Perspective Projection

Wolfgang Heidrich

Course News

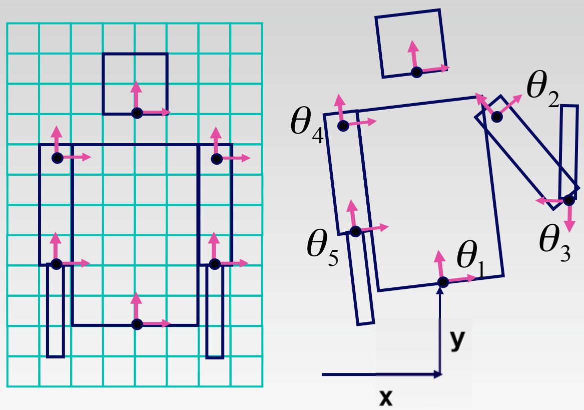
Assignment 1
- Due February 2

Homework 1
- Discussed in labs this week

Homework 2
- Exercise problems for perspective
- Discussed in labs next week

Reading
- Chapter 6

Recap: Transformation Hierarchies




```
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)
```


Hierarchical Modeling

Advantages

- Define object once, instantiate multiple copies
- Transformation parameters often good control knobs
- Maintain structural constraints if well-designed

Limitations

- Expressivity: not always the best controls
- Can't do closed kinematic chains
  - *Keep hand on hip*

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) ;
```

Wolfgang Heidrich

Display Lists

Advantages:

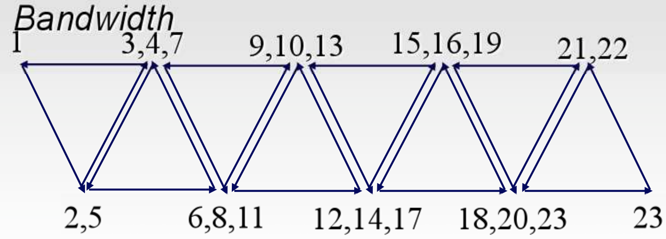
- More efficient than individual function calls for every vertex/attribute
- Can be cached on the graphics board (bandwidth!)
- Display lists exist across multiple frames
  - *Represent static objects in an interactive application*

Wolfgang Heidrich

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*

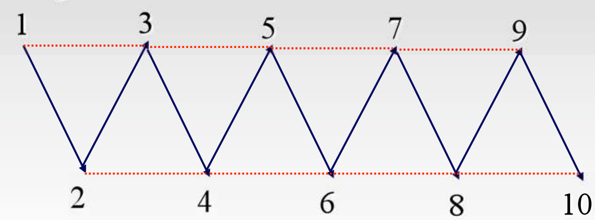


Wolfgang Heidrich

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

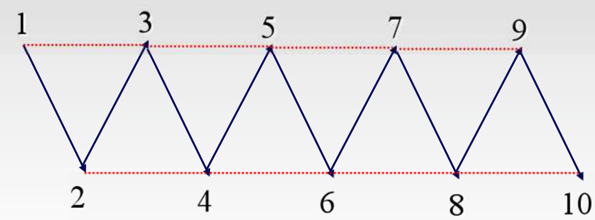


Wolfgang Heidrich

Triangle Strips

Orientation:

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

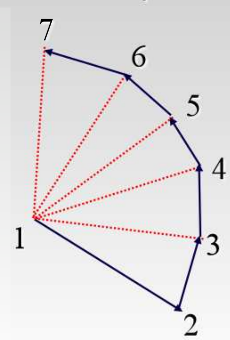


Wolfgang Heidrich

Triangle Fans

Similar concept:

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



Wolfgang Heidrich

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

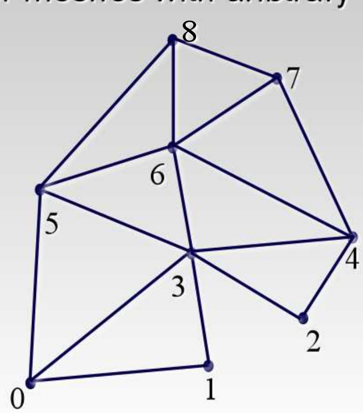
Wolfgang Heidrich

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 );
```



Wolfgang Heidrich

Vertex Arrays

Benefits:

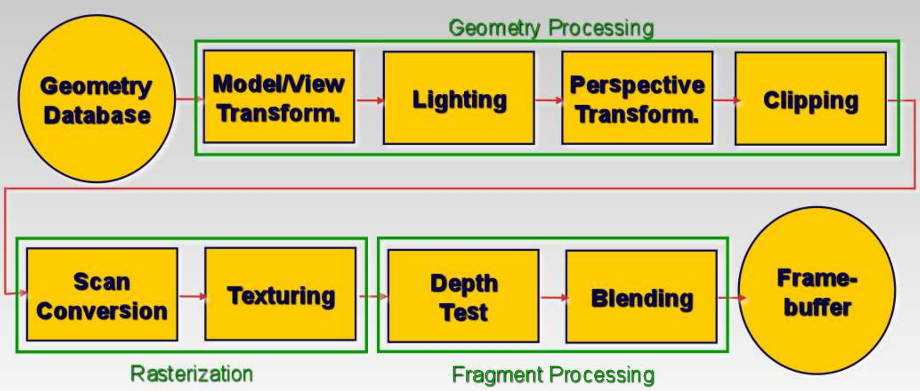
- Ideally, vertex array fits into memory on GPU
- Then all vertices are transformed exactly once

In practice:

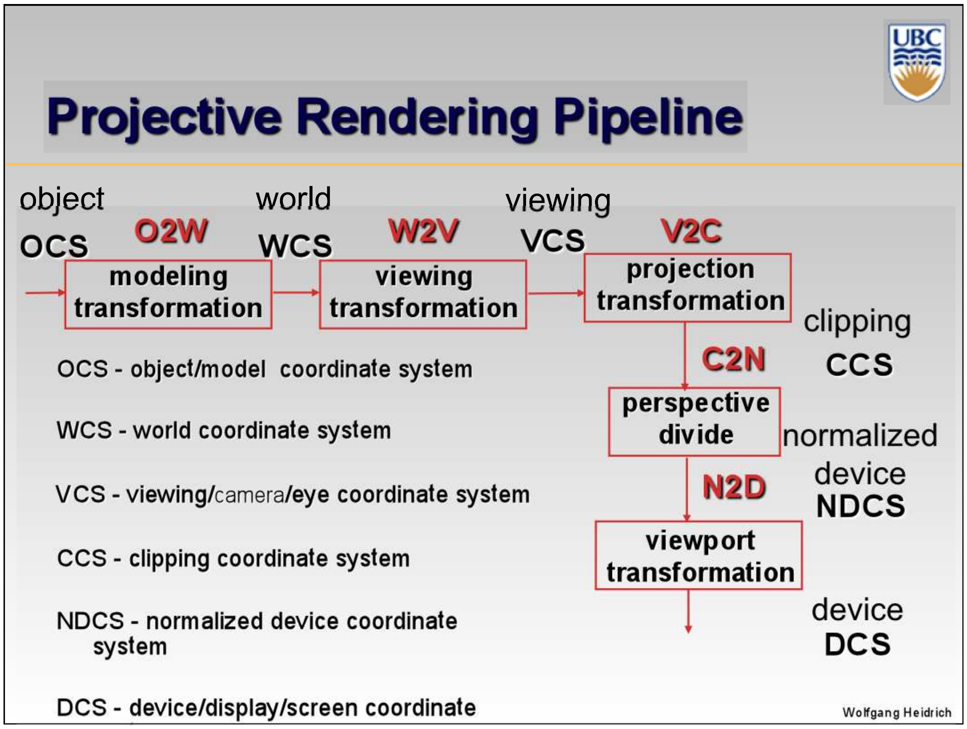
- Graphics memory may not be sufficient to hold model
- Then either:
  - *Cache only parts of the vertex array on board (may lead to cache trashing!)*
  - *Transform everything in software and just send results for individual triangles (bandwidth problem: multiple transfers of same vertex!)*

