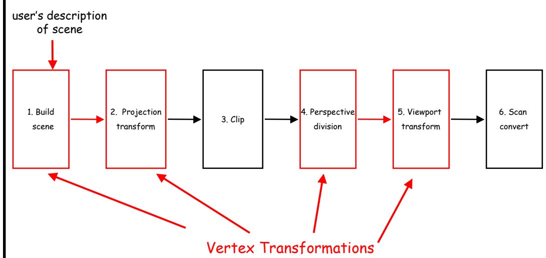
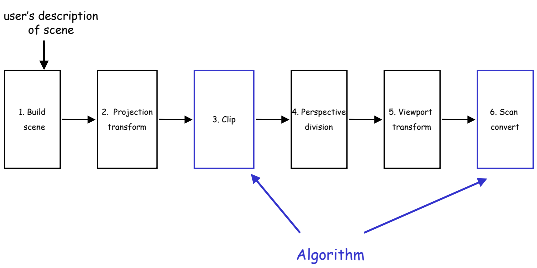
cs155 - z sweedyk

graphics
pipeline
systems

graphics pipeline



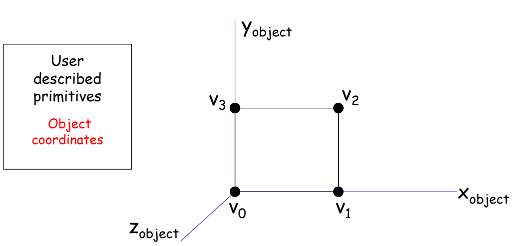
graphics pipeline



graphics pipeline overview

- simplified pipeline
- general pipeline ⟵ transformations!

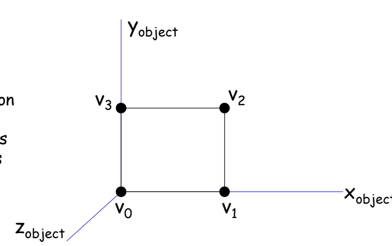
build scene



v_0= (0,0,0), v_1= (1,0,0), v_2= (1,1,0), v_3= (0,1,0)

build scene

pipeline representation is in homogeneous coordinates

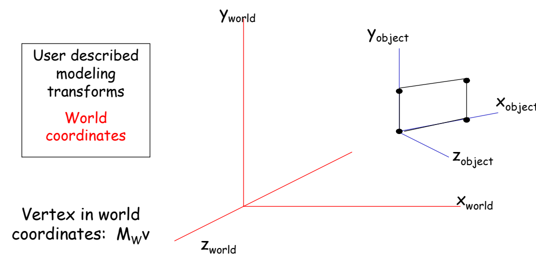


v_0= (0,0,0,1), v_1= (1,0,0,1), v_2= (1,1,0,1), v_3= (0,1,0,1)

primitives

- points
- line segments
- polygons

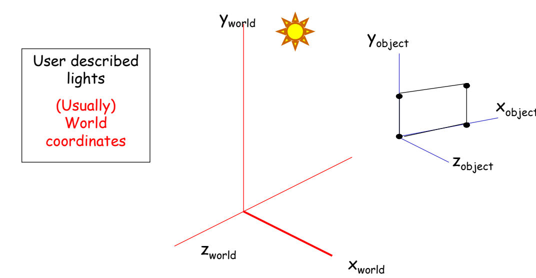
build scene



modeling transforms

- scale
- rotate
- translate

build scene



Lights

- Ambient
- Directional
- Point
- Spot

Exercise (teams of 2)

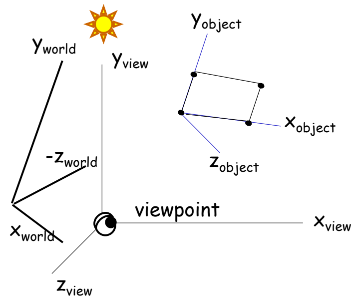
- Vertex in object coordinates: (1,1,2)
- Scale by 2 in x
- Translate by 3 in x and -4 in z
- What is result?
- Write sequence of modeling transform
- Multiply to get composite transform
- Multiply to get vertex in world coordinates
- Put your matrix on the board

