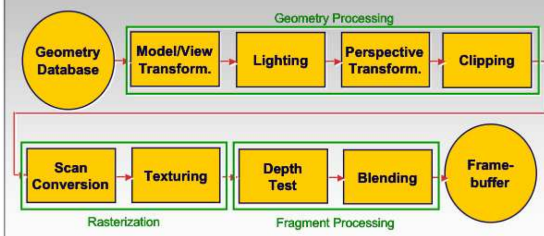
Perspective Projection

CPSC 314

The Rendering Pipeline



Homogeneous Coordinates

Homogeneous representation of points:

- Add an additional component $w=1$ to all *points*
- All multiples of this vector are considered to represent the same 3D point
- All points are represented as *column vectors*

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x \cdot w \\ y \cdot w \\ z \cdot w \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix}, \forall w \neq 0$$

Homogeneous Matrices

Affine Transformations

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous Vectors

Representing vectors in homogeneous coordinates

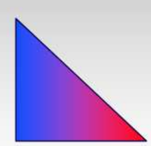
- Column vectors with $w=0$

$$T\left(\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}\right) = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & t_x \\ m_{2,1} & m_{2,2} & m_{2,3} & t_y \\ m_{3,1} & m_{3,2} & m_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix}$$

Rendering Geometry in OpenGL

Example:

```
glBegin( GL_TRIANGLES );
    glColor3f( 1.0, 0.0, 0.0 );
    glVertex3f( 1.0, 0.0, 0.0 );
    glColor3f( 0.0, 0.0, 1.0 );
    glVertex3f( 1.0, 0.0, 0.0 );
    glVertex3f(0.0, 0.0, 0.0 );
glEnd();
```



Matrix Operations in OpenGL

Specifying matrices (replacement)

- glLoadIdentity()
- glLoadMatrixf(GLfloat *m) // 16 floats

Specifying matrices (multiplication)

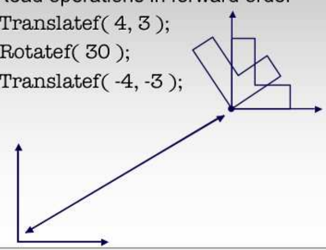
- glMultMatrixf(GLfloat *m) // 16 floats
- glRotatef(GLfloat angle, GLfloat x, Glfloat y, GLfloat z) // angle and axis
- glScalef(GLfloat x, GLfloat y, GLfloat z)
- glTranslatef(GLfloat x, GLfloat y, GLfloat z)

Interpreting Composite Transformations

Interpretation 1: moving the coordinate system

- Read operations in forward order

```
glTranslatef( 4, 3 );
glRotatef( 30 );
glTranslatef( -4, -3 );
```

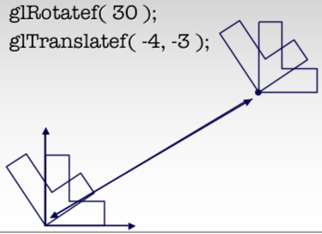


Interpreting Composite Transformations

Interpretation 2: moving the object

- Read operations in reverse order

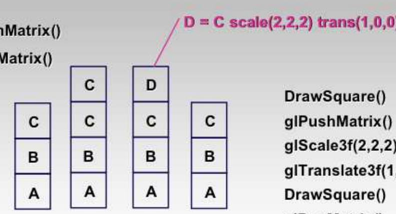
```
glTranslatef( 4, 3 );
glRotatef( 30 );
glTranslatef( -4, -3 );
```



Matrix Stacks

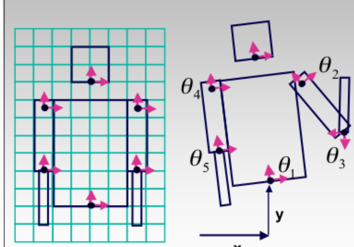
glPushMatrix()

glPopMatrix()




```
DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()
```


Transformation Hierarchy Example 4




```
glTranslate3f(x,y,0);
glRotatef( θ1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslate3f(0,7,0);
  DrawHead();
glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef( θ2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef( θ3,0,0,1);
  DrawLArm();
glPopMatrix();
... (draw other arm)
```


Display Lists

Concept:

- If multiple copies of an object are required, it can be compiled into a display list:

```
glNewList( listId, GL_COMPILE );
    glBegin( …);
    … // geometry goes here
glEndList();
// render two copies of geometry offset by 1 in z-direction:
glCallList( listId );
glTranslatef( 0.0, 0.0, 1.0 );
glCallList( listId) ;
```

Display Lists

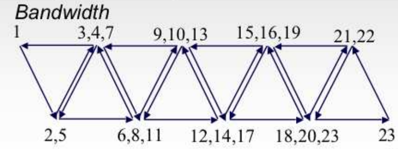
Advantages:

- More efficient than individual function calls for every vertex/attribute
- Can be cached on the graphics board (bandwidth!)
- Display lists exist across multiple frames
  - *Represent static objects in an interactive application*

Shared Vertices

Triangle Meshes

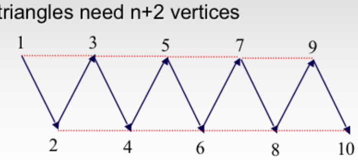
- Multiple triangles share vertices
- If individual triangles are sent to graphics board, every vertex is sent and transformed multiple times!
  - *Computational expense*
  - *Bandwidth*



Triangle Strips

Idea:

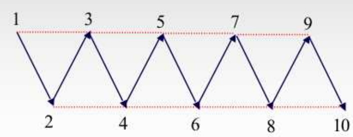
- Encode neighboring triangles that share vertices
- Use an encoding that requires only a constant-sized part of the whole geometry to determine a single triangle
- N triangles need n+2 vertices



Triangle Strips

Orientation:

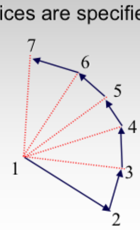
- Strip starts with a counter-clockwise triangle
- Then alternates between clockwise and counter-clockwise



Triangle Fans

Similar concept:

- All triangles share on center vertex
- All other vertices are specified in CCW order



Triangle Strips and Fans

Transformations:

- n+2 for n triangles
- Only requires 3 vertices to be stored according to simple access scheme
- Ideal for pipeline (local knowledge)

Generation

- E.g. from directed edge data structure
- Optimize for longest strips/fans

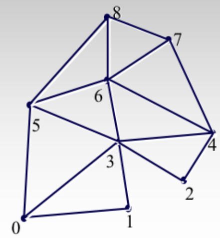
Strippification by Dana Sharon

Vertex Arrays

Concept:

- Store array of vertex data for meshes with aribtrary connectivity (topology)

```
GLfloat *points[3*nvertices];
GLfloat *colors[3*nvertices];
Glint *tris[numtris]=
    {0,1,3,  3,2,4, …};
glVertexPointer( …, points );
glColorPointer( …,colors );
glDrawElements(
    GL_TRIANGLES,…,tris );
```



Vertex Arrays

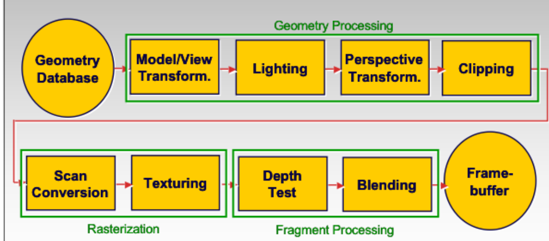
Benefits:

- Ideally, vertex array fits into memory on graphics chip
- Then all vertices are transformed exactly once

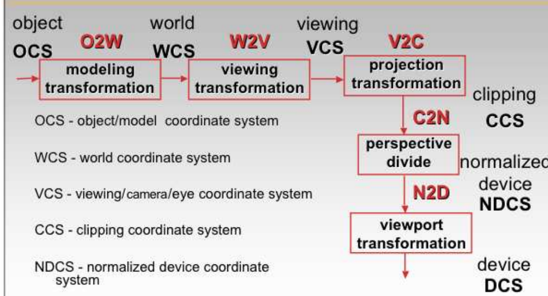
In practice:

- Graphics memory may not be sufficient to hold model
- Then either:
  - *Cache only parts of the vertex array on board (may lead to cache trashing!)*
  - *Transform everything in software and just send results for individual triangles (bandwidth problem: multiple transfers of same vertex!)*