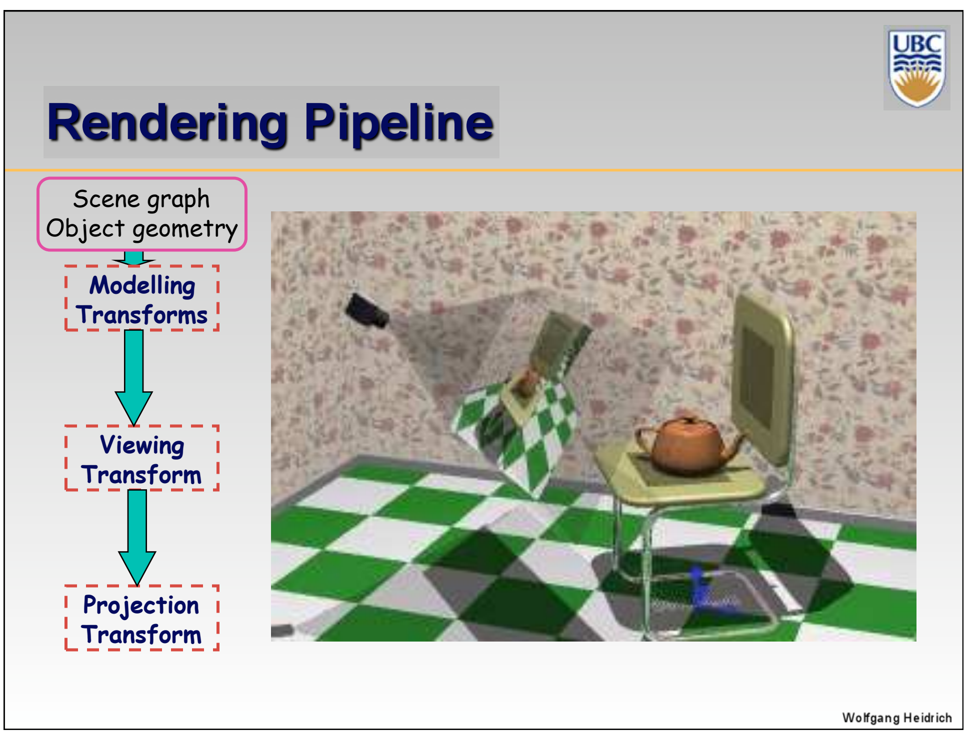
Wolfgang Heidrich

The Rendering Pipeline

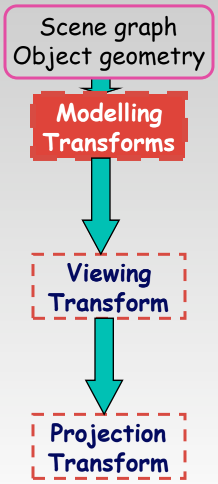


Wolfgang Heidrich


Projective Rendering Pipeline
object
OCS
O2W
modeling transformation
world
WCS
W2V
viewing transformation
viewing
VCS
V2C
projection transformation
clipping
CCS
C2N
perspective divide
normalized device
NDCS
N2D
viewport transformation
device
DCS
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
Wolfgang Heidrich

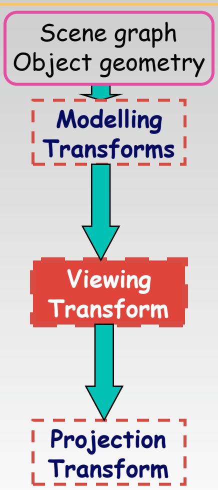

Rendering Pipeline
Scene graph
Object geometry
Modelling Transforms
Viewing Transform
Projection Transform
Wolfgang Heidrich

Rendering Pipeline



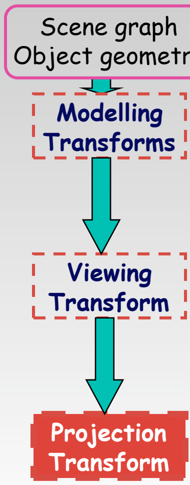
- result
  - all vertices of scene in shared 3D world coordinate system

Rendering Pipeline

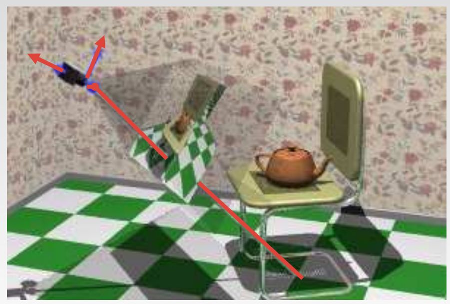


- result
  - scene vertices in 3D view (camera) coordinate system

Rendering Pipeline

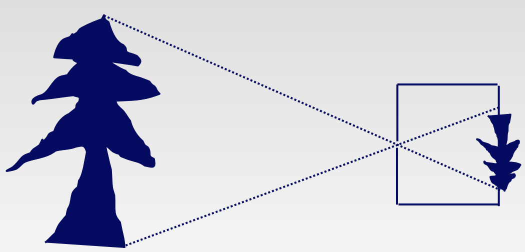


- result
  - 2D screen coordinates of clipped vertices



Perspective Transformation

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



Perspective Transformation

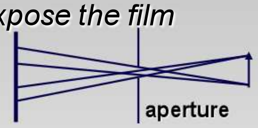
- ***Pinhole Camera***

Wolfgang Heidrich

Real Cameras

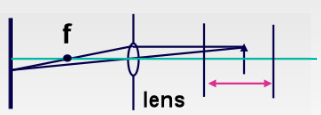
- pinhole camera has small aperture (lens opening)
  - *hard to get enough light to expose the film*

