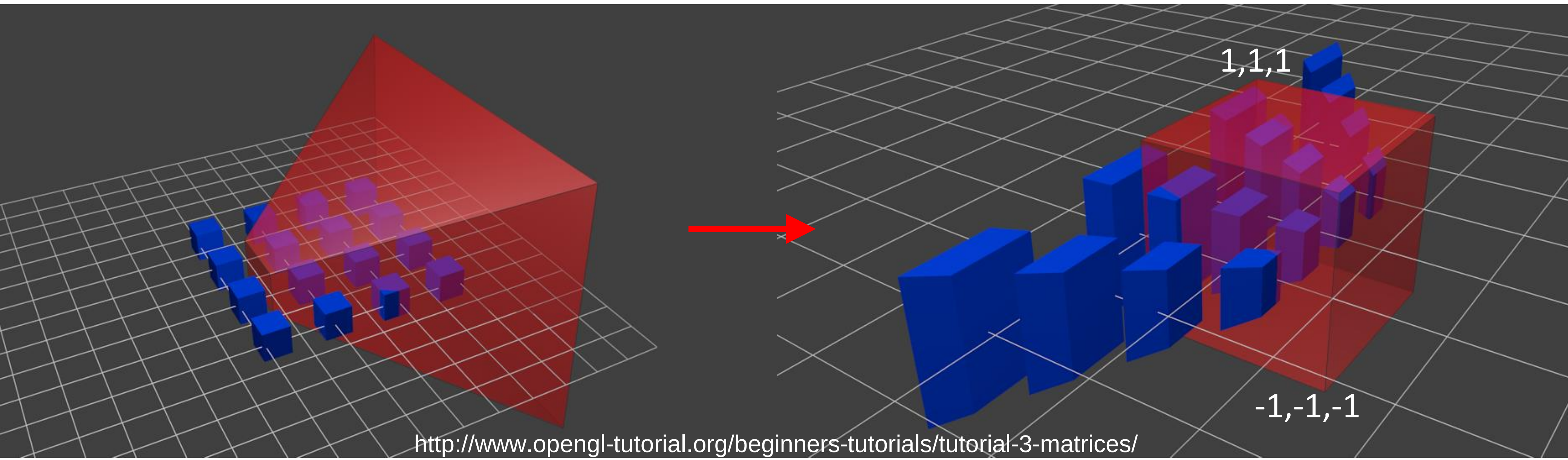


1. Project geometry onto the screen frame

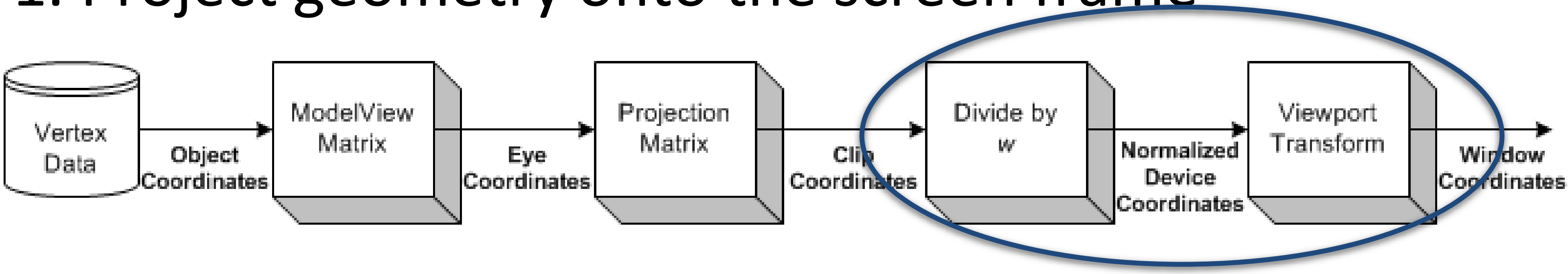


- From model to world space (4x4 matrix)
- From world to camera space (4x4 matrix)
- From camera to “screen” space (4x4 matrix)

can be concatenated!