build scene

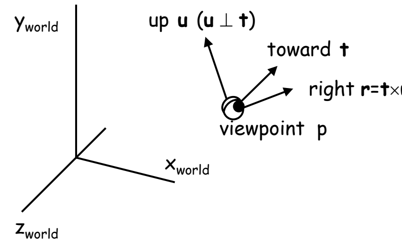
User defined viewpoint
View coordinates

vertex in view coordinates: $M_V M_W v$

lights in view coordinates: $M_v p$, $M_v \mathbf{d}$

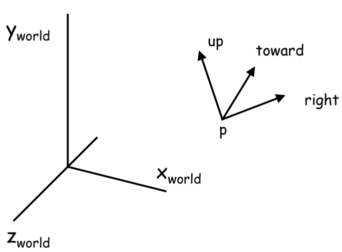


view in world coordinates



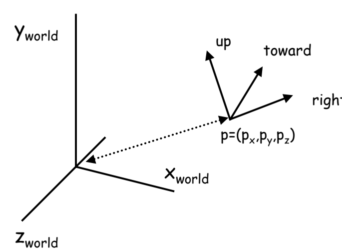
world ↔ view coordinates

translate & rotate: $M_V=M_R M_T$



world ↔ view coordinates

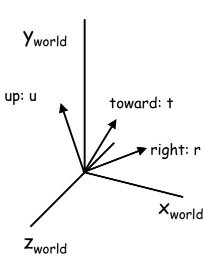
M_T: translate by $(-p_x,-p_y,-p_z)$



world ↔ view coordinates

rotation

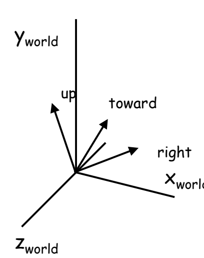
$M_R r=(1,0,0)^T$
$M_R u=(0,1,0)^T$
$M_R t=(0,0,-1)^T$



world ↔ view coordinates

rotation

$r=M_R^{-1}(1,0,0)^T$
$u=M_R^{-1}(0,1,0)^T$
$t=M_R^{-1}(0,0,-1)^T$



$$M_R^{-1}=\begin{pmatrix} r_x & u_x & -t_x \\ r_y & u_y & -t_y \\ r_z & u_z & -t_z \end{pmatrix}$$

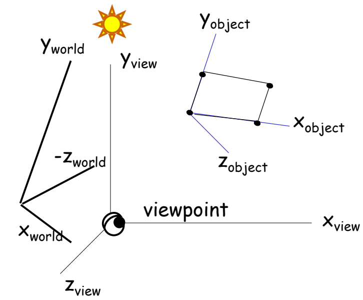
Exercise cont.

- The viewer is at (4,0,-2) looking in the (1,0,0) direction. Up is (0,1,0).
- What is our vertex in view coordinates?
- Write the translation and rotation matrices needed to convert to viewpoint coordinates. (You should be able to compute the inverse matrix by inspection.)
- Multiply to create the composite view transform.
- Multiply to get point in view coordinates.
- Write your view transform on the board.

build scene

vertex in view coordinates: $M_V M_W v$

lights in view coordinates: $M_v p$, $M_v \mathbf{d}$



geometric primitives

object coordinates: v — description of vertex

world coordinates: $M_W v$ — description of vertex situated in world

view coordinates: $M_V M_W v$ — description of vertex in world as seen from a particular viewpoint

lights

world coordinates: p, **d** — description of light position/direction in world

view coordinates: $M_V p$, $M_v \mathbf{d}$ — description of light position/direction in world as seen from a particular viewpoint

note: $M_v \mathbf{d}$ is shorthand for the "multiply vector" operation we've used before!

Application Programmer

Primitive in object coordinates →

```
glBegin(gTriangles);
glVertexf(-,-,-);
glVertexf(-,-,-);
glVertexf(-,-,-);
glEnd();
```

see Woo for details!

Application Programmer

Primitive in world coordinates

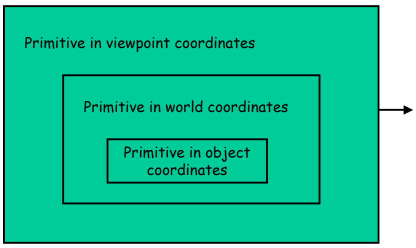
Primitive in object coordinates

→

```
glScalef(2,1,1)
glTranslatef(10,0,0);
glBegin(gTriangles);
glVertexf(-,-,-);
glVertexf(-,-,-);
glVertexf(-,-,-);
glEnd();
```

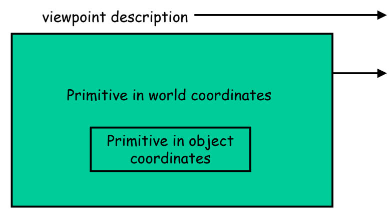
see Woo for details!

