Fundamentals of Computer Graphics

Lecture 6. OpenGL pipeline (geometry)

Yong-Jin Liu

liuyongjin@tsinghua.edu.cn

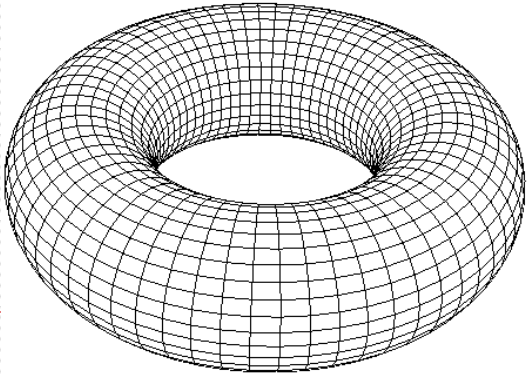
Outline

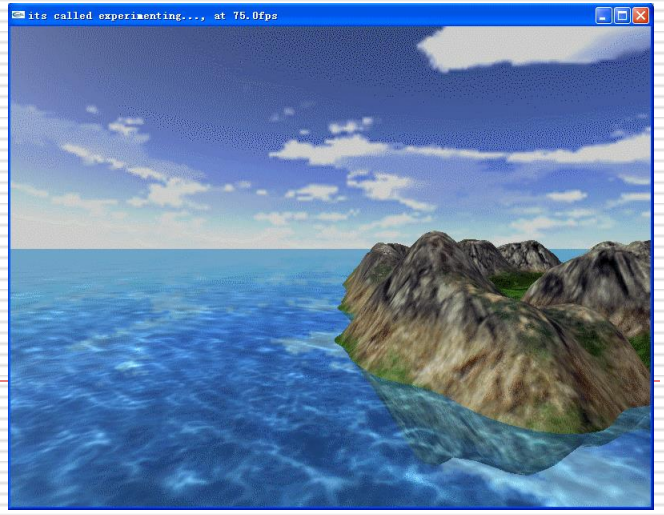
OpenGL rendering pipeline

- Geometry (object transformation)

  Learn how to carry out transformation in OpenGL

- Color (lighting, texture, etc.)

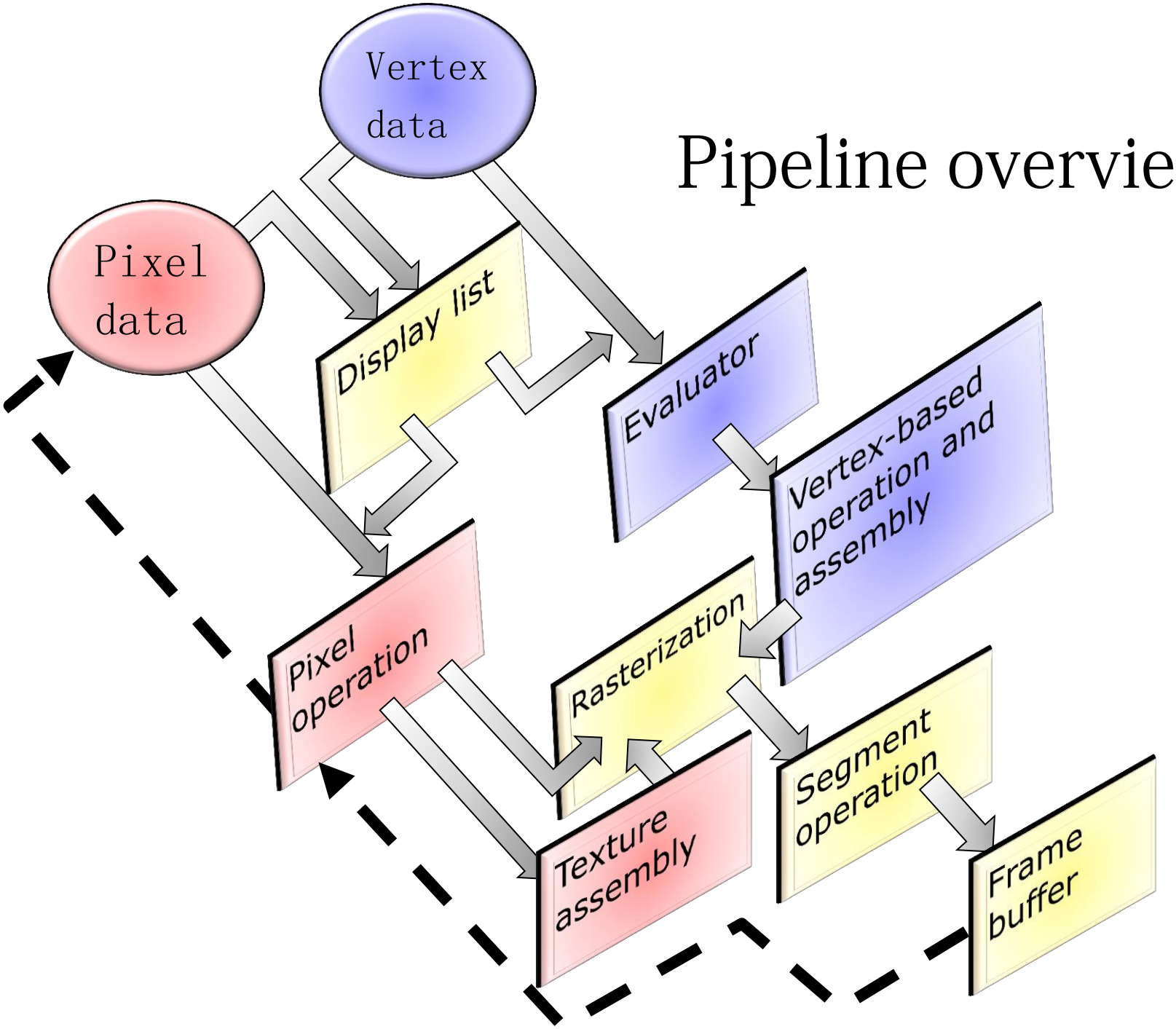




Outline

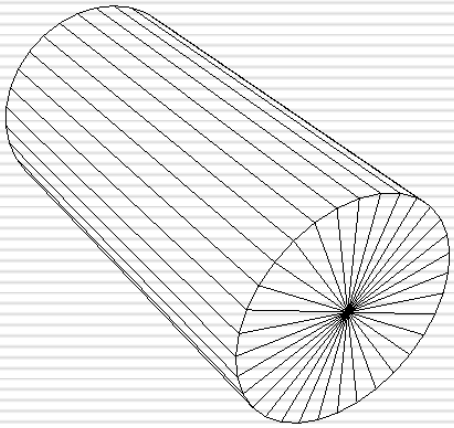
OpenGL rendering pipeline

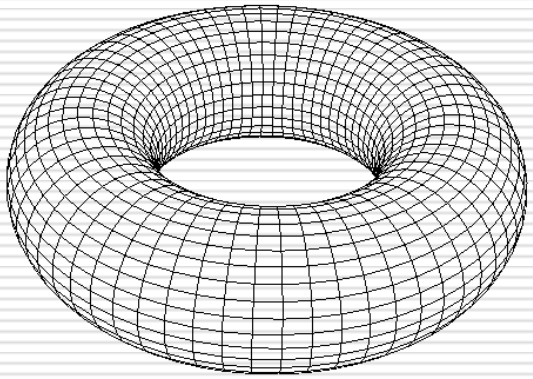
Pipeline overview

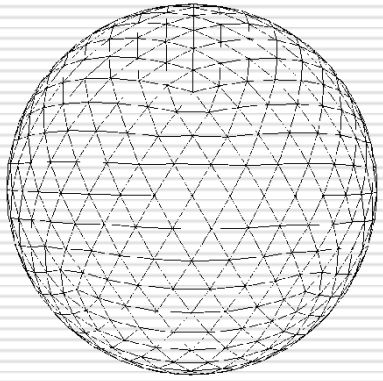


Geometry transformation pipeline

1. 3D object stored in its own (local) coordinate system

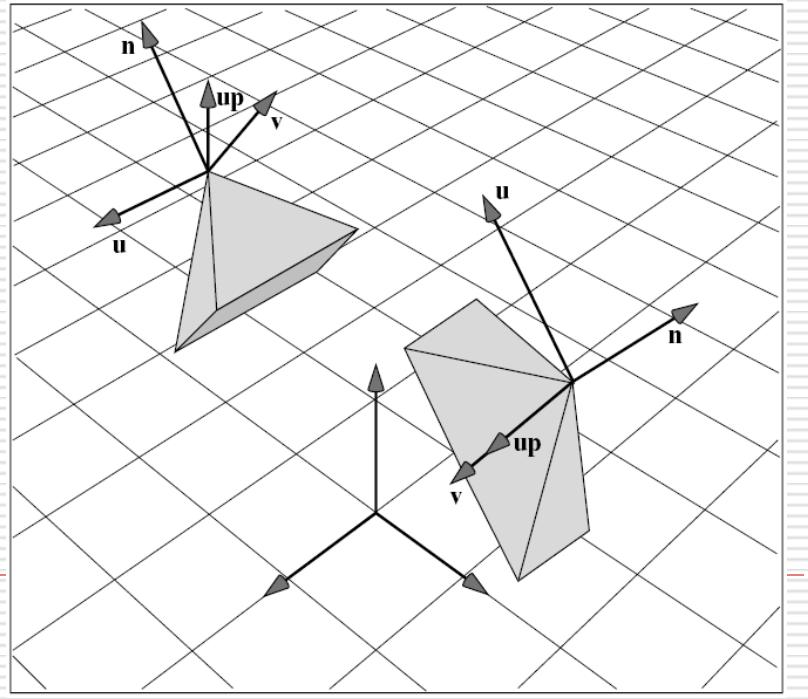






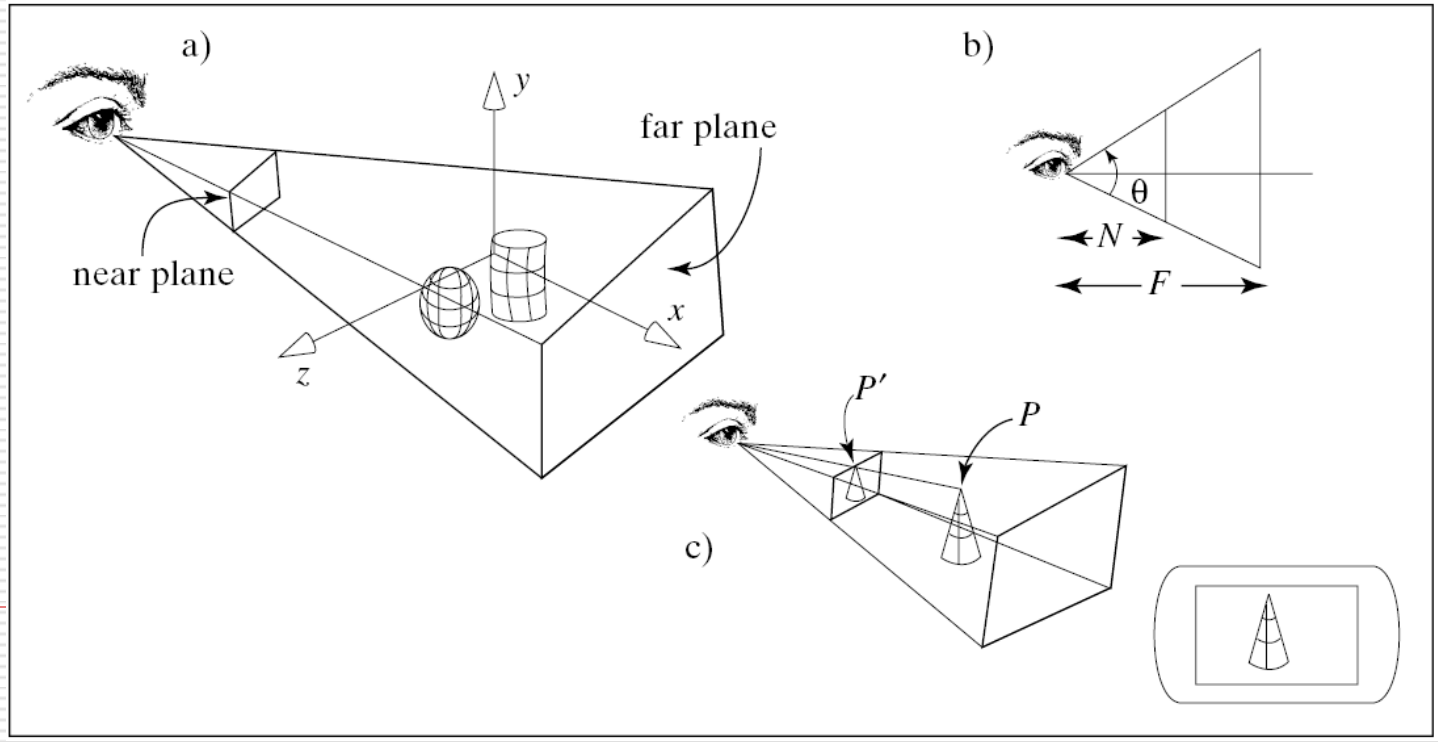
Geometry transformation pipeline

1. 3D object stored in its own (local) coordinate system
2. We need to assembly lots of 3D objects in a global scene