The Rendering Pipeline



Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

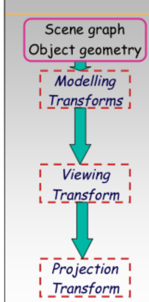
VCS - viewing/camera/eye coordinate system

CCS - clipping coordinate system

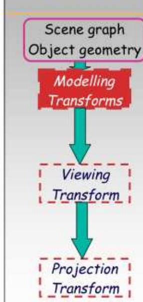
NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system

Rendering Pipeline

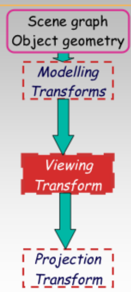


Rendering Pipeline



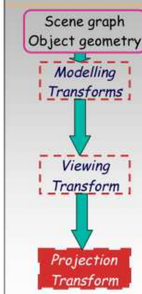
- result
  - all vertices of scene in shared 3D world coordinate system

Rendering Pipeline

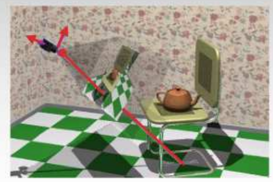


- result
  - scene vertices in 3D view (camera) coordinate system

Rendering Pipeline



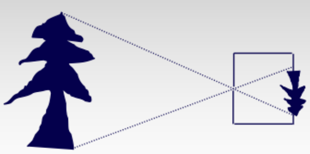
- result
  - 2D screen coordinates of clipped vertices



Perspective Transformation

Pinhole Camera:

- Light shining through a tiny hole into a dark room yields upside-down image on wall



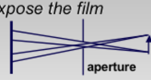
Perspective Transformation

Pinhole Camera

Real Cameras

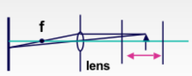
- pinhole camera has small aperture (lens opening)
  - *hard to get enough light to expose the film*

real pinhole camera



- lens permits larger apertures
- lens permits changing distance to film plane without actually moving the film plane

camera

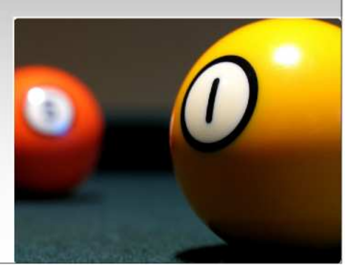


price to pay: limited depth of field

Real Cameras - Depth of Field

Limited depth of field

- Can be used to direct attention
- Artistic purposes



Perspective Transformation

In computer graphics:

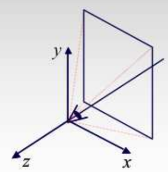
- Image plane is conceptually *in front* of the center of projection

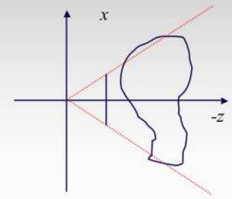
- Perspective transformations belong to a class of operations that are called *projective transformations*
- Linear and affine transformations also belong to this class
- *All* projective transformations can be expressed as *4x4* matrix operations

Perspective Projection

Synopsis:

- Project all geometry through a common *center of projection* (*eye point*) onto an *image plane*

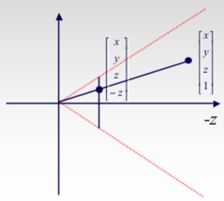




Perspective Projection

Example:

- Assume image plane at $z=-1$
- A point $[x,y,z,1]^T$ projects to $[-x/z,-y/z,-z/z,1]^T \equiv [x,y,z,-z]^T$



Perspective Projection

Analysis:

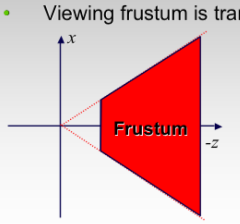
- This is a special case of a general family of transformations called *projective transformations*
- These can be expressed as 4x4 homogeneous matrices!
  - *E.g. in the example:*

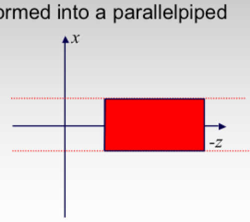
$$T\left(\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z \end{bmatrix} \equiv \begin{bmatrix} -x/z \\ -y/z \\ -1 \\ 1 \end{bmatrix}$$

Projective Transformations

Transformation of space:

- Center of projection moves to infinity
- Viewing frustum is transformed into a parallelpiped





Projective Transformations

Convention:

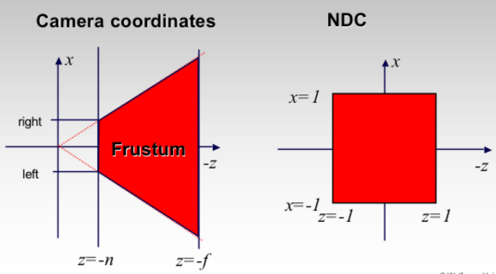
- Viewing frustum is mapped to a specific parallelpiped
  - *Normalized Device Coordinates (NDC)*
- Only objects inside the parallelpiped get rendered
- Which parallelpied is used depends on the rendering system

OpenGL:

- Left and right image boundary are mapped to $x=-1$ and $x=+1$
- Top and bottom are mapped to $y=-1$ and $y=+1$
- Near and far plane are mapped to -1 and 1

Projective Transformations

OpenGL Convention



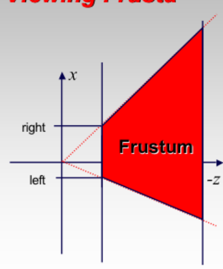
Projective Transformations

Why near and far plane?

- Near plane:
  - *Avoid singularity (division by zero, or very small numbers)*
- Far plane:
  - *Store depth in fixed-point representation (integer), thus have to have fixed range of values (0…1)*
  - *Avoid/reduce numerical precision artifacts for distant objects*

Projective Transformations

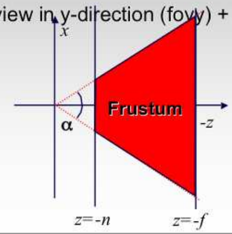
Asymmetric Viewing Frusta



Projective Transformations

Alternative specification of symmetric frusta

- Field-of-view (fov) α
- Fov/2
- Field-of-view in y-direction (fovy) + aspect ratio



Demos

Tuebingen applets from Frank Hanisch

- http://www.gris.uni-tuebingen.de/projects/grdev/doc/html/etc/AppletIndex.html#Transformationen

Projective Transformations

Properties:

- All transformations that can be expressed as homogeneous 4x4 matrices (in 3D)
- 16 matrix entries, but multiples of the same matrix all describe the same transformation
  - *15 degrees of freedom*
  - *The mapping of 5 points uniquely determines the transformation*

Projective Transformations

Determining the matrix representation

- Need to observe 5 points in general position, e.g.
  - $[left,0,0,1]^T \rightarrow [1,0,0,1]^T$
  - $[0,top,0,1]^T \rightarrow [0,1,0,1]^T$
  - $[0,0,-f,1]^T \rightarrow [0,0,1,1]^T$
  - $[0,0,-n,1]^T \rightarrow [0,0,0,1]^T$
  - $[left*f/n,top*f/n,-f,1]^T \rightarrow [1,1,1,1]^T$
- Solve resulting equation system to obtain matrix

Perspective Derivation

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x' = Ex + Az \quad y' = Fy + Bz \quad z' = Cz + D \quad w' = -z$$

$$x = left \rightarrow x'/w' = 1$$
$$x = right \rightarrow x'/w' = -1$$
$$y = top \rightarrow y'/w' = 1$$
$$y = bottom \rightarrow y'/w' = -1$$
$$z = -near \rightarrow z'/w' = 1$$
$$z = -far \rightarrow z'/w' = -1$$

$$y' = Fy + Bz, \quad \frac{y'}{w'} = \frac{Fy + Bz}{w'}, \quad 1 = \frac{Fy + Bz}{w'}, \quad 1 = \frac{Fy + Bz}{-z},$$

$$1 = F\frac{y}{-z} + B\frac{z}{-z}, \quad 1 = F\frac{y}{-z} - B, \quad 1 = F\frac{top}{-(-near)} - B,$$

$$1 = F\frac{top}{near} - B$$

Perspective Derivation

similarly for other 5 planes
6 planes, 6 unknowns

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

Perspective Example

view volume
left = -1, right = 1
bot = -1, top = 1
near = 1, far = 4

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -5/3 & -8/3 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Projective Transformations