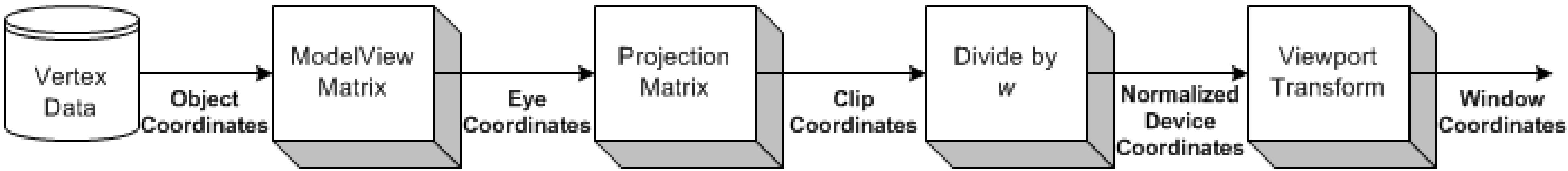
1. Project geometry onto the screen frame



- Back to cartesian coordinates...
- … and to screen coordinates

1: graphics pipeline

1. Project geometry onto the screen frame

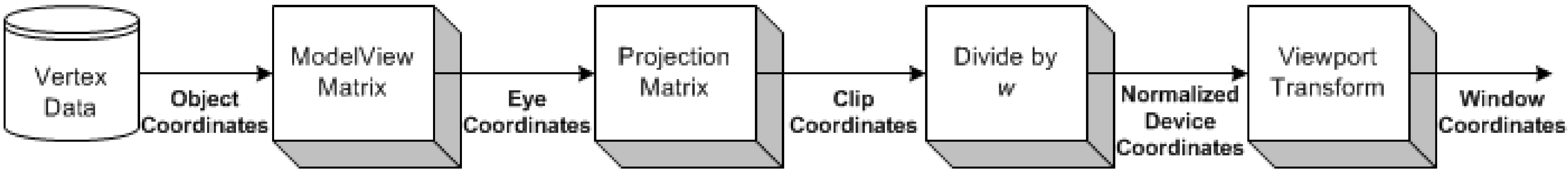


Questions:

- How to (intuitively) define a camera matrix?

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{array}{c} \begin{array}{cccc} \boldsymbol{r} & \boldsymbol{u} & \boldsymbol{v} & \boldsymbol{t} \end{array} \\ \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

1. Project geometry onto the screen frame



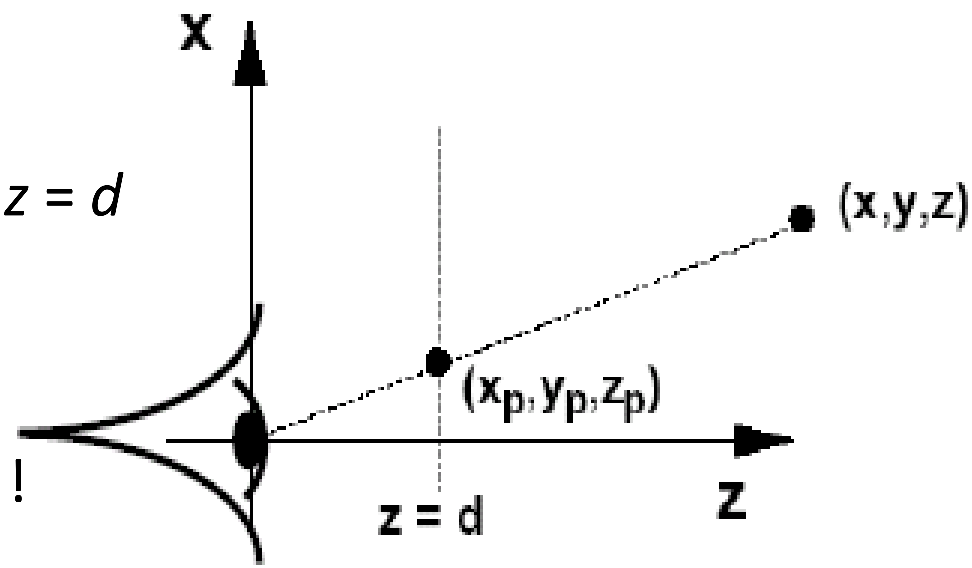
Questions:

- Finding a projection matrix

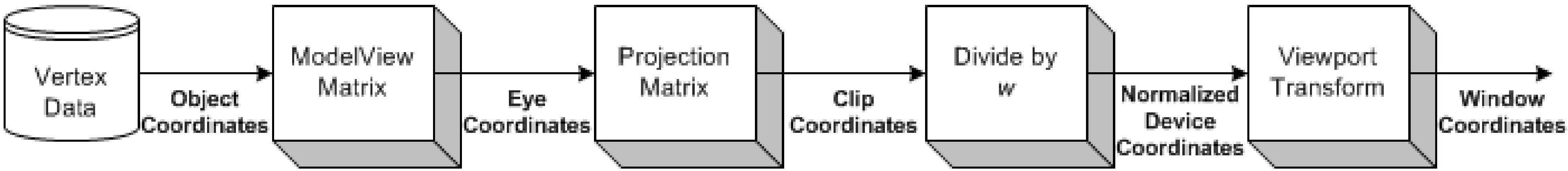
Project all points to the image plane $z = d$

1. Compute x_p and y_p
2. Find the 4x4 matrix M needed

M should be independent from x, y, z !



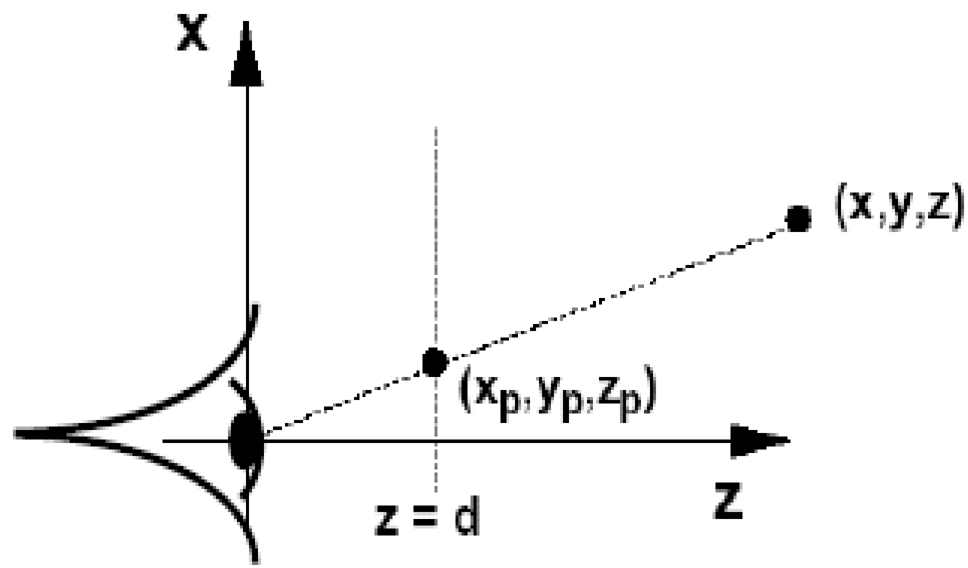
1. Project geometry onto the screen frame



Thales theorem

xp/zp = x/z, with zp=d

yp/zp = y/z, with zp=d



$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}$$

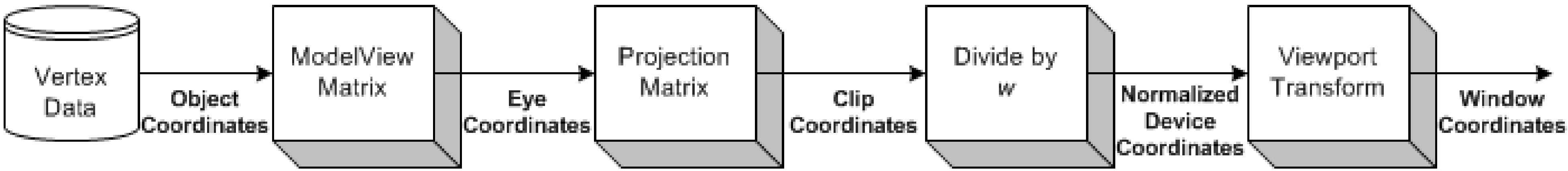
$$y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}$$

$$z_p = d$$

$\longrightarrow$ Find a transform M that divides by z/d

1: graphics pipeline

1. Project geometry onto the screen frame



Solution:

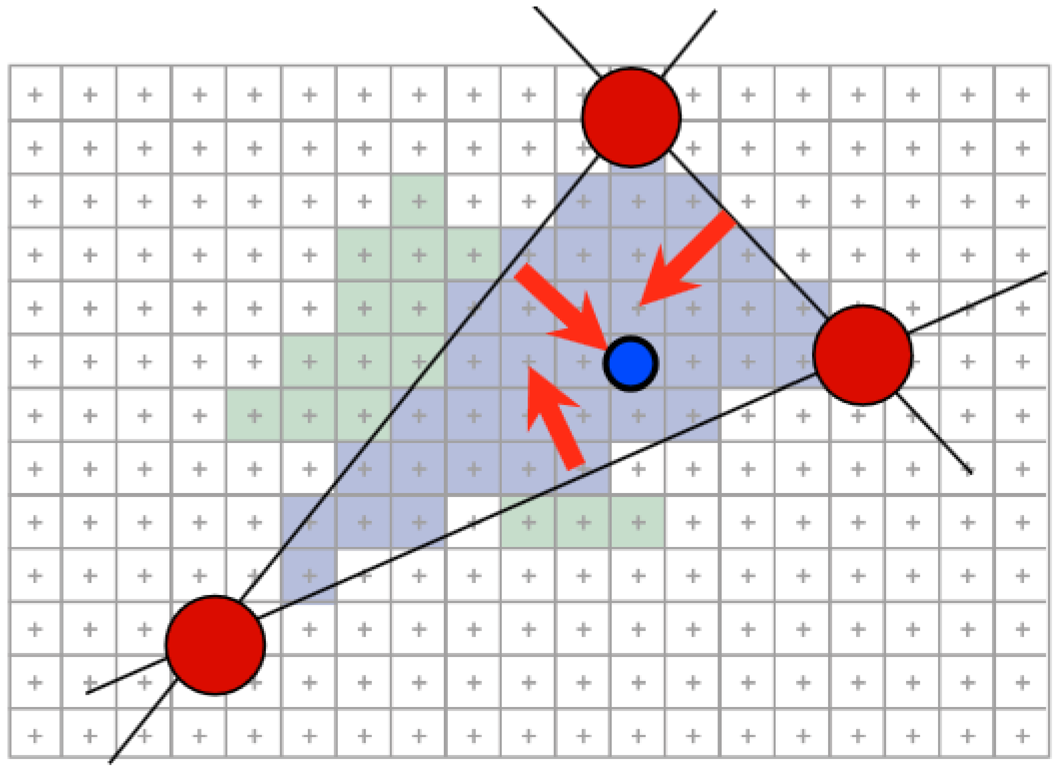
- A transform that divides all coords by z/d → set w to z/d

homogenize

$$\text{New 3D point} = \begin{pmatrix} x * d / z \\ y * d / z \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} = \underbrace{\boxed{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}}}_{\text{Projective transform}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