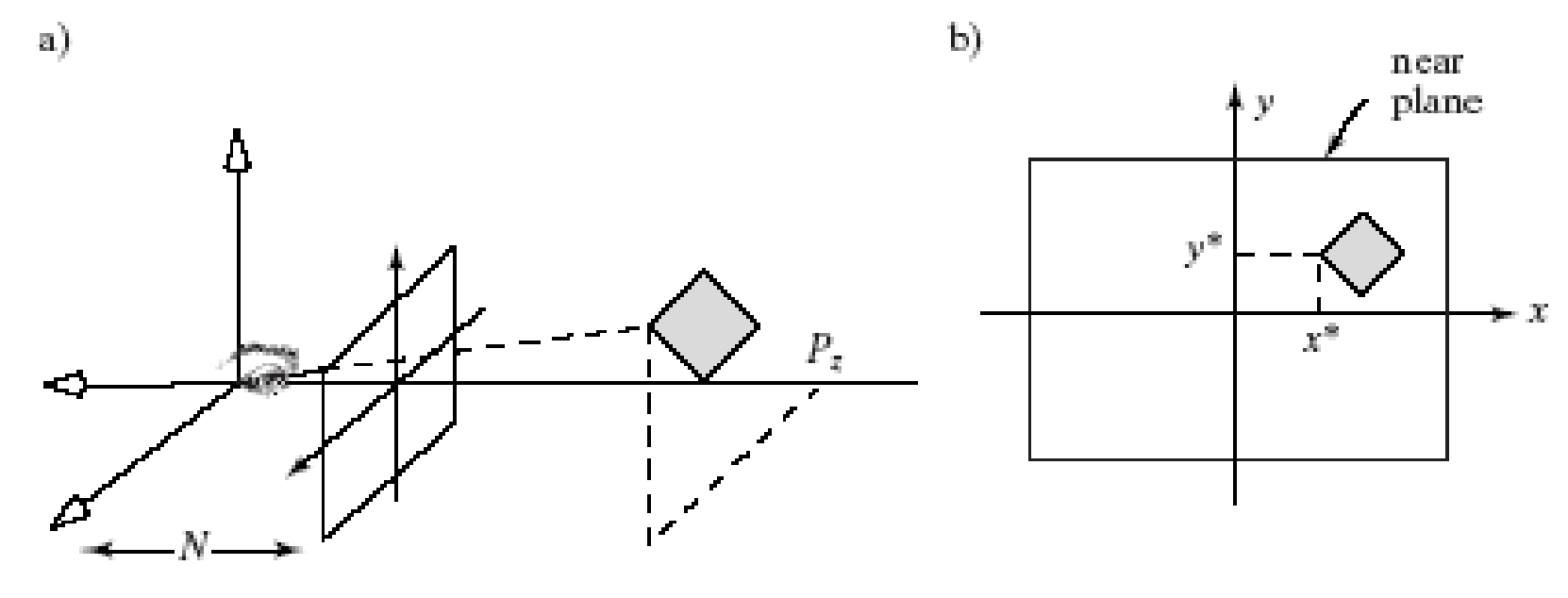
Geometry transformation pipeline

2. We need to assembly lots of 3D objects in a global scene
3. Specify camera/eye positions

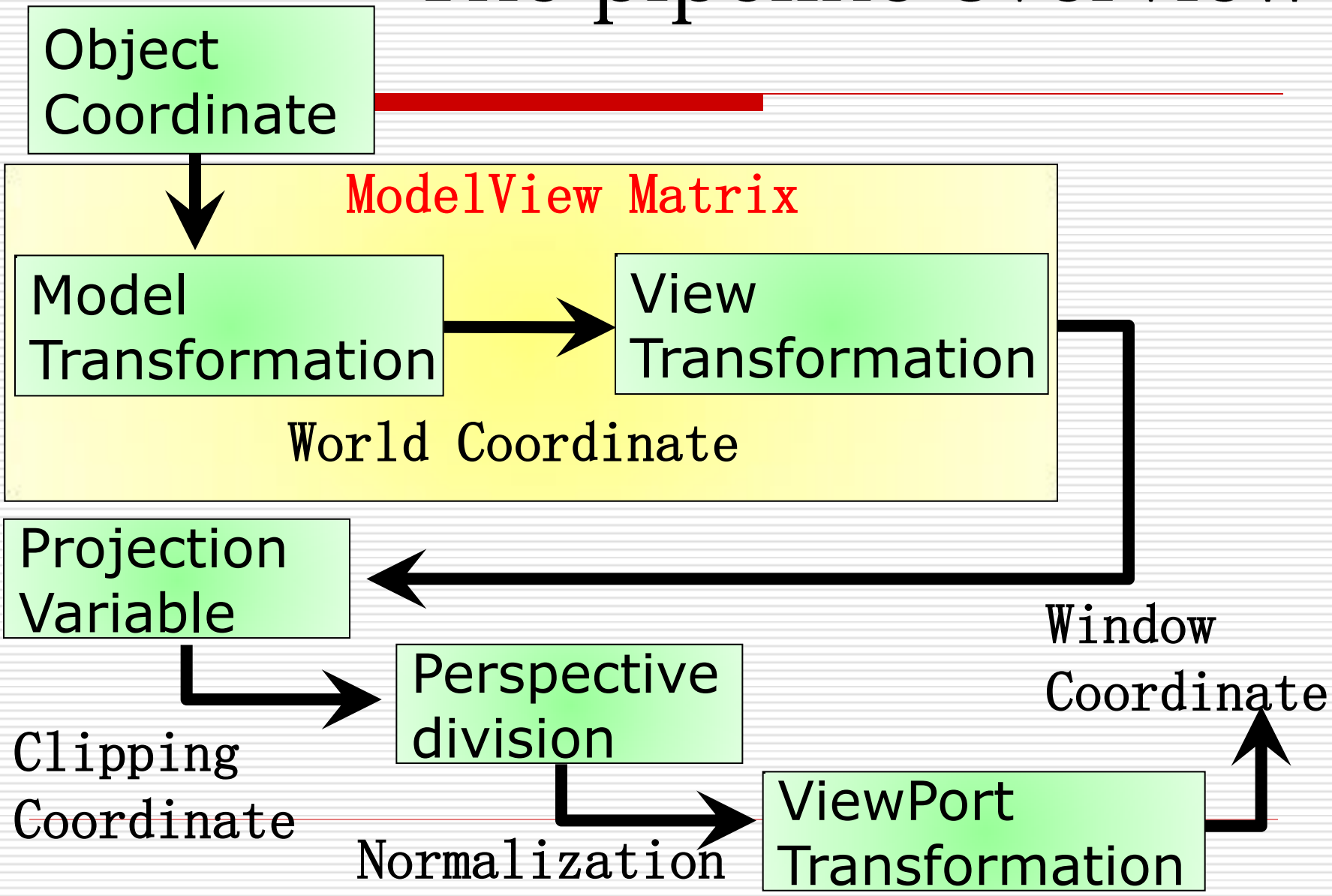


Geometry transformation pipeline

3. Specify camera/eye positions
4. Specify a projection plane
5. Mapping the projection plane to screen

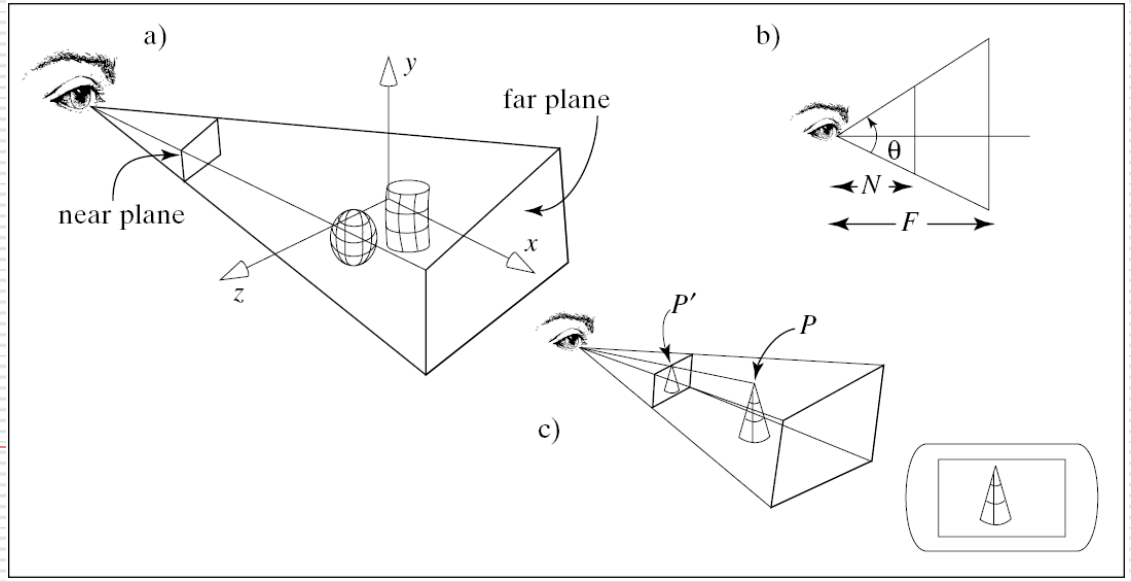


The pipeline overview



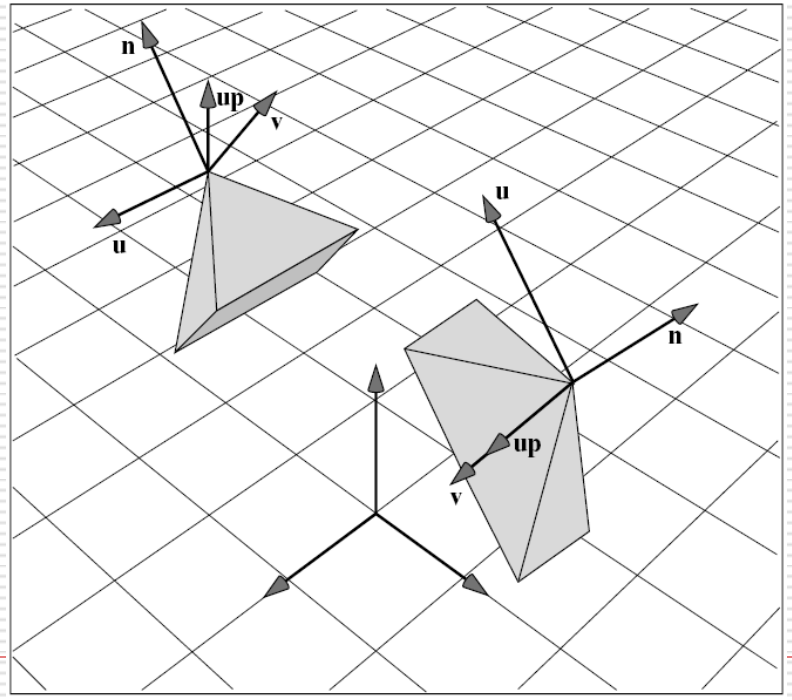
OpenGL transformation

- Scene layout
  - Three transformation matrices
  - glMatrixMode(…)
- Viewport vs. window in OpenGL