real pinhole camera



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

camera

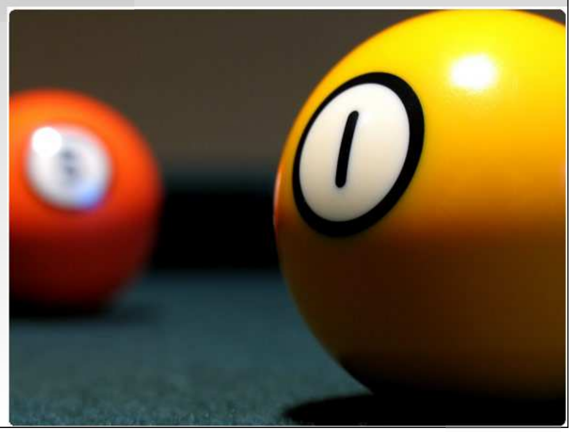


price to pay: limited depth of field

Wolfgang Heidrich

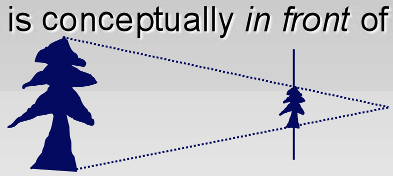
Real Cameras - Depth of Field

- ***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

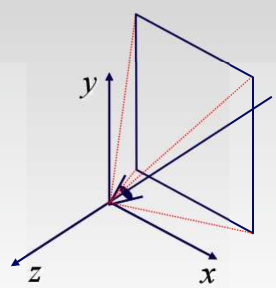


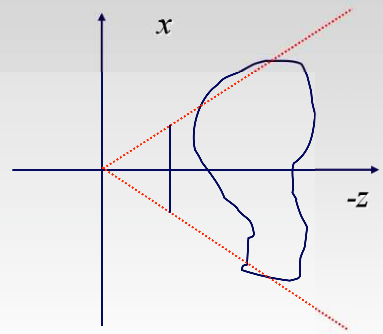
  - Perspective transformations belong to a class of operations that are called *projective transformations*
  - Linear and affine transformations also belong to this class
  - *All* projective transformations can be expressed as *4x4* matrix operations

Wolfgang Heidrich

Perspective Projection

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

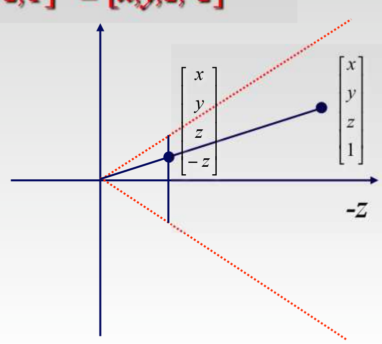




Wolfgang Heidrich

Perspective Projection

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



Wolfgang Heidrich

Perspective Projection

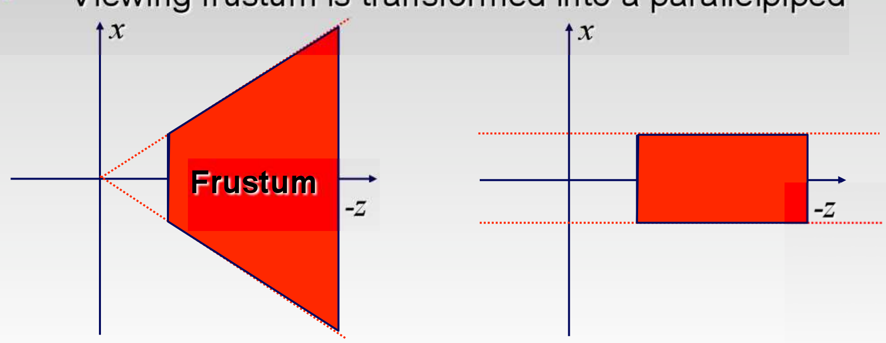
- ***Analysis:***
  - This is a special case of a general family of transformations called *projective transformations*
  - These can be expressed as 4x4 homogeneous matrices!
    - *E.g. in the example:*