Application Programmer I



ViewpointTranforms
glScalef(-,-,-)
glTranslatef(-,-,-);
glBegin(gTriangles);
glVertexf(-,-,-);
glVertexf(-,-,-);
glVertexf(-,-,-);
glEnd();

Application Programmer II



gluLookAt(-,-,...)
glScalef(-,-,-)
glTranslatef(-,-,-);
glBegin(gTriangles);
glVertexf(-,-,-);
glVertexf(-,-,-);
glVertexf(-,-,-);
glEnd();

see Woo for details!

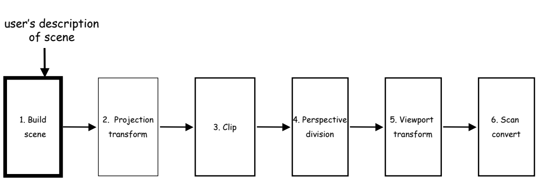
Modelview Matrix

glLoadIdentity() → I
gluLookAt(-,-,...) → I^*M_V
glScalef(-,-,-) → $I^*M_VM_1$
glTranslatef(-,-,-); → $I^*M_VM_1M_2$
glBegin(gTriangles);
glVertexf(-,-,-); → $IM_V\underbrace{M_1M_2}_{M_w}u$
glVertexf(-,-,-);
glVertexf(-,-,-);
glEnd();

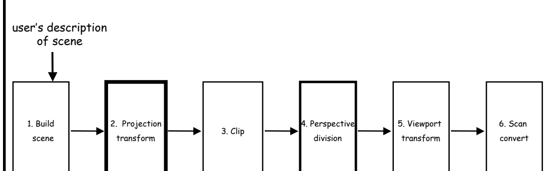
Modelview Matrix

glLoadIdentity() → I
gluLookAt(-,-,...) → I^*M_V
glPushMatrix() → I^*M_V
glScalef(-,-,-) → $I^*M_VM_1$
glTranslatef(-,-,-); → $I^*M_VM_1M_2$
glBegin(gTriangles);
glVertexf(-,-,-); → $IM_V\underbrace{M_1M_2}_{M_w}$
glVertexf(-,-,-);
glVertexf(-,-,-);
glEnd();
glPopMatrix() → I^*M_V

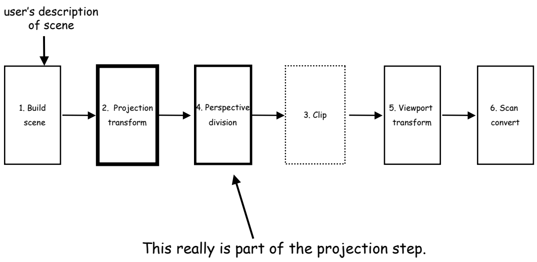
graphics pipeline



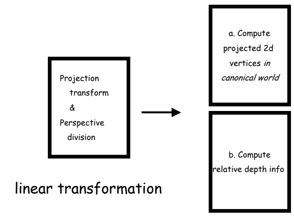
graphics pipeline



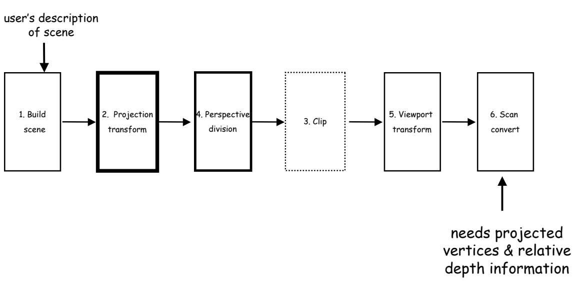
graphics pipeline – with a twist



Projection/Perspective Division



graphics pipeline – with a twist



Projection mode

- Orthographic projection
- Perspective projection

orthographic projection

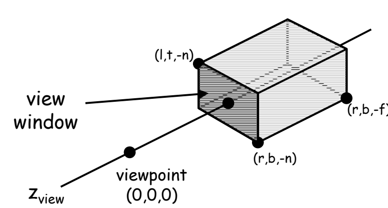
$$(x,y,z,1) \rightarrow (x',y',z',1)$$

where (x',y') is the vertex projected into a canonical 2x2 view window and z' is vertex's relative depth based on a canonical 2x2x2 world