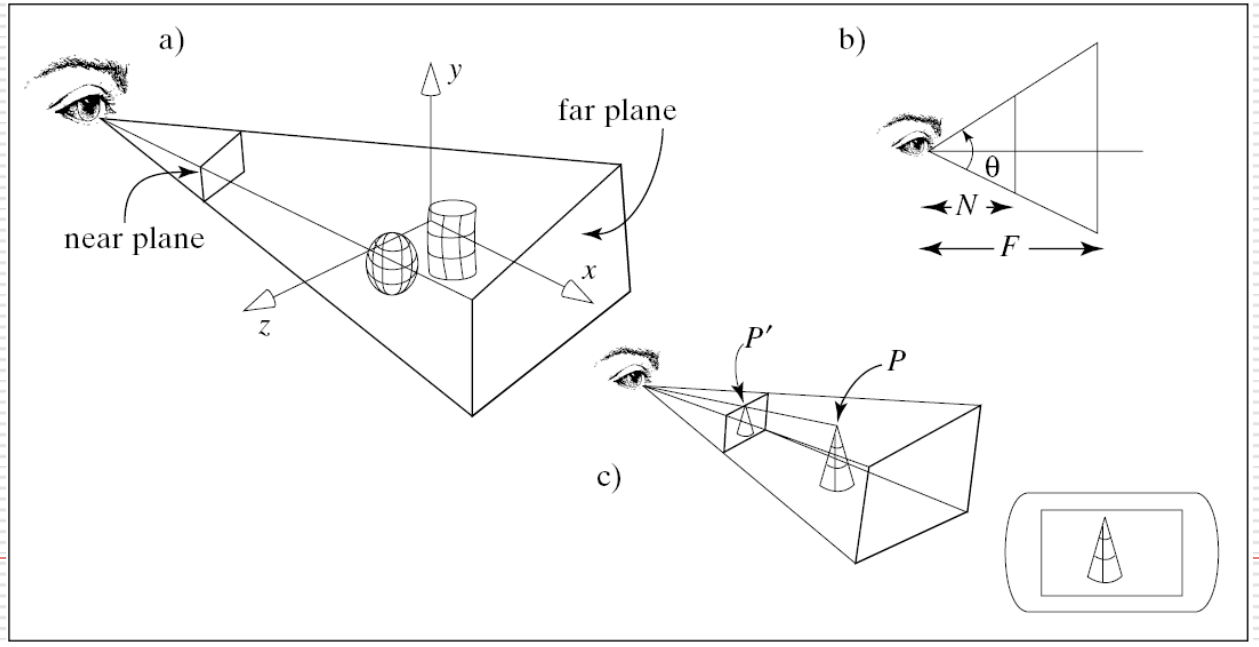
OpenGL transformation

- Three transformation matrices
  - Model transformation



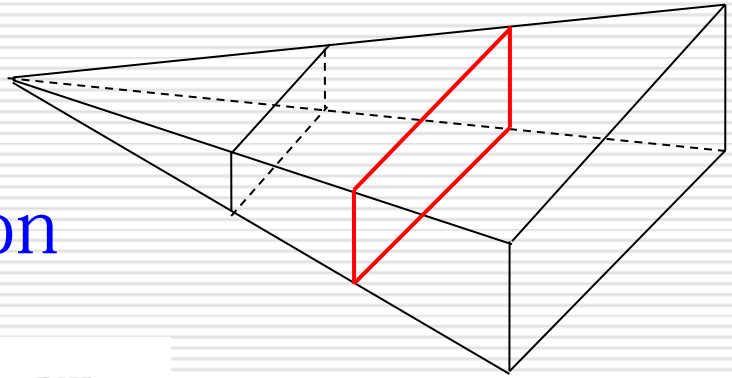
OpenGL transformation

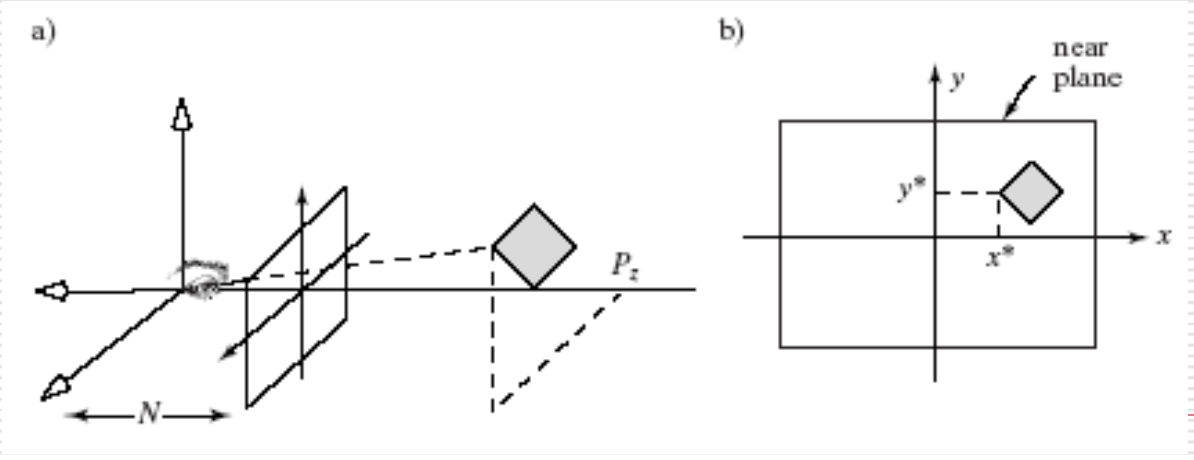
- Three transformation matrices
  - Model transformation
  - View transformation



OpenGL transformation

- Three transformation matrices
  - Model transformation
  - View transformation
  - Projection transformation





OpenGL transformation

glMatrixMode(parameter)

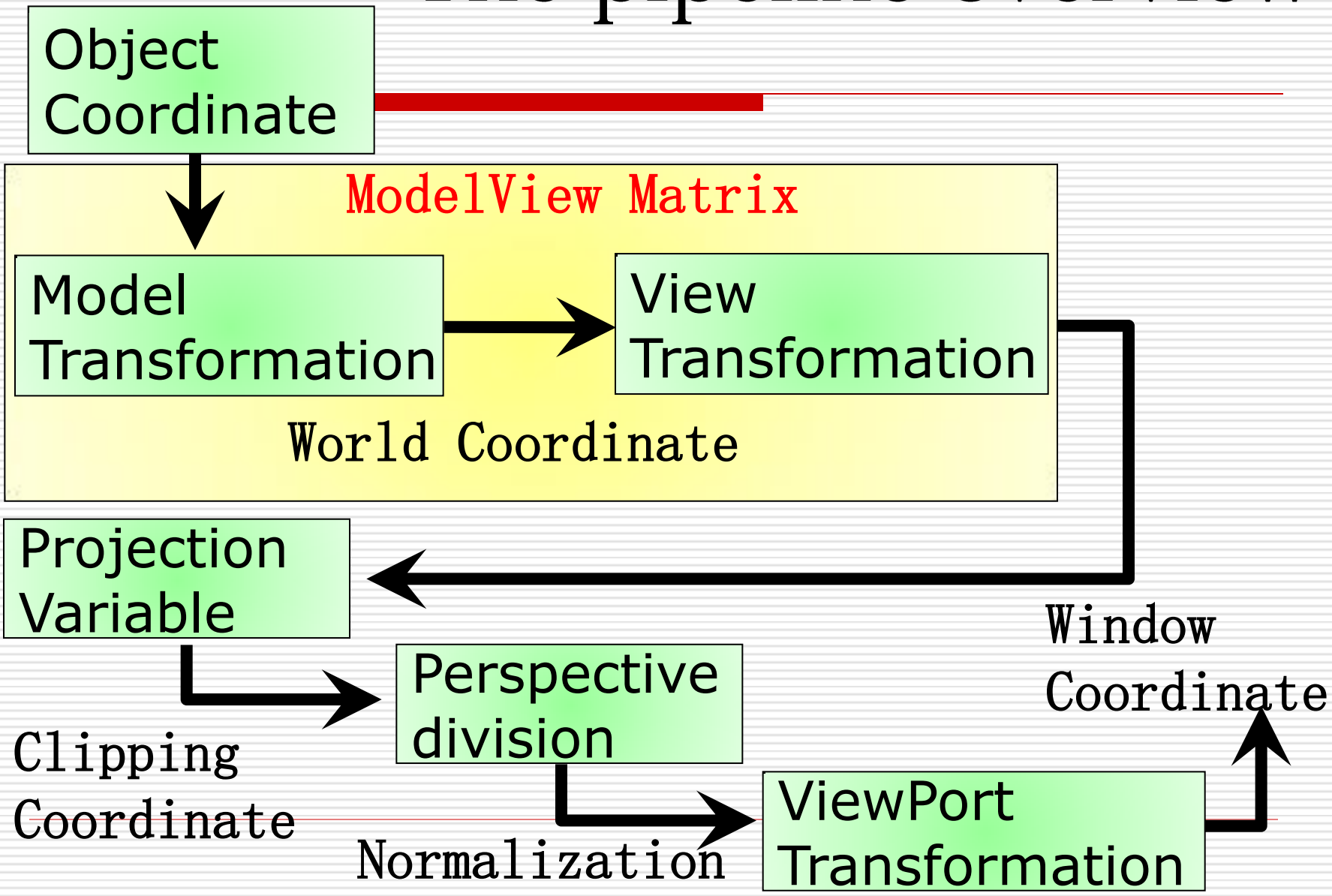
- Model Transformation
- View Transformation

glMatrixMode(GL_MODELVIEW)

- Projection Transformation

glMatrixMode(GL_PROJECTION)

The pipeline overview

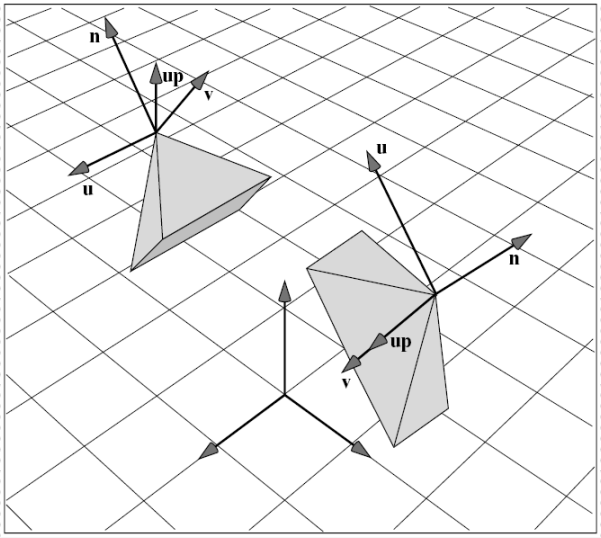


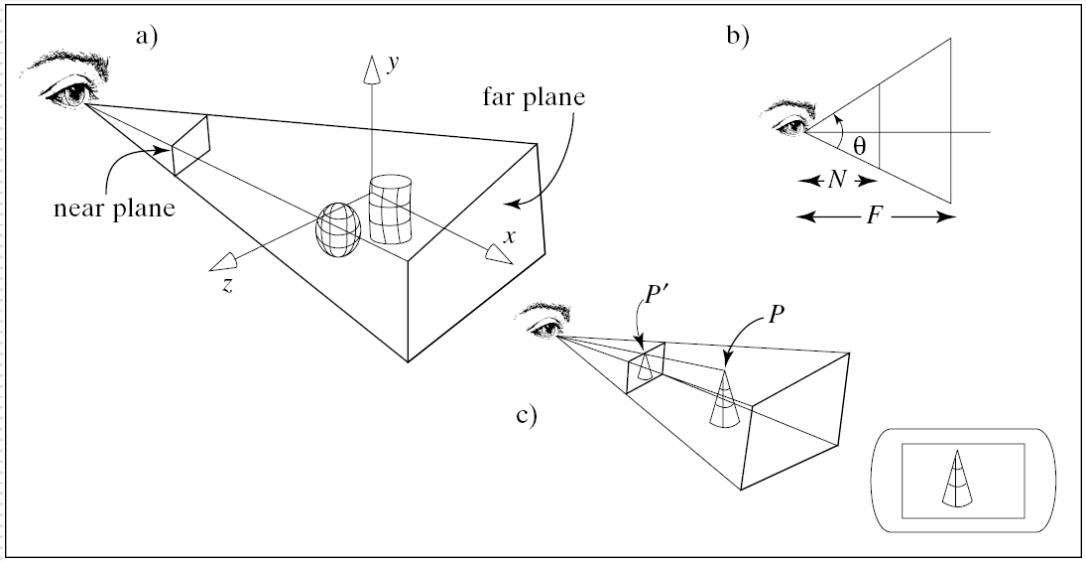
Model/View vs. Projection

- In OpenGL the model-view matrix is used to
  - Position the camera
    - Can be done by rotations and translations but is often easier to use `gluLookAt`
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens

OpenGL transformation

- Three transformation matrices
  - Model transformation
  - View transformation





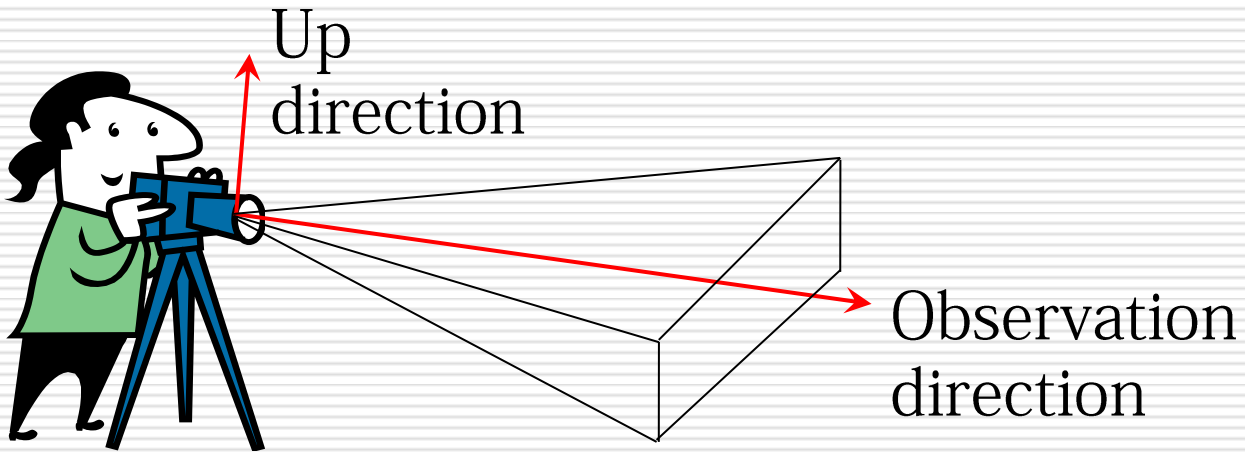
OpenGL transformation

- glMatrixMode(GL_MODELVIEW)
  - Model transformation
  - View transformation

```
glMatrixMode(GL_MODELVIEW);
// make the modelview matrix current
glLoadIdentity();
// start with a unit matrix
gluLookAt(eye.x, eye.y, eye.z, look.x,
look.y, look.z, up.x, up.y, up.z);
```