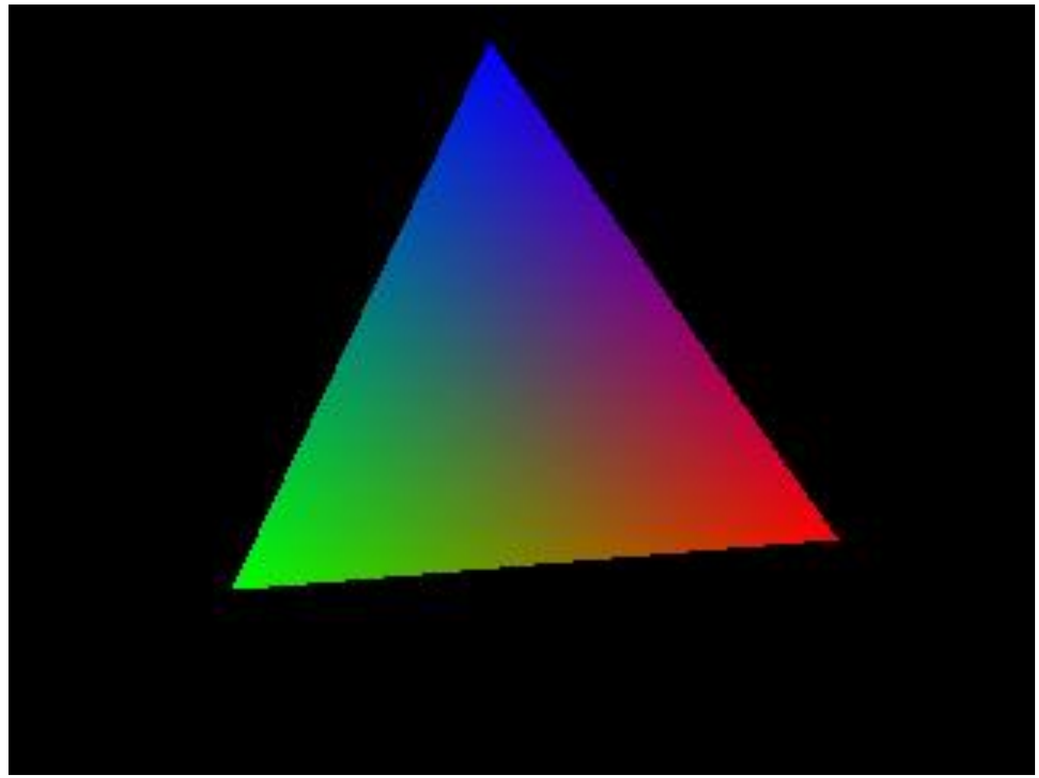
1. Project geometry onto the screen frame
2. Rasterize triangles

- For each pixel
  - test 3 edge equations
  - if all pass, draw
- Interpolate vertex data
  - using barycentric coords
  - positions/normals/colors/...



1. Project geometry onto the screen frame
2. Rasterize triangles

- For each pixel
  - test 3 edge equations
  - if all pass, draw
- Interpolate vertex data
  - using barycentric coords
  - positions/normals/colors/...