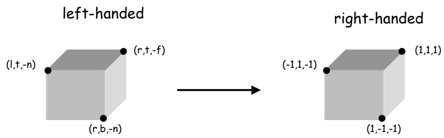
NOTE: for orthographic all we do is convert to the canonical world

orthographic view volume

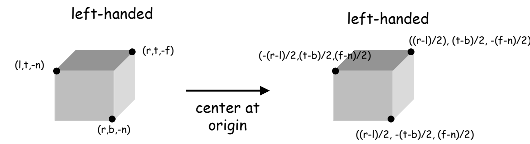
axes aligned parallelpiped



Canonical world

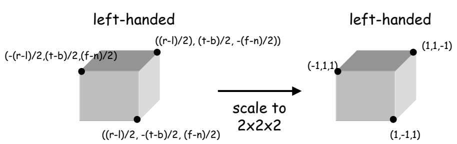


center at origin



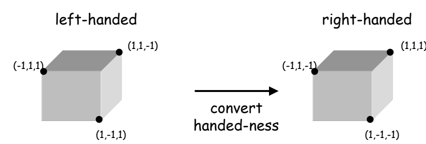
1. translate by (-(r+l)/2, -(t+b)/2, (f+n)/2))

scale to 2x2x2



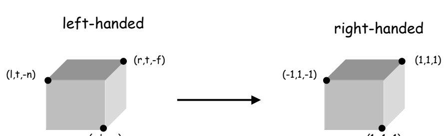
2. scale by (2/(r-l), 2/(t-b), 2/(f-n))

change handed-ness



3. scale by (0,0,-1)

Canonical world transform



1. translate by (-(r+l)/2, -(t+b)/2, (f+n)/2))
2&3. scale by (2/(r-l), 2/(t-b), -2/(f-n))

Canonical world transform

$$\underbrace{\begin{pmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & -2/(f-n) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{scale}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & -(r+l)/2 \\ 0 & 1 & 0 & -(t+b)/2 \\ 0 & 0 & 1 & (f+n)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{translate}} = \underbrace{\begin{pmatrix} 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\ 0 & 2/(t-b) & 0 & -(t+b)/(t-b) \\ 0 & 0 & -2/(f-n) & -(f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M_p}$$

orthographic projection matrix

$$M_p = \begin{pmatrix} 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\ 0 & 2/(t-b) & 0 & -(t+b)/(t-b) \\ 0 & 0 & -2/(f-n) & -(f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

orthographic projection

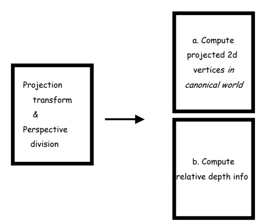
$$(x,y,z,1) \xrightarrow{M_p} (x',y',z',1)$$

where (x',y') is the vertex projected into a canonical 2x2 view window and z' is vertex's relative depth based on a canonical 2x2x2 world

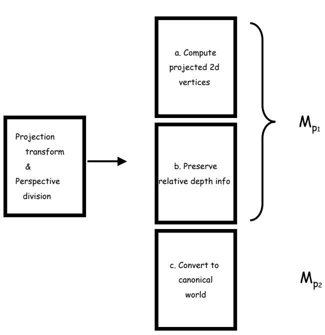
Projection mode

- Orthographic projection
- **Perspective projection**

Projection/Perspective Division

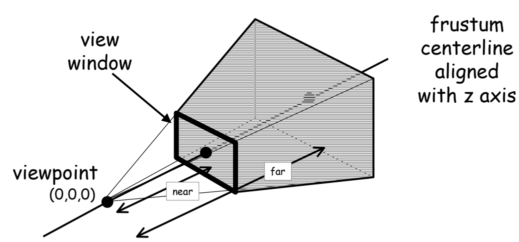


Perspective mode



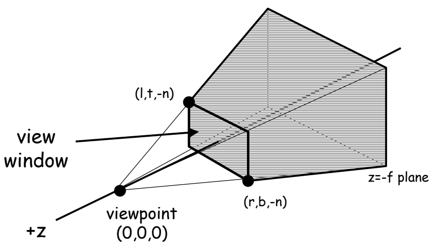
perspective view volume

in view coordinates

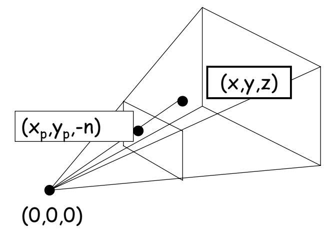


perspective view volume (frustum)