OpenGL transformation

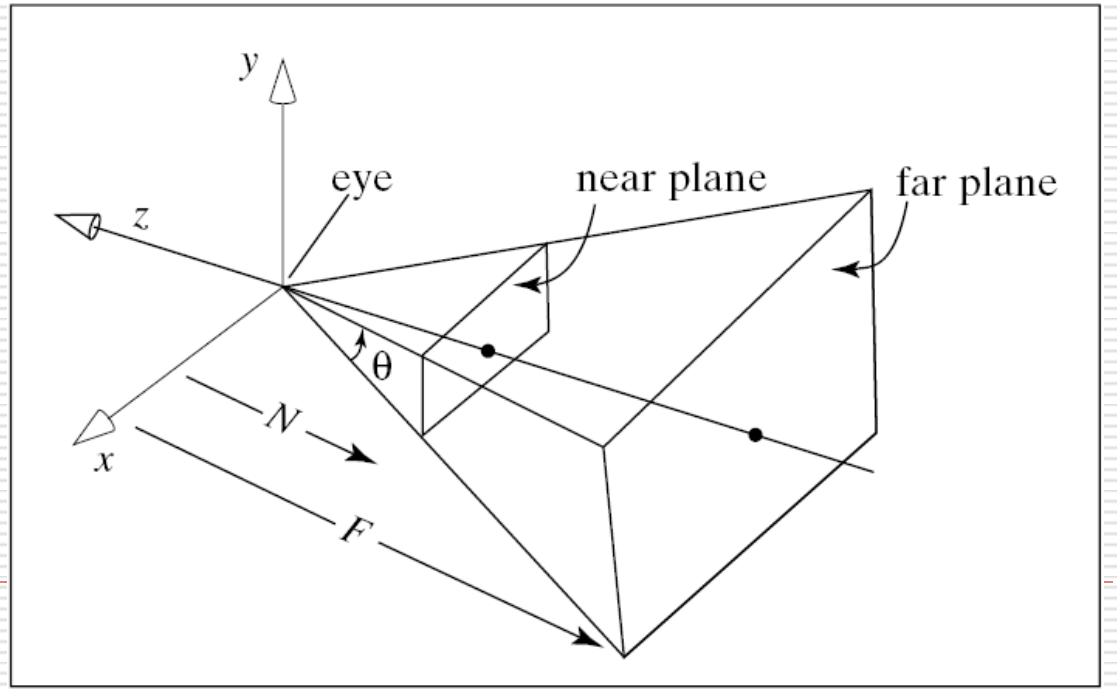
- Set view volume
  - Set eye/camera positions
  - Set the volume




```
gluLookAt(eye.x, eye.y, eye.z, look.x, look.y,
look.z, up.x, up.y, up.z);
```

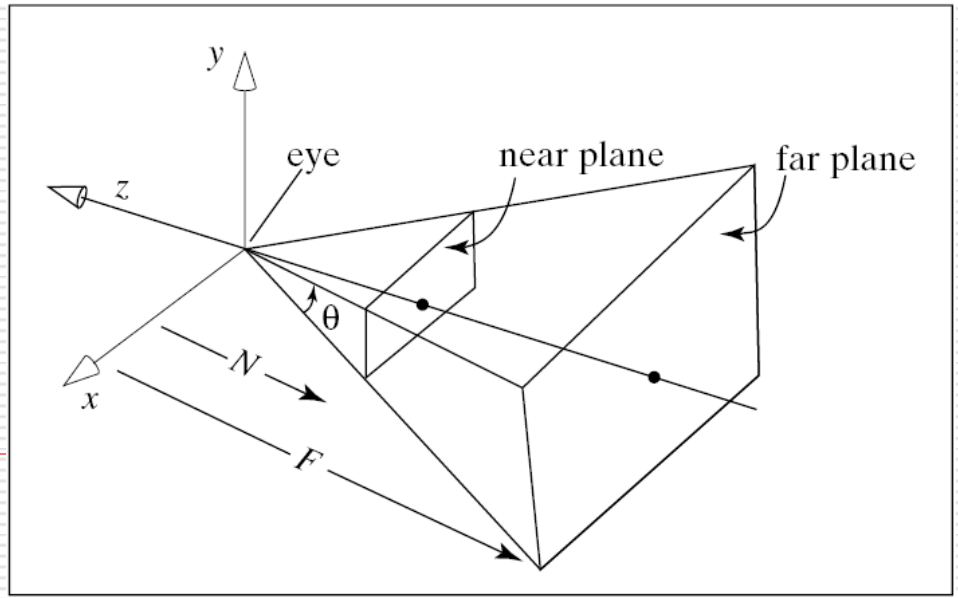
OpenGL transformation

- Set view volume
  - Set eye positions
  - Set the viewing volume



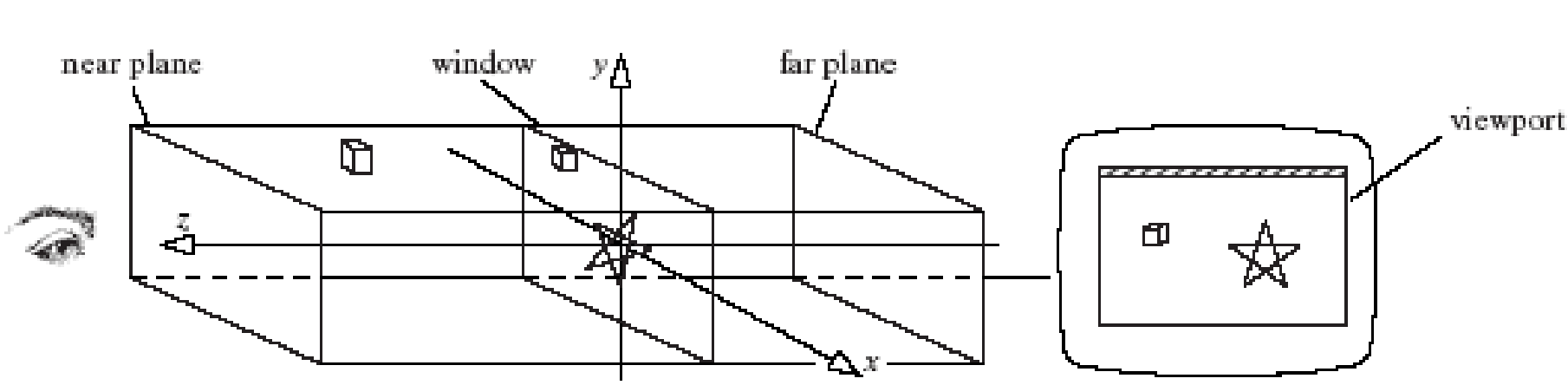
OpenGL transformation

- Set view volume
  - Set eye positions
  - Set the viewing volume
- gluPerspective(viewAngle, aspect, nearDist, farDist);



OpenGL transformation

- gluPerspective(viewAngle, aspect, nearDist, farDist);
- glOrtho (left, right, bottom, top, near, far);
  // sets the view volume parellelpiped.



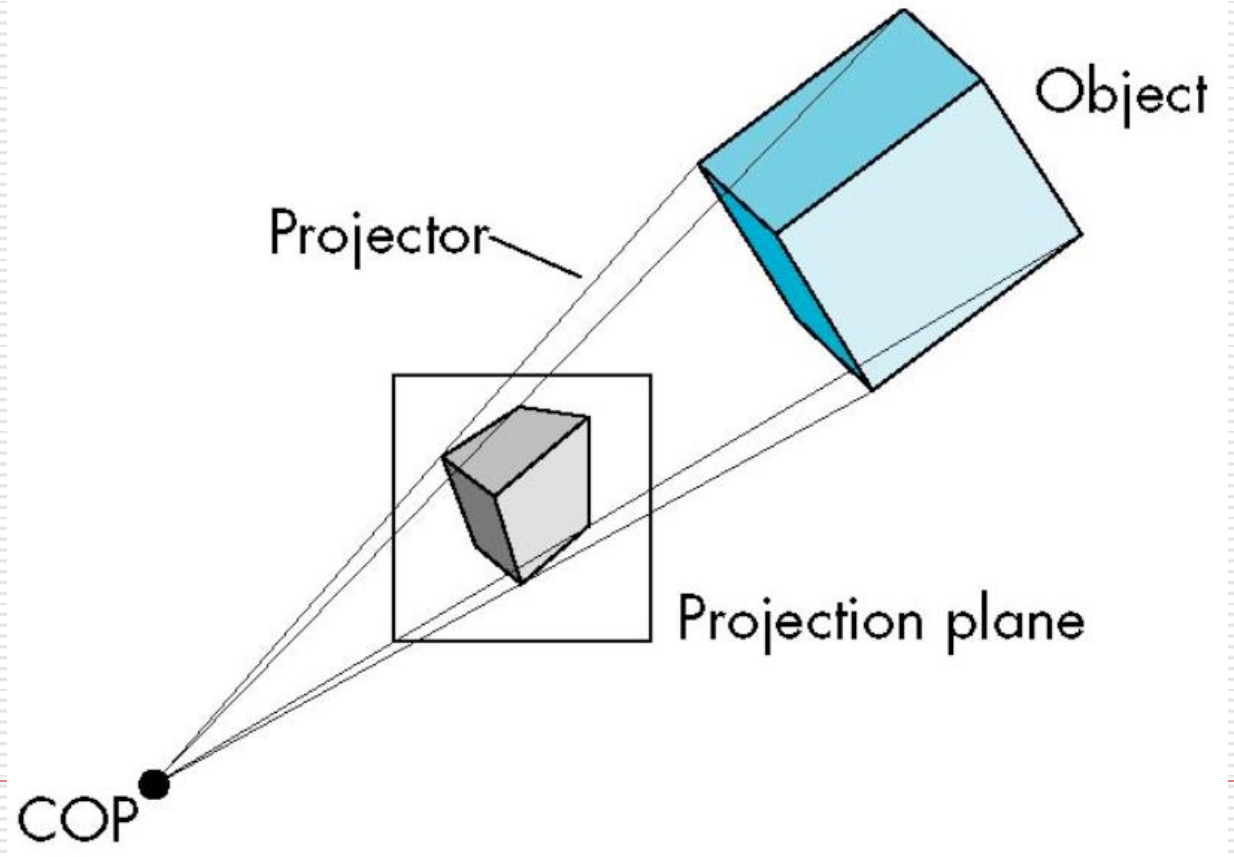
OpenGL transformation

- gluPerspective(viewAngle, aspect, nearDist, farDist);
- glOrtho (left, right, bottom, top, near, far);