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

Wolfgang Heidrich

Projective Transformations

- ***Transformation of space:***
  - Center of projection moves to infinity
  - Viewing frustum is transformed into a parallelpiped



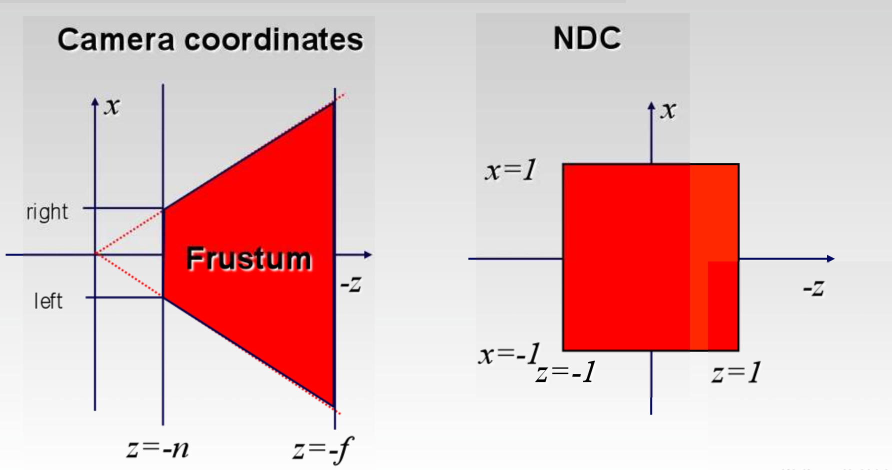
Wolfgang Heidrich

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***



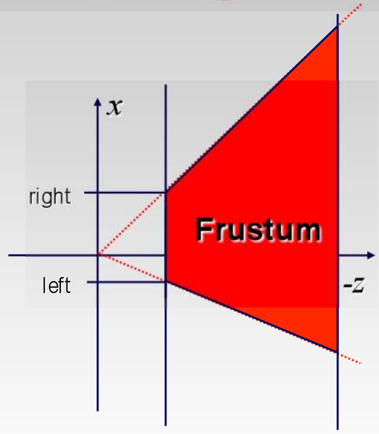
Projective Transformations

- ***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*

Wolfgang Heidrich

Projective Transformations

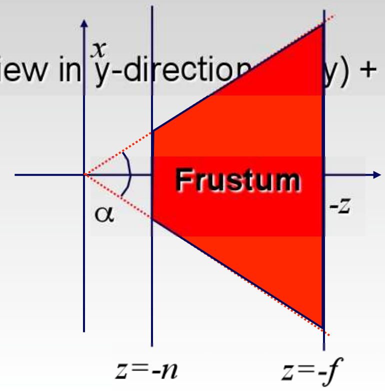
- ***Asymmetric Viewing Frusta***



Wolfgang Heidrich

Projective Transformations

- ***Alternative specification of symmetric frusta***
  - Field-of-view (fov) α
  - Fov/2
  - Field-of-view in y-direction [illegible]y) + aspect ratio



Wolfgang Heidrich

Demos

- ***Tuebingen applets from Frank Hanisch***
  - http://www.gris.uni-tuebingen.de/projects/grdev/doc/html/etc/AppletIndex.html#Transformationen

Wolfgang Heidrich

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*

Wolfgang Heidrich

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

Wolfgang Heidrich

Perspective Derivation

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

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

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

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

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

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

Wolfgang Heidrich

Perspective Derivation

- *similarly for other 5 planes*
- *6 planes, 6 unknowns*

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

Wolfgang Heidrich

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)*

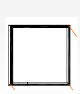
Orthographic Camera Projection

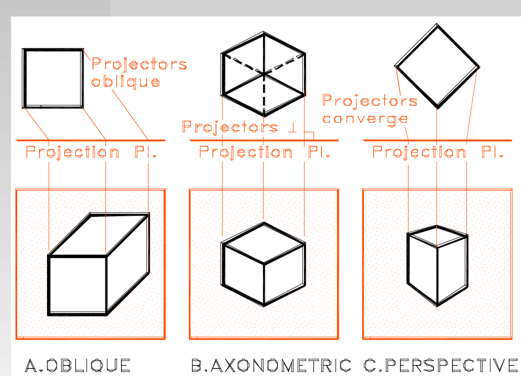
- Camera's back plane parallel to lens
- Infinite focal length
- No perspective convergence
- Just throw away z values