in view coordinates

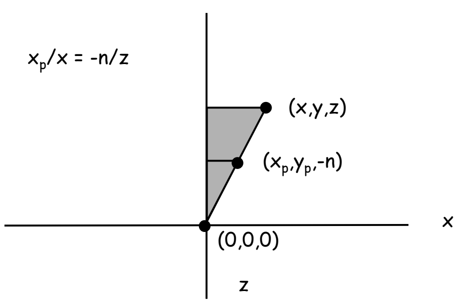


perspective projection

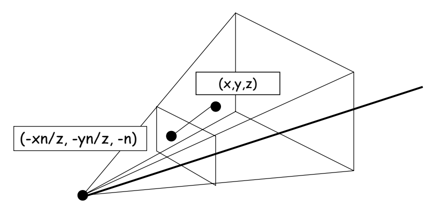


x-projection

$x_p/x = -n/z$



perspective projection



2a. compute projected 2d vertices: (-xn/z,-yn/z)

2b. preserve relative depth information: z

$M_{p_1} = ?$

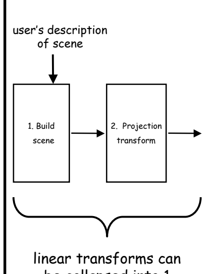
perspective M_{p_1}

what is wrong with this picture

$$\begin{pmatrix} -n/z & 0 & 0 & 0 \\ 0 & -n/z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} -nx/z \\ -ny/z \\ z \\ 1 \end{pmatrix}$$

M_{p_1}?

graphics pipeline



$v \rightarrow Mv$ where $M=M_p\underbrace{M_vM_w}_{\text{remain fixed for all vertices in scene}}$

perspective M_{p_1}

a little trick!

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ az-b \\ -z \end{pmatrix}$$

we'll specify a & b in a moment

remember scale factor -z in w-component

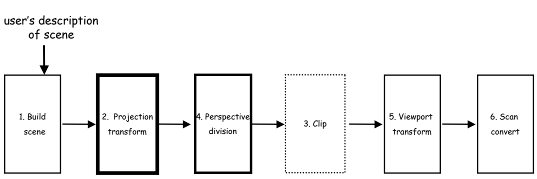
perspective M_{p_1}

a little trick!

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ az+b \\ -z \end{pmatrix} \rightarrow \begin{pmatrix} -nx/z \\ -ny/z \\ -a - b/z \\ 1 \end{pmatrix}$$

later we'll normalize by w-component of vertex

graphics pipeline – with a twist



later is step 4

perspective M_{p_1}

a little trick!

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ az+b \\ -z \end{pmatrix} \rightarrow \begin{pmatrix} -nx/z \\ -ny/z \\ -a - b/z \\ 1 \end{pmatrix}$$

this is where the relative depth info will be

-b/z is (almost) as good as z for any nonzero constant b so let a=0 and b=fn!

perspective M_{p_1}

a little trick!

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & 0 & fn \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ fn \\ -z \end{pmatrix} \rightarrow \begin{pmatrix} -nx/z \\ -ny/z \\ -fn/z \\ 1 \end{pmatrix}$$

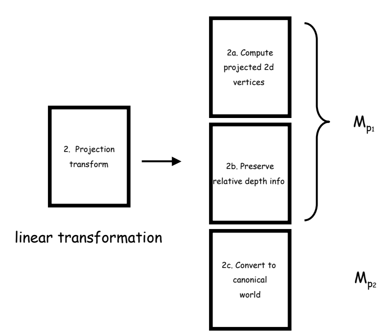
if the vertex has been clipped then

z takes on values in [-n,-f]

-fn/z takes on values in [n,f]

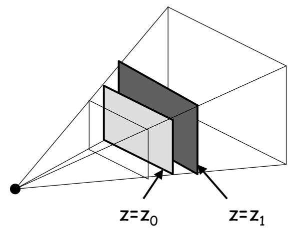
(We switched to right-handed coordinates … we'll fix this in a minute.)