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
gluPerspective(viewAngle, aspect, nearDist,
farDist);
```

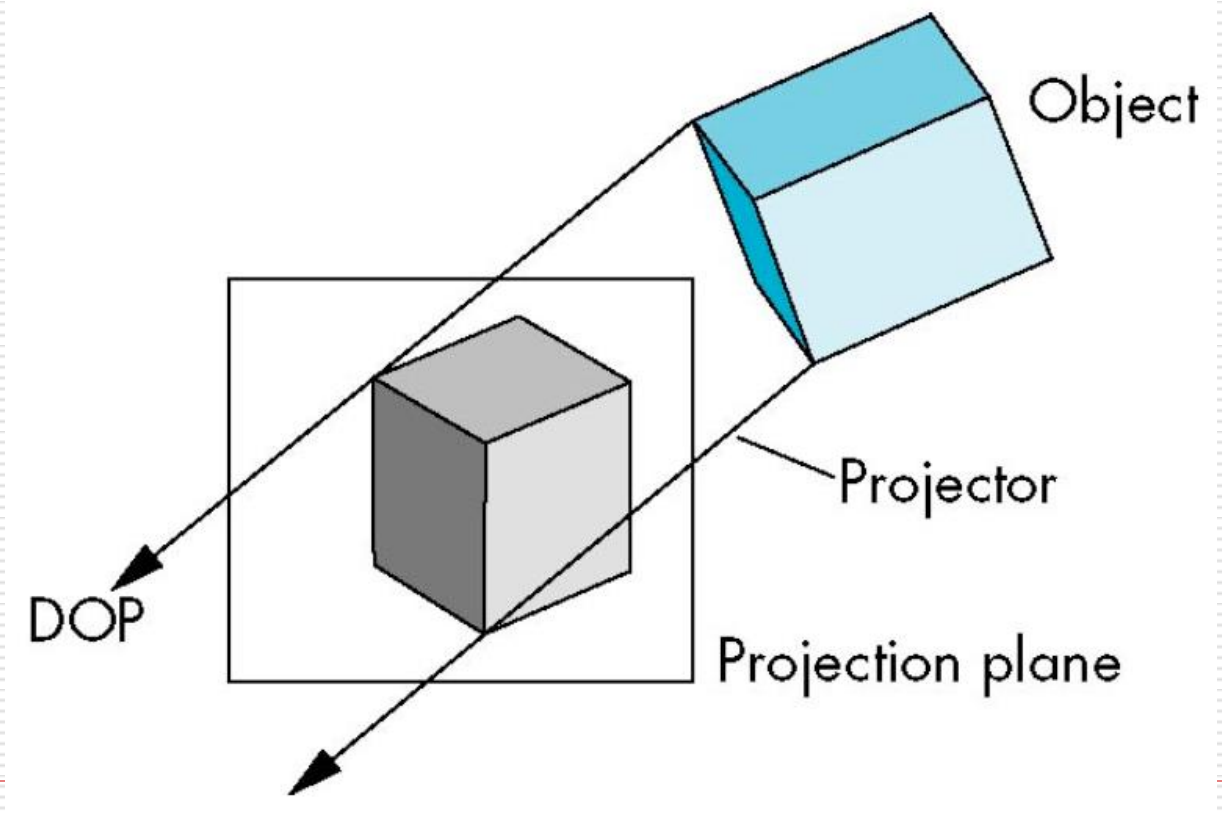
Perspective vs orthogonal projection

- Perspective



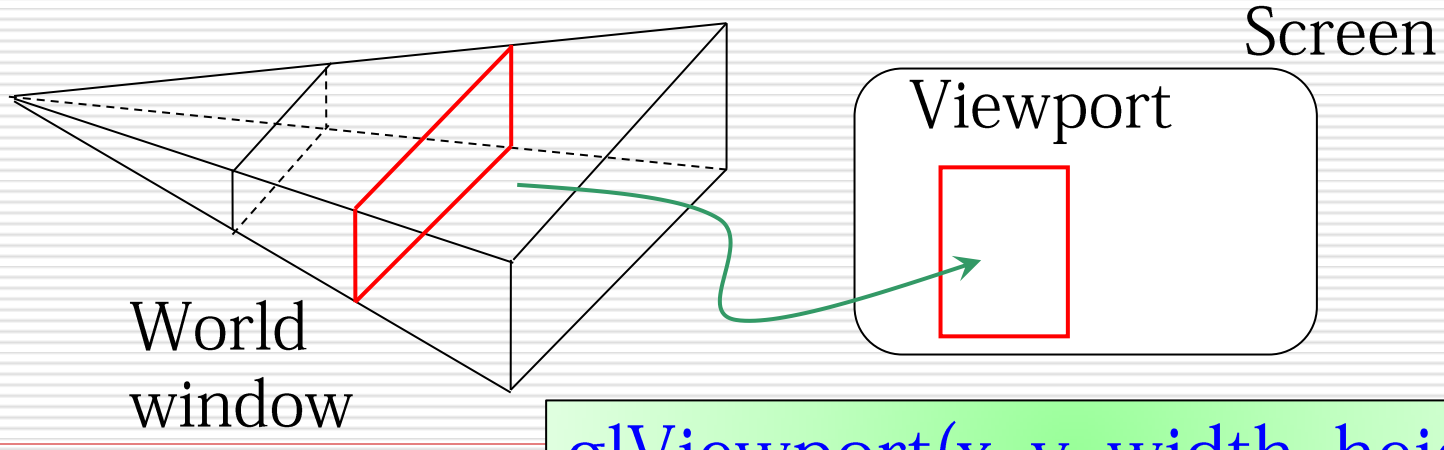
Perspective vs orthogonal projection

- Orthogonal



OpenGL transformation

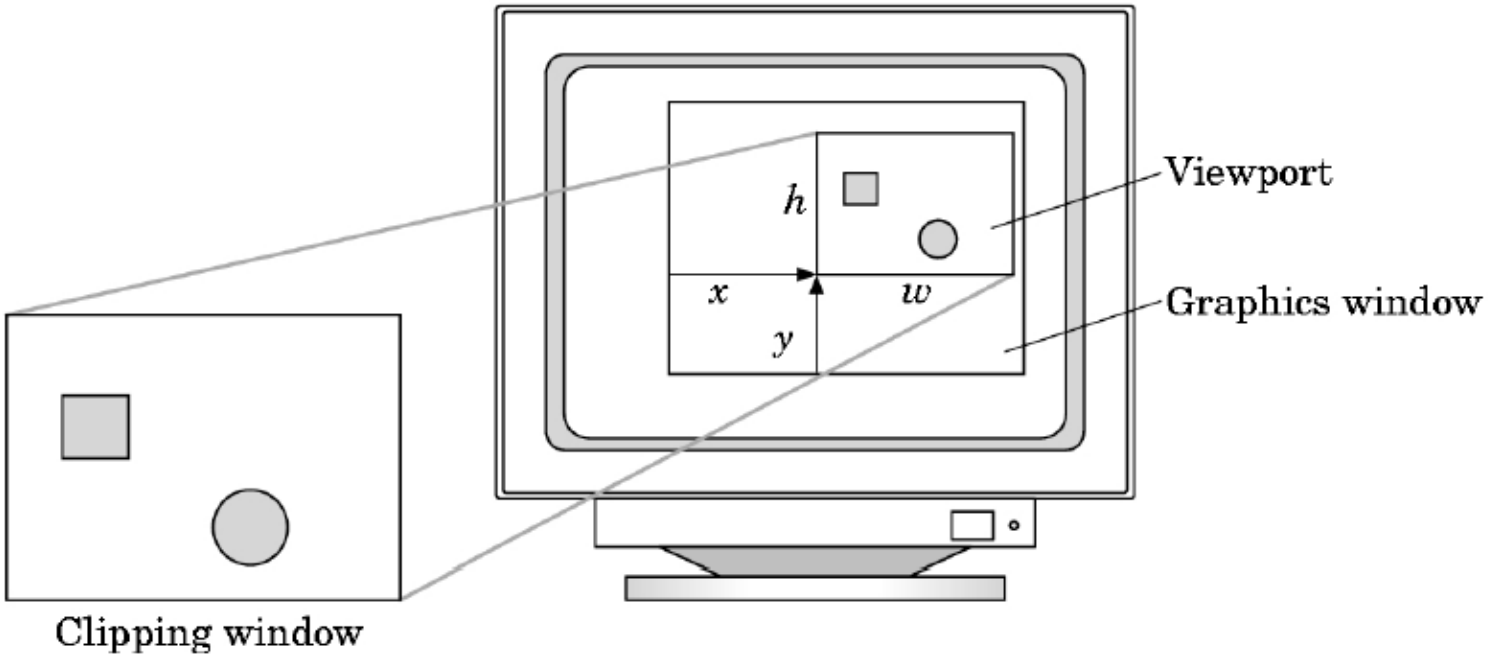
- Three transformation matrices
  - Model/view transformation
  - Projection transformation
- Viewport vs. window in OpenGL



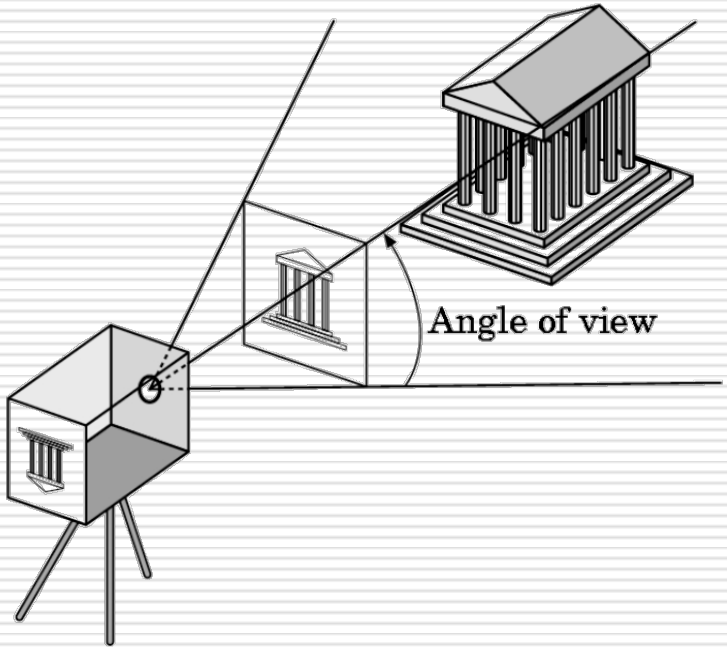
glViewport(x, y, width, height)

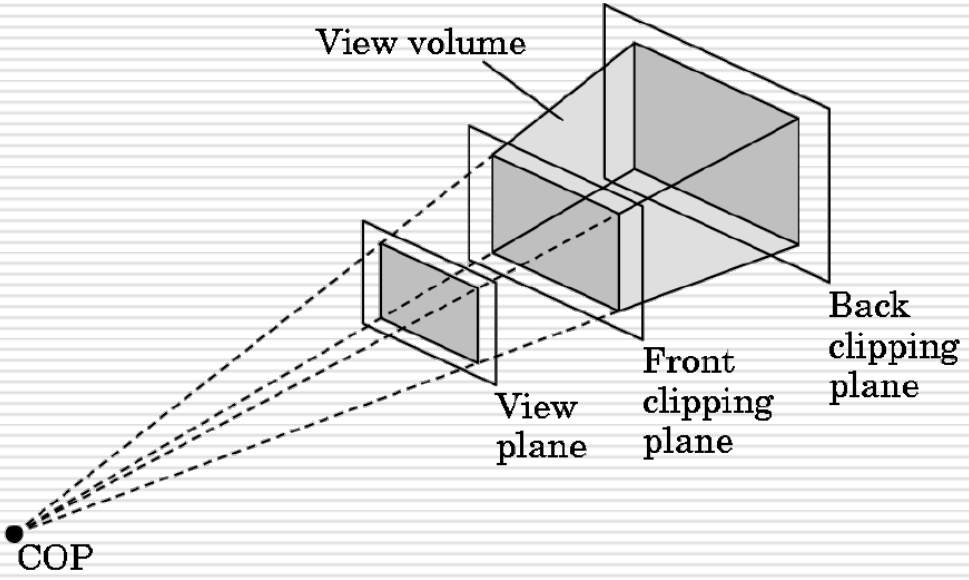
Viewports

- Do not have use the entire window for the image: **`glViewport(x,y,w,h)`**
- Values in pixels (screen coordinates)



OpenGL transformation

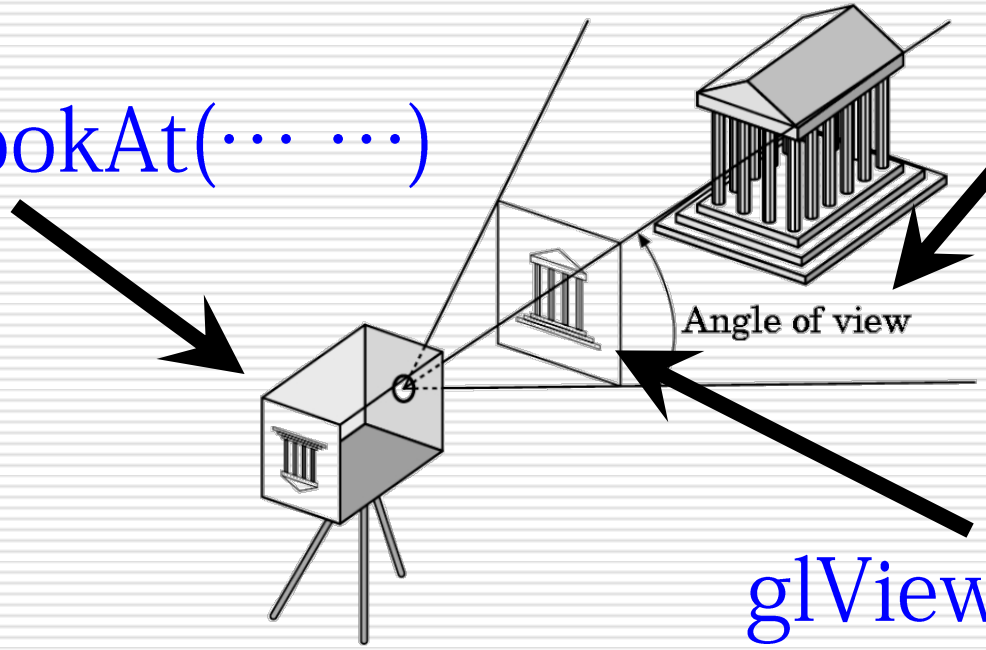




OpenGL transformation

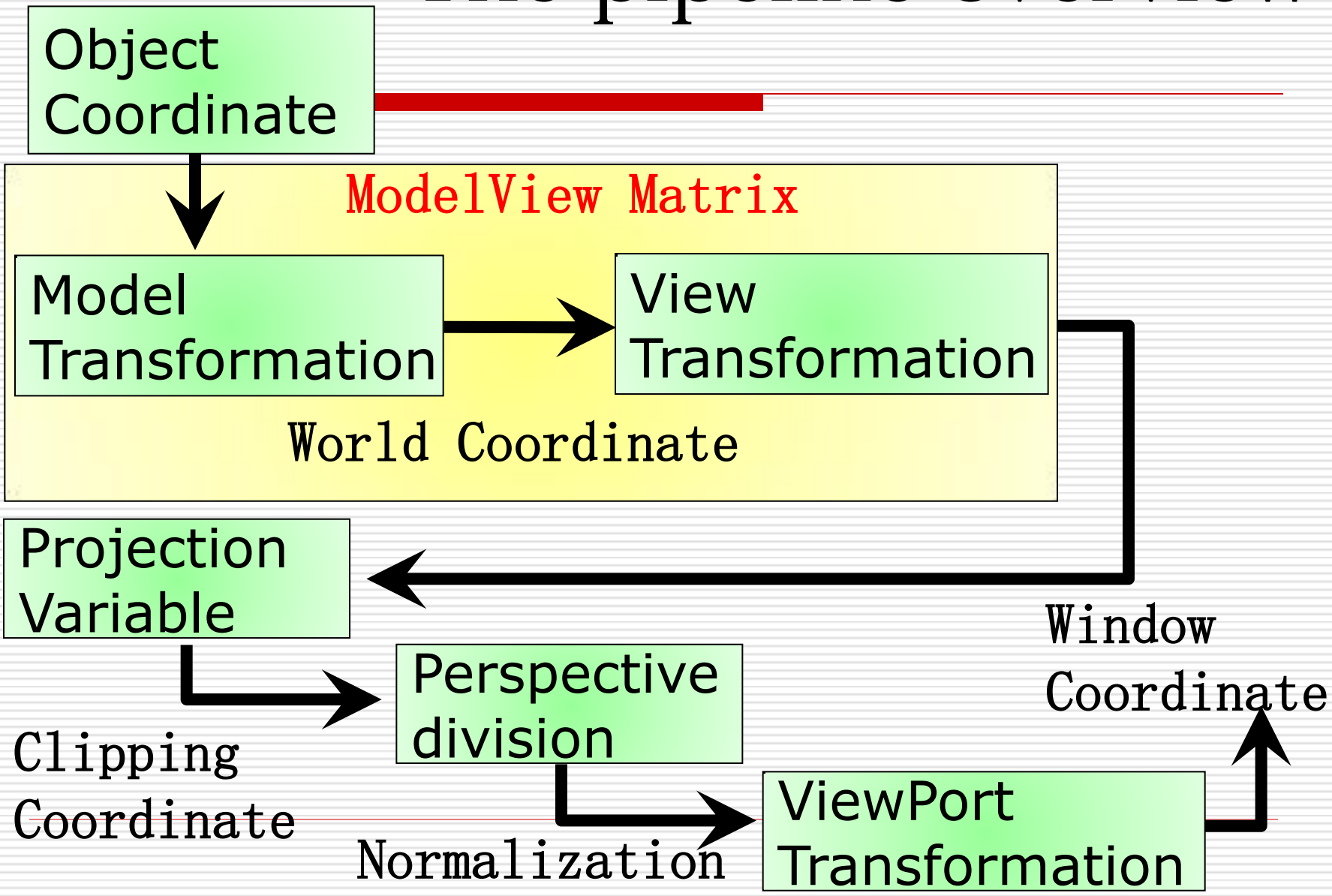
gluPerspective(… …)

gluLookAt(… …)



glViewport(… …)

The pipeline overview



Each 3D object in the scene

- How to carry transformation for each object
  - Rotation: glRotatef(theta, vx, vy, vz)
  - Translation: glTranslatef(dx, dy, dz)
  - Scaling: glScalef(sx, sy, sz)

OpenGL matrices

- In OpenGL matrices are part of the state
- Multiple types
  - Model-View (`GL_MODELVIEW`)
  - Projection (`GL_PROJECTION`)
  - Texture (`GL_TEXTURE`) (ignore for now)
  - Color(`GL_COLOR`) (ignore for now)
- Single set of functions for manipulation
- Select which to manipulated by
  - `glMatrixMode(GL_MODELVIEW);`
  - `glMatrixMode(GL_PROJECTION);`

Matrix representation in OpenGL

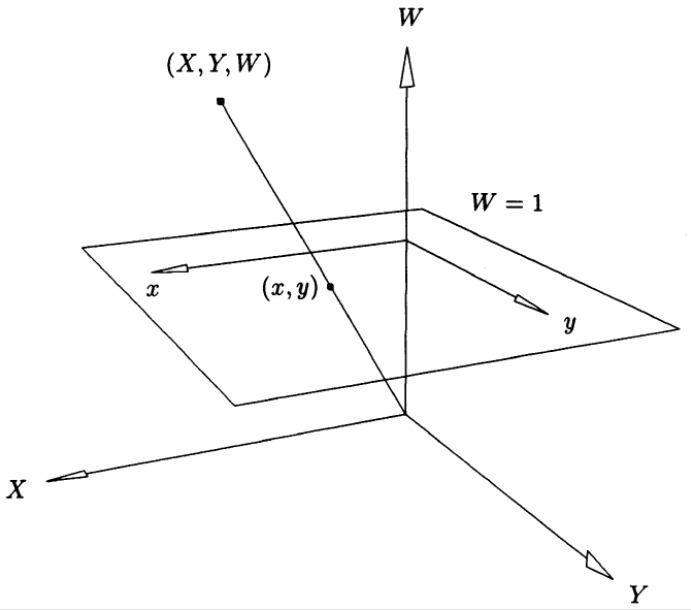
- Vector $l = (x, y, z)$
- Coord-Sys (o, i, j, k)

$$l = o + xi + yj + zk$$

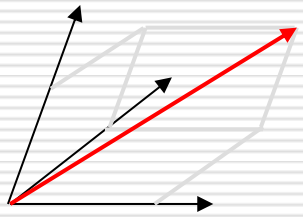
- Homogeneous coordinates

$$(x, y) \Leftrightarrow (wx, wy, w),\ w \neq 0$$

$$(x, y, z) \Leftrightarrow (wx, wy, wz, w),\ w \neq 0$$



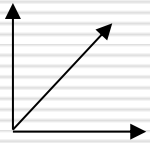
Matrix representation in OpenGL



Directional vector or a point

- Direction $\mathrm{v} = (v_1, v_2, v_3)$

  $\mathrm{v} = v_1\mathrm{i} + v_2\mathrm{j} + v_3\mathrm{k}$

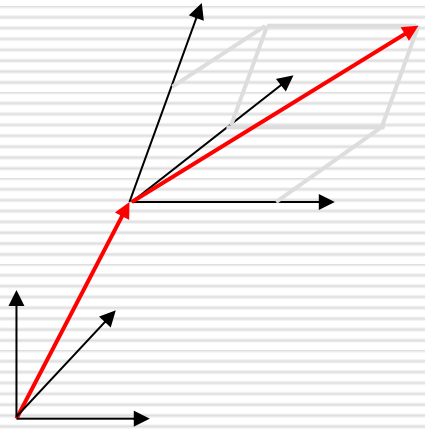


- Homogeneous coordinates

  $(v_1, v_2, v_3, 0)$ $\qquad \tilde{\mathbf{v}} = \begin{pmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix}$

Matrix representation in OpenGL

Directional vector or a point



- A point: $\mathrm{p} = (p_x, p_y, p_z)$

  $\mathrm{p} = \mathrm{o} + p_x\mathrm{i} + p_y\mathrm{j} + p_z\mathrm{k}$

- Homogeneous coordinates

  $(p_x, p_y, p_z, 1)$

$$\tilde{\mathbf{p}} = \begin{pmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix}$$

Rigid transformation (Isometries)

- Translation
- Rotation
  - 3 dof (degree of freedom)
  - $\varepsilon = 1$: orientation preservation
  - $\varepsilon = -1$: orientation reversing

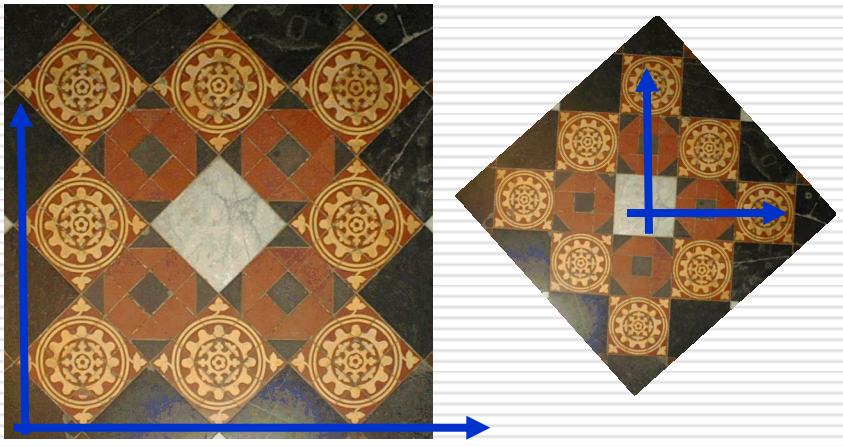
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \varepsilon\cos\theta & -\sin\theta & t_x \\ \varepsilon\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Similarity transformation

- Translation
- Rotation
- Scaling
  - 4 dof (degree of freedom)

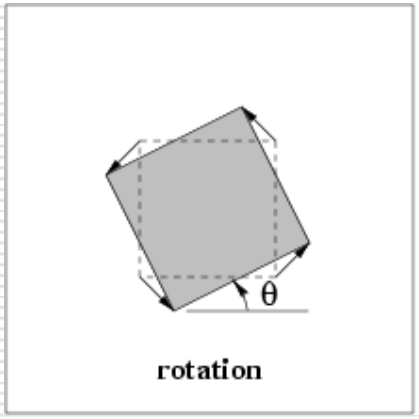
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

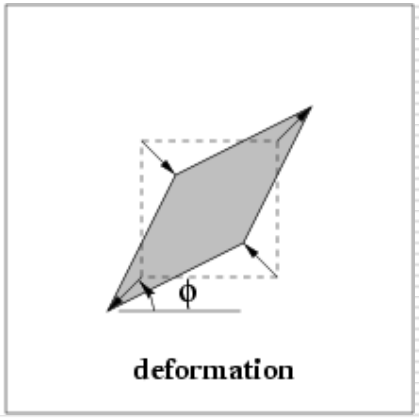
$$\mathbf{x}' = \mathrm{H}_S\mathbf{x} = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}$$



Affine transformation

- Translation
- Rotation
- Scaling
- Shearing
  - 6 dof (degree of freedom)





$$\mathbf{x}' = \mathrm{H}_{\mathrm{A}}\mathbf{x} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \mathbf{x}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

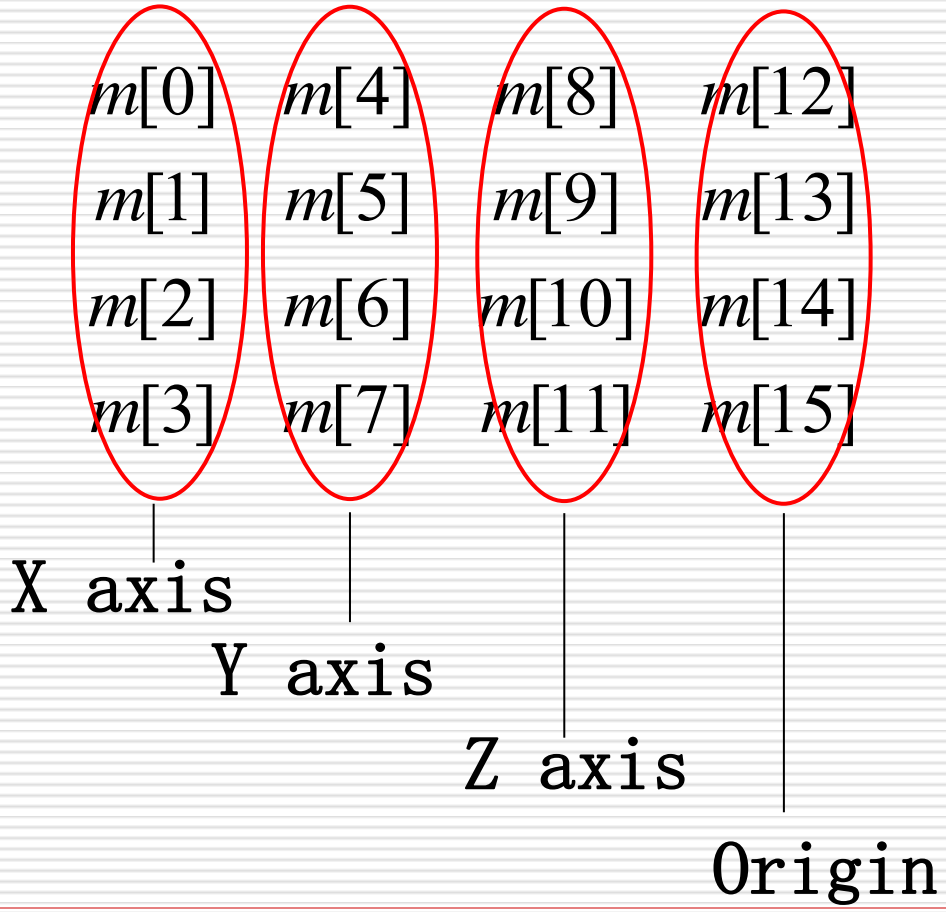
Affine transformation

Matrix in OpenGL

- OpenGL: matrices are all 4x4
- C++：one dimensional (1D) array
- Column first, then row
- OpenGL: rows are also vectors

$$\begin{matrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{matrix}$$

Matrix as a coordinate system



Matrix as a coordinate system

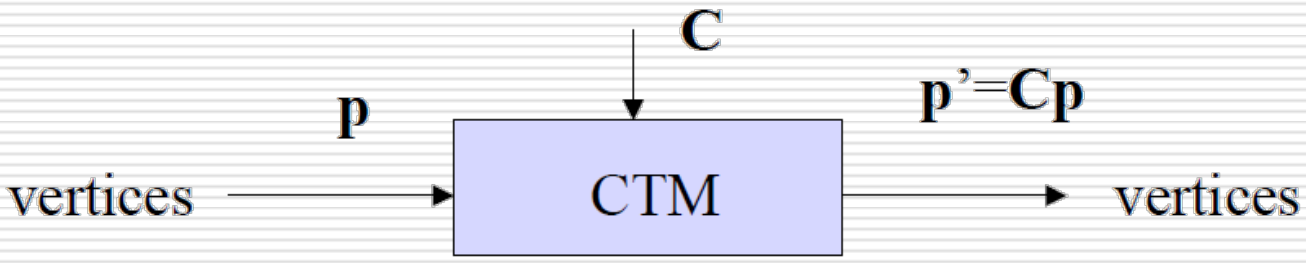
$$\begin{pmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

$$= p_x \begin{pmatrix} m[0] \\ m[1] \\ m[2] \\ m[3] \end{pmatrix} + p_y \begin{pmatrix} m[4] \\ m[5] \\ m[6] \\ m[7] \end{pmatrix} + p_z \begin{pmatrix} m[8] \\ m[9] \\ m[10] \\ m[11] \end{pmatrix} + \begin{pmatrix} m[12] \\ m[13] \\ m[14] \\ m[15] \end{pmatrix}$$

X axis Y axis Z axis Origin

Current matrix transformation (CMT)

- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit



Matrix stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures
  - Avoiding state changes when executing display lists
- OpenGL maintains stacks for each type of matrix
  - Access present type (as set by **`glMatrixMode)`** by