Properties

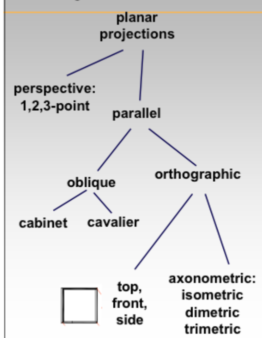
- Lines are mapped to lines and triangles to triangles
- Parallel lines do NOT remain parallel
  - *E.g. rails vanishing at infinity*
- Affine combinations are NOT preserved
  - *E.g. center of a line does not map to center of projected line (perspective foreshortening)*

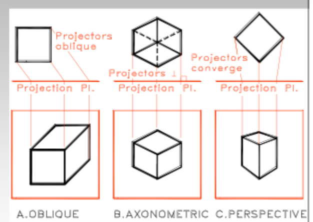
Orthographic Camera Projection

- Camera's back plane parallel to lens
- Infinite focal length
- No perspective convergence
- Just throw away z values

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

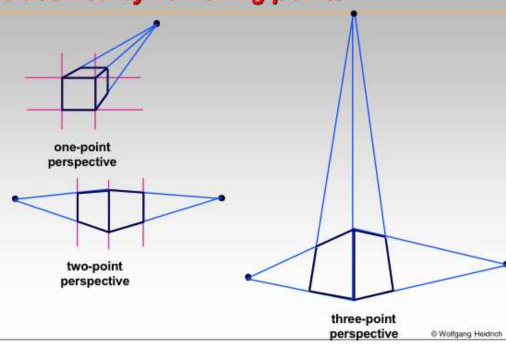
Projection Taxonomy





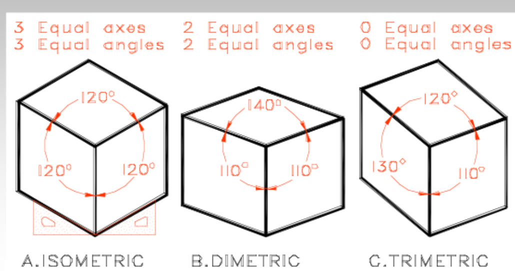
http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Perspective Projections
classified by vanishing points



Axonometric Projections

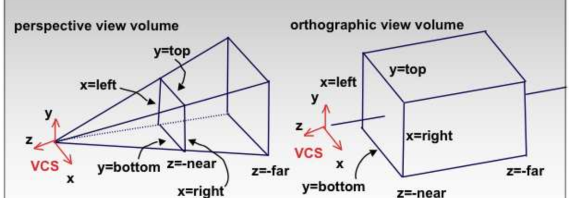
- projectors perpendicular to image plane



http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

View Volumes

- specifies field-of-view, used for clipping
- restricts domain of ***z*** stored for visibility test



View Volume

Convention

- Viewing frustum mapped to specific parallelepiped
  - *Normalized Device Coordinates (NDC)*
  - *Same as clipping coords*
- Only objects inside the parallelepiped get rendered
- Which parallelepiped?
  - *Depends on rendering system*

Perspective Matrices in OpenGL

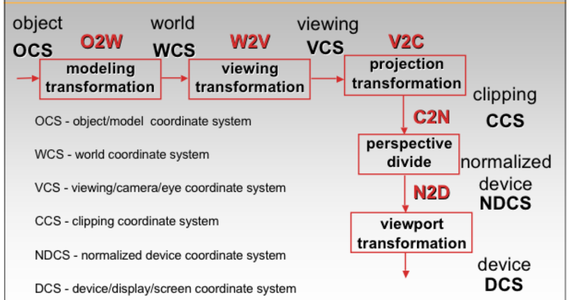
Perspective Matrices:

- glFrustum(left, right, bottom, top, near, far)
  - *Specifies perspective xform (near, far are always positive)*
- glOrtho(left, right, bottom, top, near, far)

Convenience Functions:

- gluPerspective(fovy, aspect, near, far)
  - *Another way to do perspective*
- gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
  - *Useful for viewing transform*

Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system

Window-To-Viewport Transformation

Generate pixel coordinates

- Map x, y from range –1…1 (*normalized device coordinates*) to pixel coordinates on the screen
- Map z from -1…1 to 0…1 (used later for visibility)
- Involves 2D scaling and translation