Projection ABC



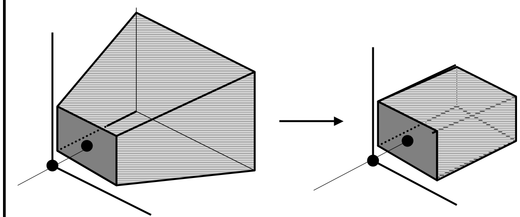
perspective projection

$(x,y,z) \rightarrow (-xn/z,\ \ -yn/z,\ \ -fn/z)$



depth-dependent scale

depth dependent x,y scale



perspective projection

1. M_{p_1}
2. perspective division
3. orthographic M_{p_2}

this would work

alternative

1. M_{p_1}
2. orthographic M_{p_2}
3. perspective division

does this work?

sure – perspective division is multiplication by a scalar

(albeit a special one)

perspective projection matrix

$$\begin{pmatrix} 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\ 0 & 2/(t-b) & 0 & -(t+b)/(t-b) \\ 0 & 0 & -2/(f-n) & -(f+n)/(f-n) \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & 0 & fn \\ 0 & 0 & -1 & 0 \end{pmatrix} =$$

$$\begin{pmatrix} 2n/(r-l) & 0 & (r+l)/(r-l) & 0 \\ 0 & 2n/(t-b) & (t+b)/(t-b) & 0 \\ 0 & 0 & (f+n)/(f-n) & -2fn/(f-n) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

except we're back in left-handed coordinates

projection matrix: M_P

$$\begin{pmatrix} -2n/(r-l) & 0 & (r+l)/(r-l) & 0 \\ 0 & -2n/(t-b) & (t+b)/(t-b) & 0 \\ 0 & 0 & -(f+n)/(f-n) & -2fn/(f-n) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

geometric primitives

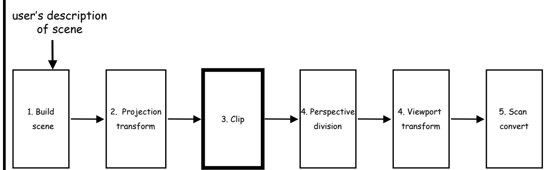
object coordinates: v — description of vertex

world coordinates: $M_W v$ — description of vertex situated in world

view coordinates: $M_V M_W v$ — description of vertex in world as seen from a particular viewpoint

clip coordinates: $M_p M_V M_W v$ — description of vertex seen from viewpoint in a normalized world

graphics pipeline



can we clip before perspective division?

yes – otherwise big problems

interpolate w just like other coordinates

Application programmer

Define projection mode and view volume.

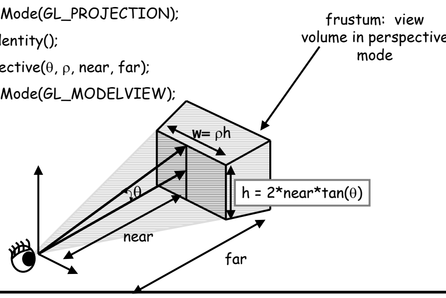
Example:

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluPerspective(θ, ρ, near, far);

glMatrixMode(GL_MODELVIEW);



Application programmer

Define projection mode and view volume.

Example:

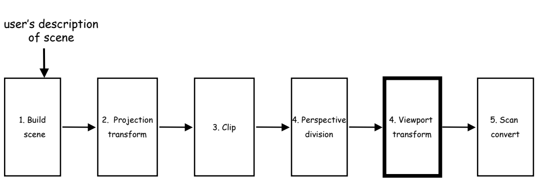
glMatrixMode(GL_PROJECTION);

glLoadIdentity();

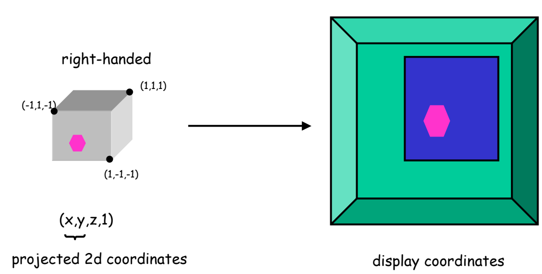
gluPerspective(θ, ρ, near, far); → glFrustum(...) or glOrtho(...)

glMatrixMode(GL_MODELVIEW);