```
glPushMatrix()
glPopMatrix()
```

CMT operations

- The CTM can be altered either by loading a new CTM or by postmutiplication

Load an identity matrix: **C ← I**
Load an arbitrary matrix: **C ← M**

Load a translation matrix: **C ← T**
Load a rotation matrix: **C ← R**
Load a scaling matrix: **C ← S**

Postmultiply by an arbitrary matrix: **C ← CM**
Postmultiply by a translation matrix: **C ← CT**
Postmultiply by a rotation matrix: **C ← C R**
Postmultiply by a scaling matrix: **C ← C S**

Rotation about a fixed point

Start with identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
Move fixed point to origin: $\mathbf{C} \leftarrow \mathbf{CT}$
Rotate: $\mathbf{C} \leftarrow \mathbf{CR}$
Move fixed point back: $\mathbf{C} \leftarrow \mathbf{CT}^{-1}$

Result: $\mathbf{C} = \mathbf{TR}\,\mathbf{T}^{-1}$ which is **backwards**.

This result is a consequence of doing postmultiplications.

Reversing the order

We want $\mathbf{C} = \mathbf{T}^{-1}\,\mathbf{R}\,\mathbf{T}$
so we must do the operations in the following order

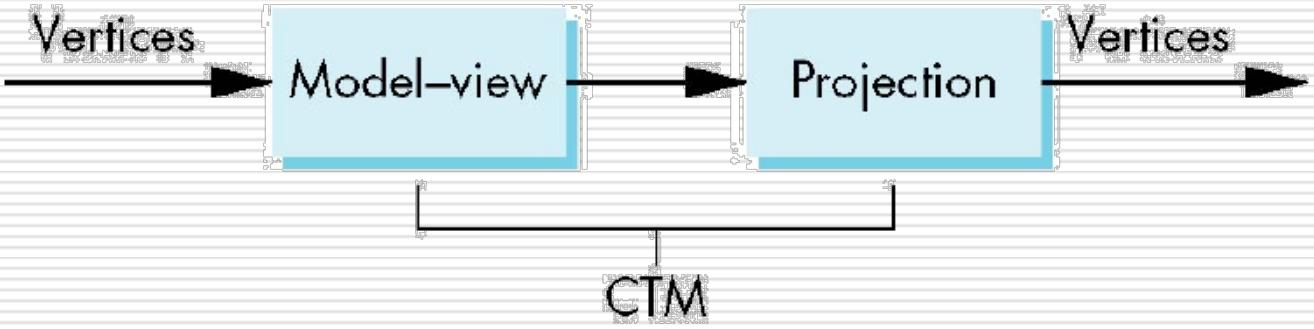
$\mathbf{C} \leftarrow \mathbf{I}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}^{-1}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{R}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}$

Each operation corresponds to one function call in the program.

Note that the last operation specified is the first executed in the program

CTM in OpenGL

- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
- Can manipulate each by first setting the correct matrix mode



Rotation, translation, scaling

Load an identity matrix:

```
glLoadIdentity()
```

Multiply on right:

```
glRotatef(theta, vx, vy, vz)
```

`theta` in degrees, (`vx, vy, vz`) define axis of rotation

```
glTranslatef(dx, dy, dz)
```

```
glScalef( sx, sy, sz)
```

Each has a float (f) and double (d) format (`glScaled`)

Example

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(-1.0, -2.0, -3.0);
```

- Remember that last matrix specified in the program is the first applied

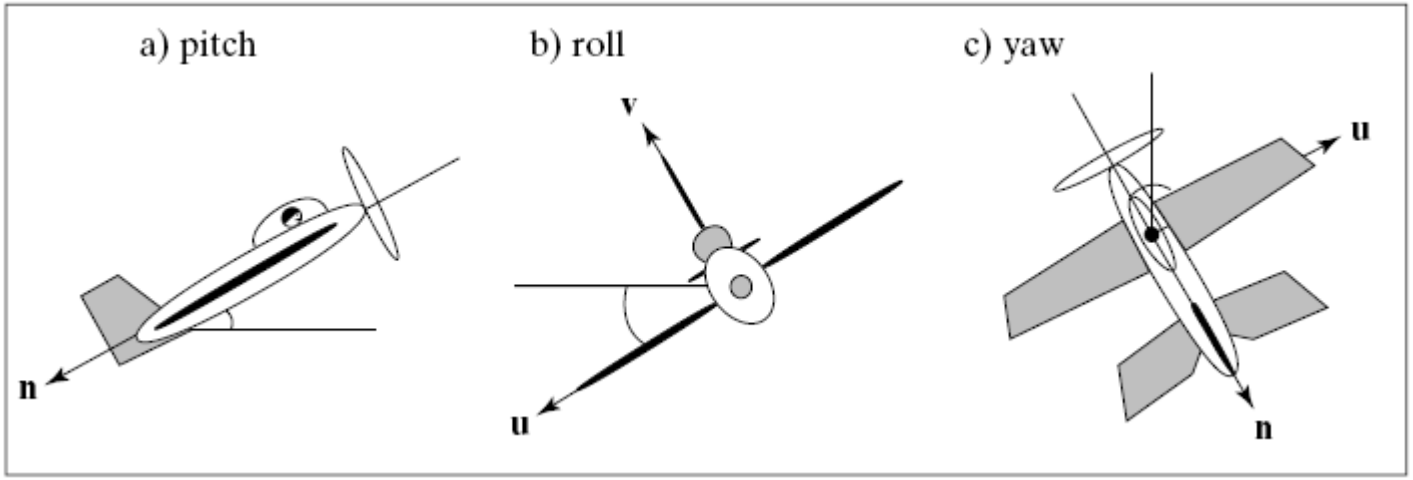
Rotation :

matrix, quaternion and Euler angle vector

- The most common coordinate transformation in 3D engine is rotation.
  - There several ways to achieve rotation: matrix, quaternion and Euler angle vector (angle or radian). The most accurate and least restrictive way is to store them in a matrix. Matrix is a mathematical concept, it is a form of organized rows and columns of rectangular blocks. When these numbers perform calculations with another matrix or numbers of point common in engine, you can change the corresponding value.

Convert between Matrix and Euler angle

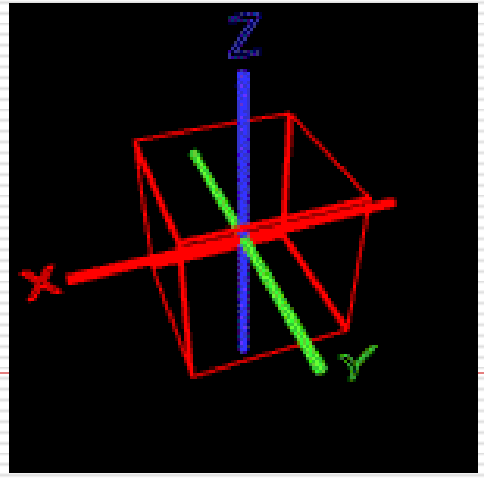
- It is easy to convert from Euler angle vector, what is hard is to convert back.
- If using Euler angle here, we need to provide coordinate in the following sequence：
  - Yaw：y axis of Euler angle vector
  - Pitch：x axis of Euler angle vector
  - Roll： z axis of Euler angle vector
- It should be noted that all rotation information use radian. To get angle, do not forget to convert



A typical example of an airliner cabin, after the definition of cabin direction as x-axis, left wing stretching direction as y-axis, z-axis was fixed by the right hand rule. pitch refers to the (aircraft's) pitch, that is, the rotation around the y-axis movement. roll (roll), rotation around the x-axis movements. Yaw is yaw, i.e., rotation around the z-axis action.

Problem of gimbal lock of Euler angle

- Gimbal lock is refer to that the two rotation axes of object point to the same direction. In fact, when the two rotation axes are parallel, we say gimbal lock phenomenon occurs, in other words, around an axis may cover the other axis of rotation, thus losing one dimension degree of freedom



Problem of gimbal lock of Euler angle

- Generally speaking, gimbal lock can occur when rotate using Euler angle, the reason is that Euler angles rotate around axis independently according to a certain sequence. Let us imagine a specific rotation of the scene, the object rotates around x axis first, then around y axis, and finally z axis is chosen to complete a rotation. Gimbal lock occurs when you rotate 90 angle around y axis, because x axis has been evaluated, and it no longer rotate with the other two axes, thus x axis and z axis point to the same direction (which is equivalent to the same axis).

Quaternion

- (s, vx, vy, vz)
  - s is a scale
  - v is a vector

$$[s_1, u_1]\ [s_2, u_2] = [(s_1 s_2 - u_1 \cdot u_2), (s_1 u_2 + s_2 u_1 + u_1 \times u_2)]$$

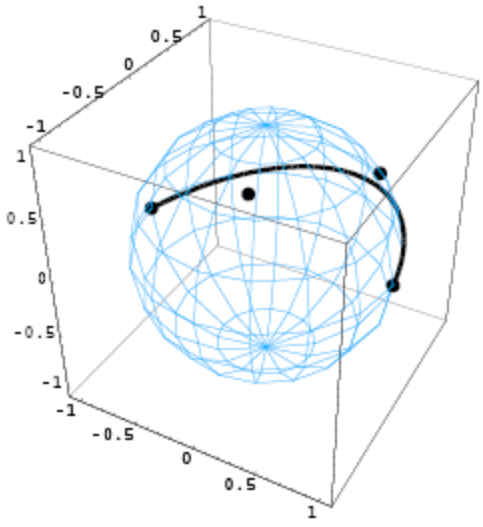
Visualizing quaternions: course notes for Siggraph 2007.
http://dl.acm.org/citation.cfm?id=1281634

Hamilton

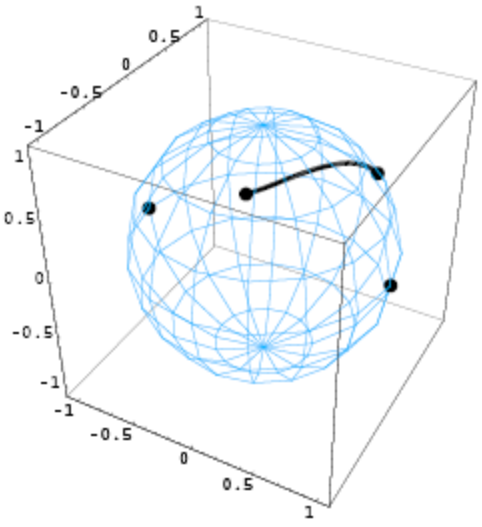
The first to ask *"If you can do 2D geometry with complex numbers, how might you do 3D geometry?"* was William Rowan Hamilton, circa 1840.