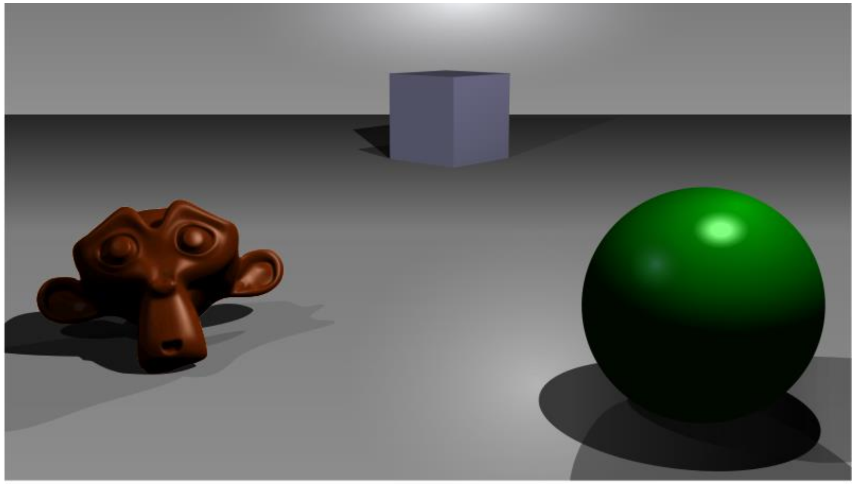


1: graphics pipeline

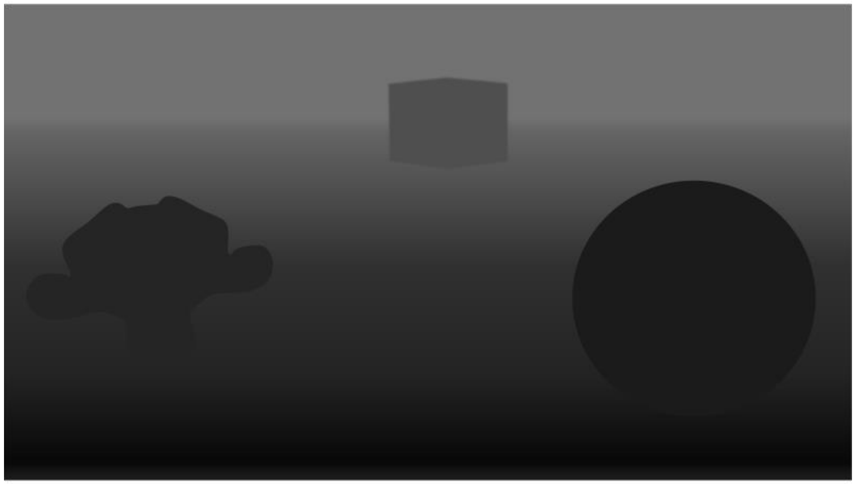
1. Project geometry onto the screen frame
2. Rasterize triangles
3. Visibility test

- For each pixel
  - Store min distance to camera
  - in a “Z-Buffer”

- if new_z<Z-Buffer[x,y]
  - Z-Buffer[x,y] = new_z
  - Framebuffer[x,y] = computePixelColor()