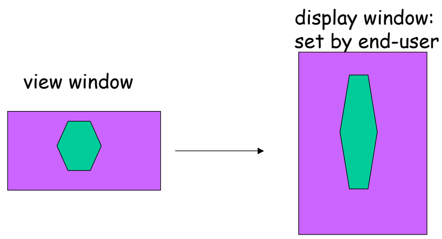
graphics pipeline



viewport transormation

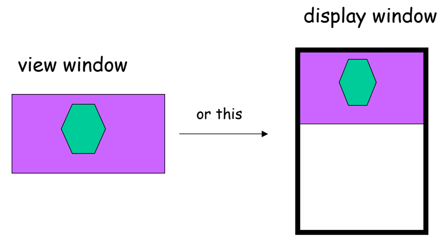


viewport transformation



aspect ratio may be different

viewport transformation

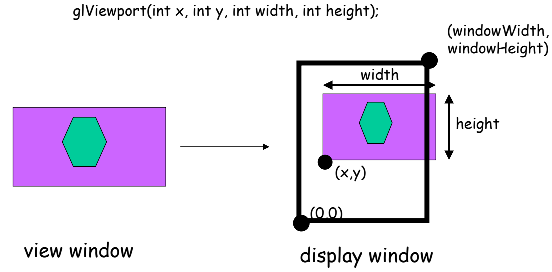


viewport transformation

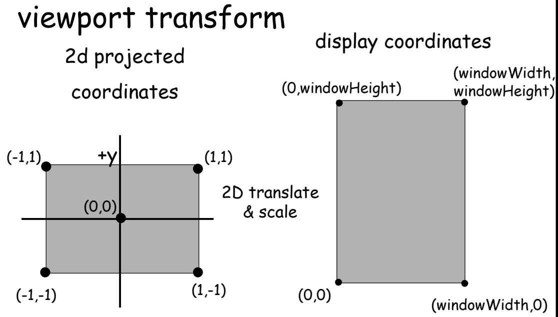
specifies how the projected world is mapped to the display window

Application Programmer

```
glViewport(int x, int y, int width, int height);
```



viewport transform



We'll call this M_I

geometric primitives

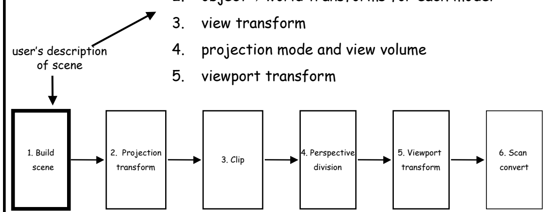
object coordinates: v	description of vertex
world coordinates: $M_W v$	description of vertex situated in world
view coordinates: $M_V M_W v$	description of vertex in world as seen from a particular viewpoint
clip coordinates: $M_p M_V M_W v$	description of vertex seen from viewpoint in a normalized world
display coordinate $M_I M_p M_V M_W v$	description of a vertex in display coordinates

Application Programmer

object coordinates: v	description of vertex
world coordinates: $M_W v$	object→world specified by transforms in scene graph
view coordinates: $M_V M_W v$	view transform: specified where the viewer (view volume) located/oriented
clip coordinates: $M_P M_V M_W v$	projection transform: specified by description of the view volume
display coordinate $M_I M_P M_V M_W v$	viewport transform: specified by viewport command

graphics pipeline

1. models
2. object→ world transforms for each model
3. view transform
4. projection mode and view volume
5. viewport transform



Application Programmer

- Initialization
- Display
- Resize

Initialization

- Create window (use GLUT)
- Define projection mode and view volume
- Define viewport transform

Display function: draws scene

Display Function:

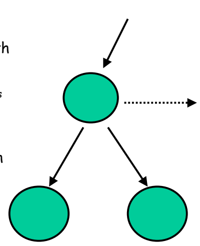
Clear frame and depth buffers

Load Identity (default is Modelview mode)

Load view transform

Traverse scene graph

Swap buffers



Push Matrix()

Local Transforms

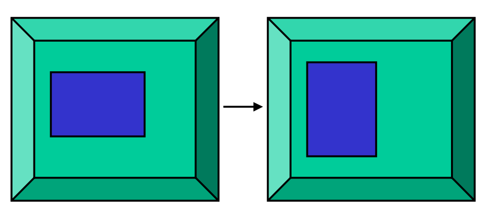
Draw local model

Visit children

Pop Matrix()

Return to parent node

Resize



1. May change view volume
2. Change viewport

Now do it!