Sir William Rowan Hamilton
4 August 1805 — 2 September 1865

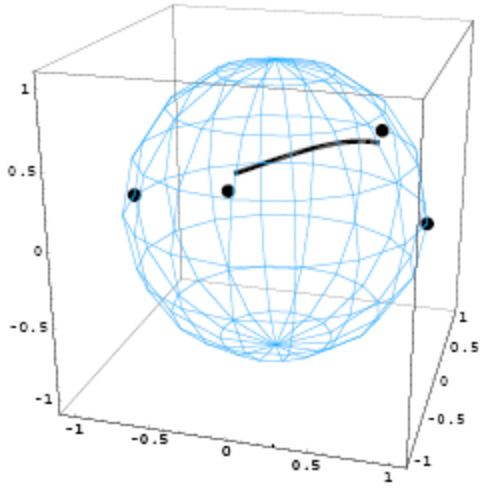
Spherical Interpolations



Bezier

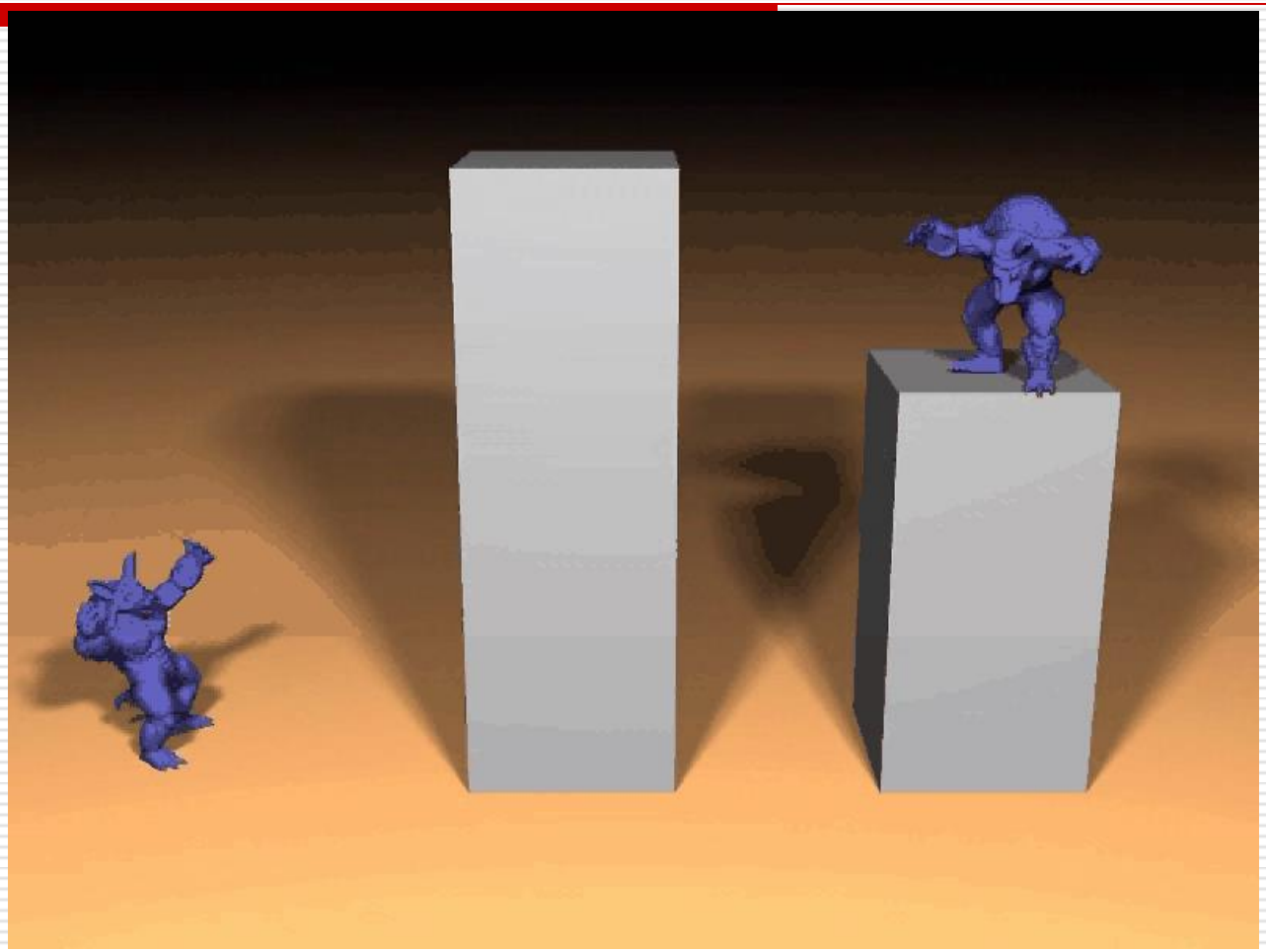


Catmull-Rom



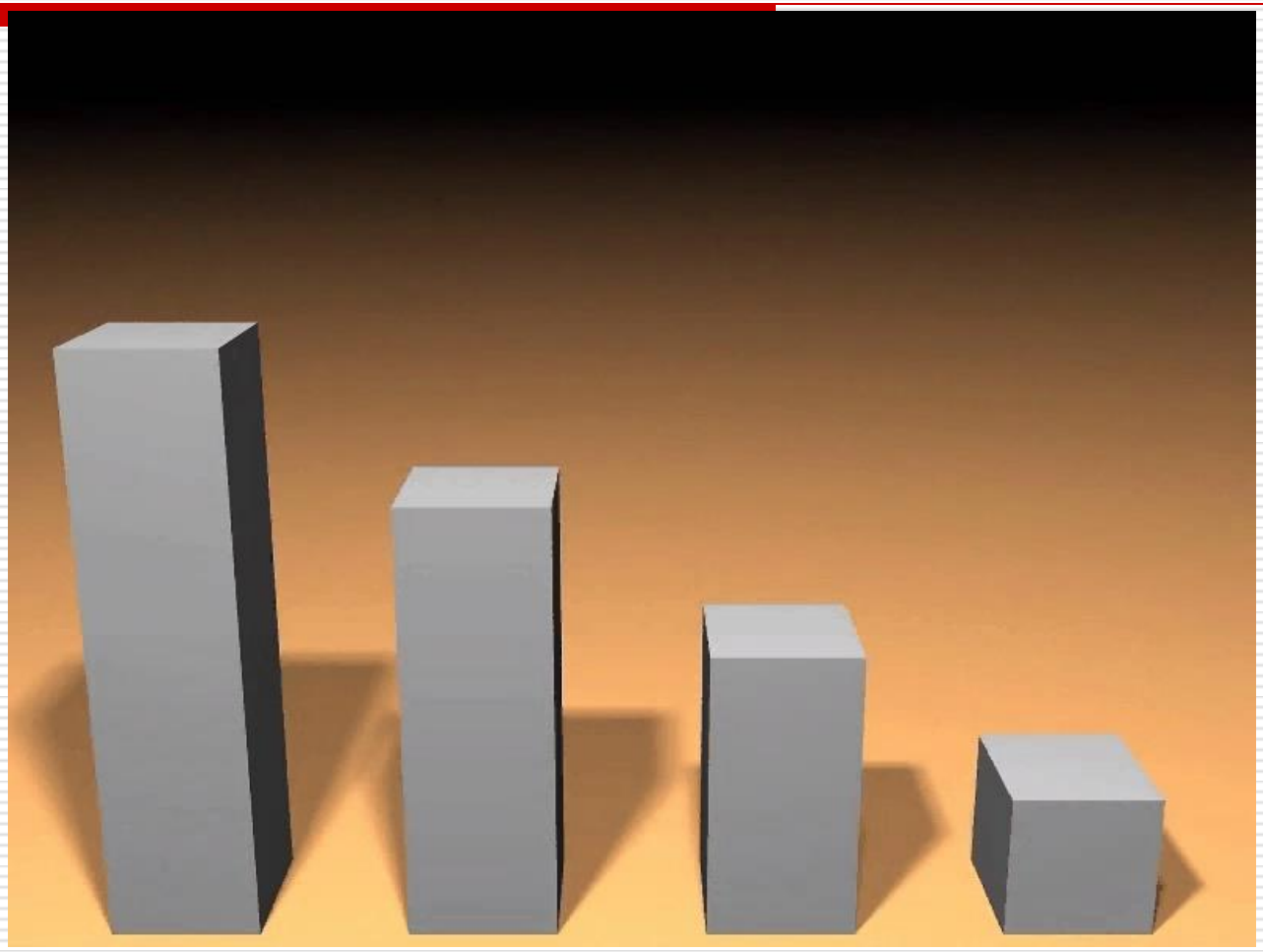
Uniform B

Shape Interpolations



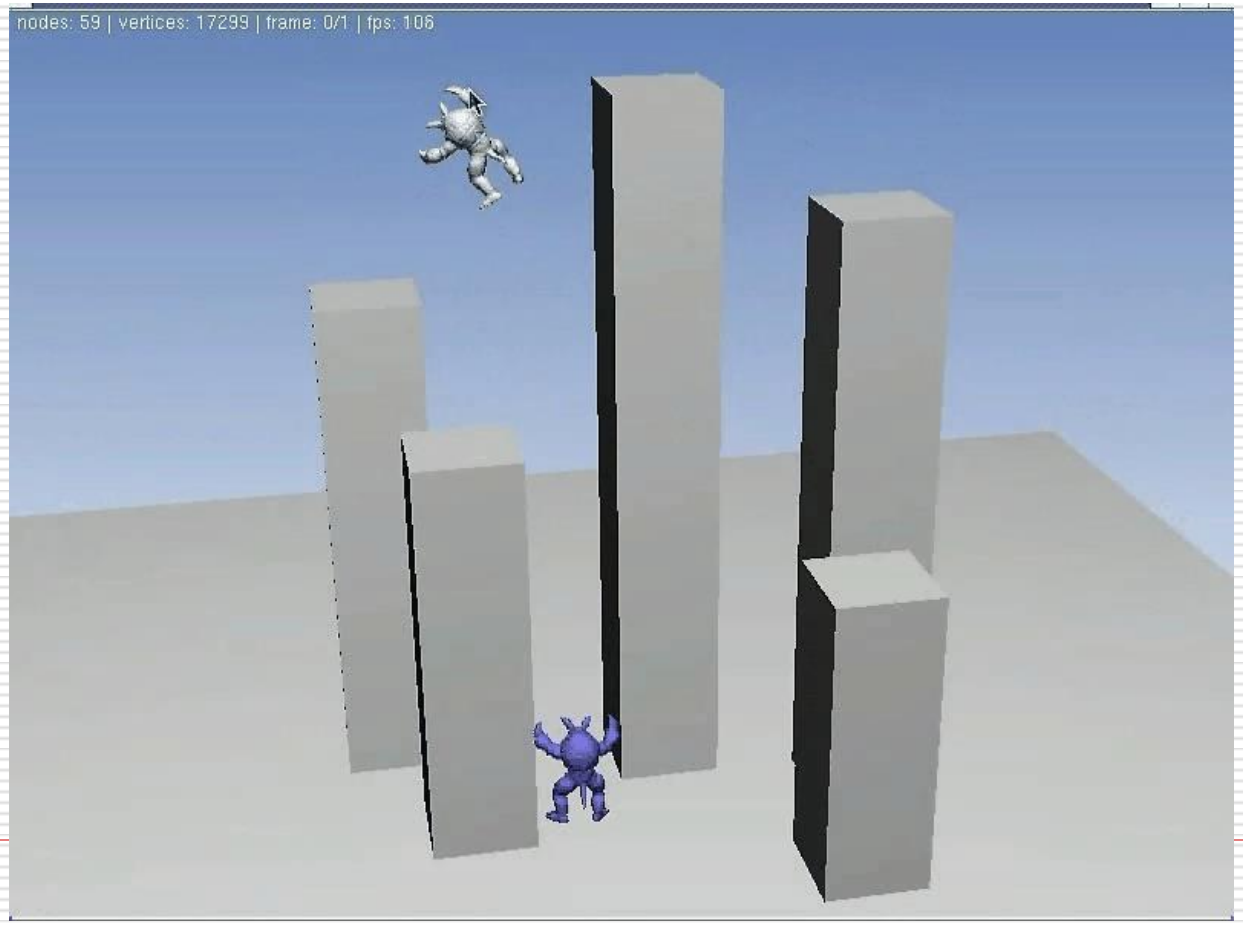
Adaptive temporal sampling

Shape Interpolations



Shape Interpolations: Objective

Find a smooth path for a deformable object from given key frame poses.

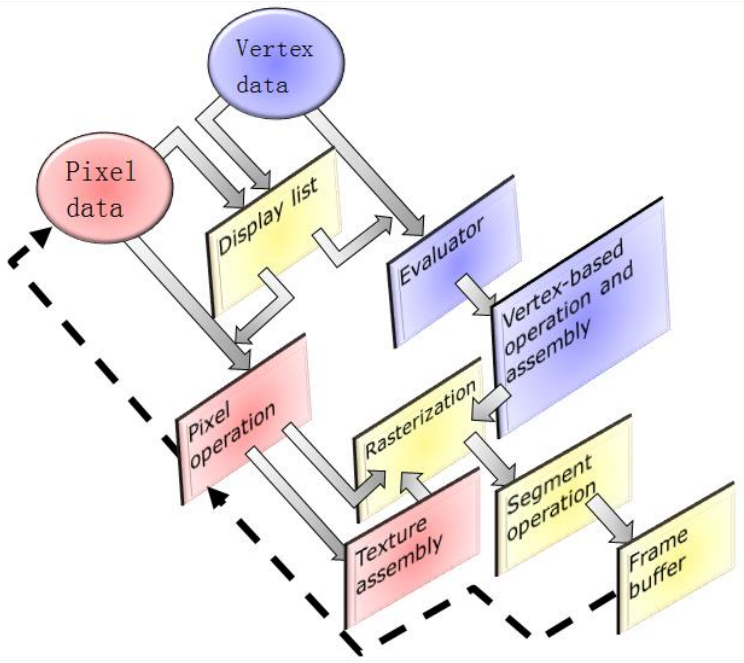


OpenGL rendering pipeline

- Geometry (object transformation)

  Learn how to carry out transformation in OpenGL

- Color (lighting, texture, etc.)



End.
Thanks