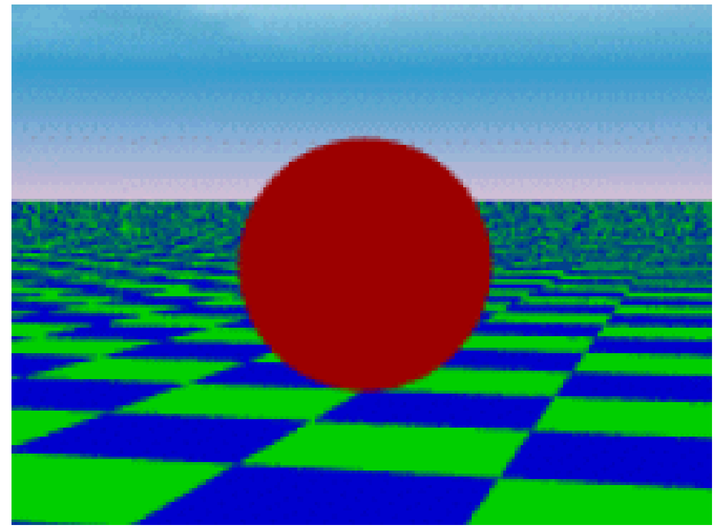
A simple three-dimensional scene



Z-buffer representation

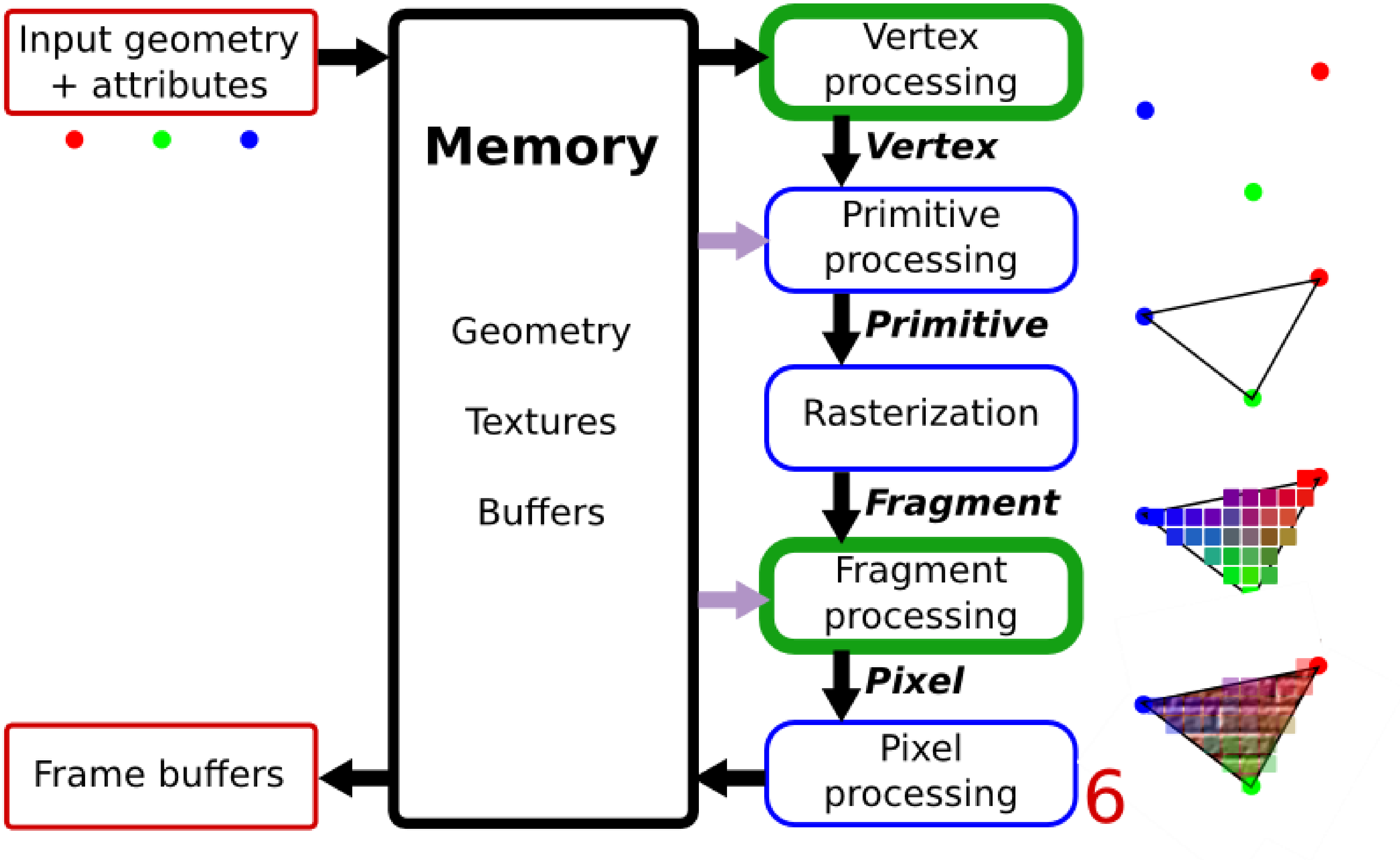
1: graphics pipeline

1. Project geometry onto the screen frame
2. Rasterize triangles
3. Visibility test
4. Compute pixel color



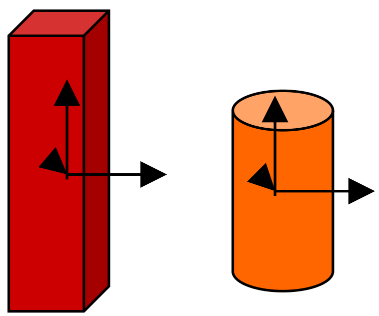
- Require more than a simple uniform color
  - This is where we will use material and lighting properties
  - to be continued… in next lectures

OpenGL pipeline

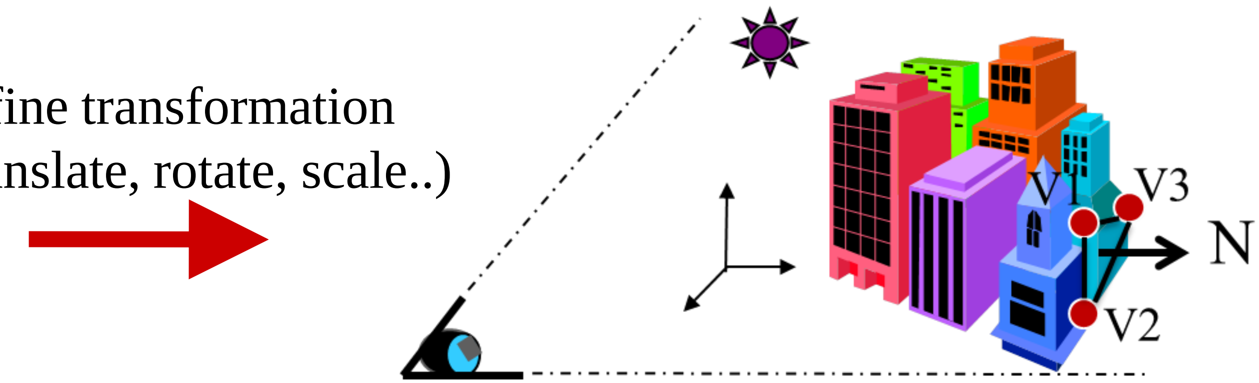


Bonus

2. Transforming normal vectors?



Affine transformation
(translate, rotate, scale..)



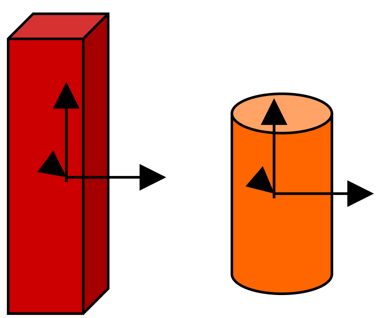
Exercise: How should we transform normal vectors?

Advice : think of the difference between

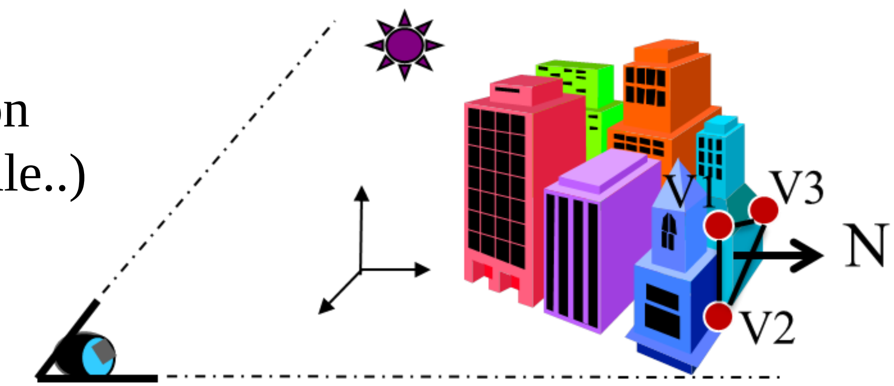
- affine transformation of points
- linear transformations of vectors

Bonus

2. Transforming normal vectors?



Affine transformation
(translate, rotate, scale..)



Apply the same transform to vectors **except translations**!

Vectorial transform

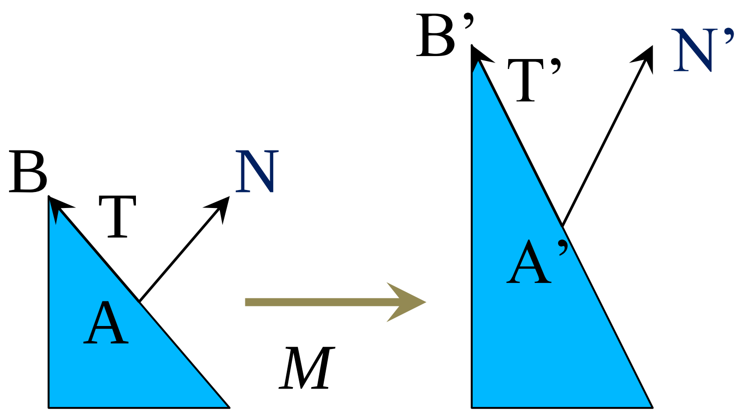
$$\begin{pmatrix} x' \\ y' \\ z' \\ 0 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

Same 4x4 matrix
Set w=0 for vectors

Bonus

2. Transforming normal vectors?

PB: the normal to a triangle does not remain a normal after scaling!



It works for tangent vectors:
T= B-A   T'=MT = MB-MA=B'-A'

How should we transform normals?
Defined by: N.T= 0

- We are looking for G such that GN.MT = 0

  $\leftrightarrow$ $N^T G^T . M T = 0$ …. so if $G^T.M= Id$, it works!

- We choose: $G = (M^{-1})^T$ (for orthogonal matrices, G=M)