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

Wolfgang Heidrich

Projection Taxonomy

- **planar projections**
  - **perspective: 1,2,3-point**
  - **parallel**
    - **oblique**
      - **cabinet**
      - **cavalier**
    - **orthographic**
      -  **top, front, side**
      - **axonometric: isometric dimetric trimetric**

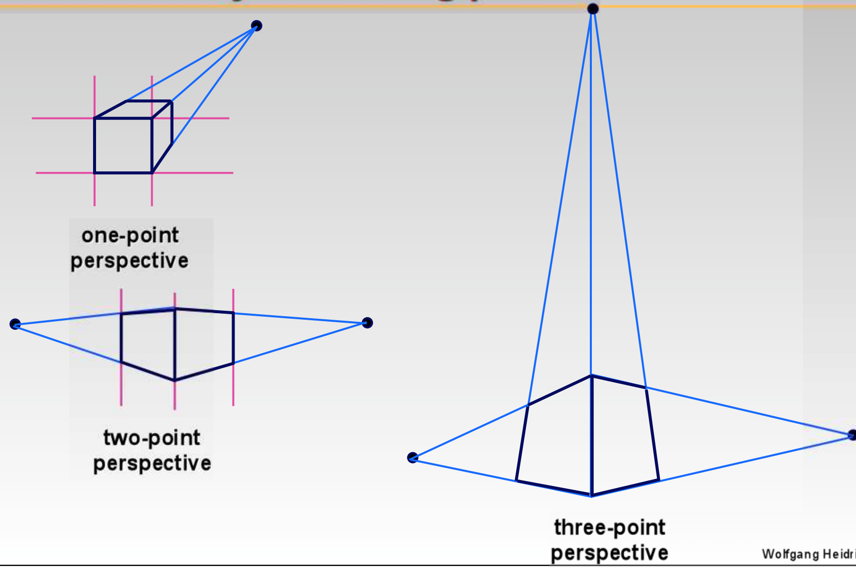


http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Wolfgang Heidrich

Perspective Projections

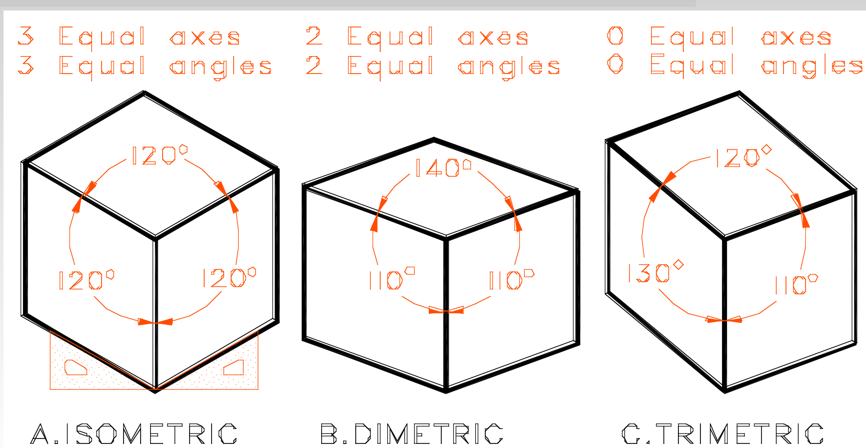
- *classified by vanishing points*



Wolfgang Heidrich

Axonometric Projections

- projectors perpendicular to image plane

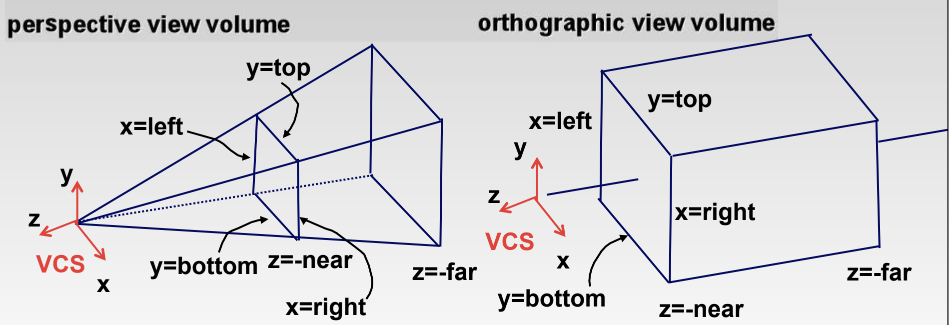


http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Wolfgang Heidrich

View Volumes

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



View Volume

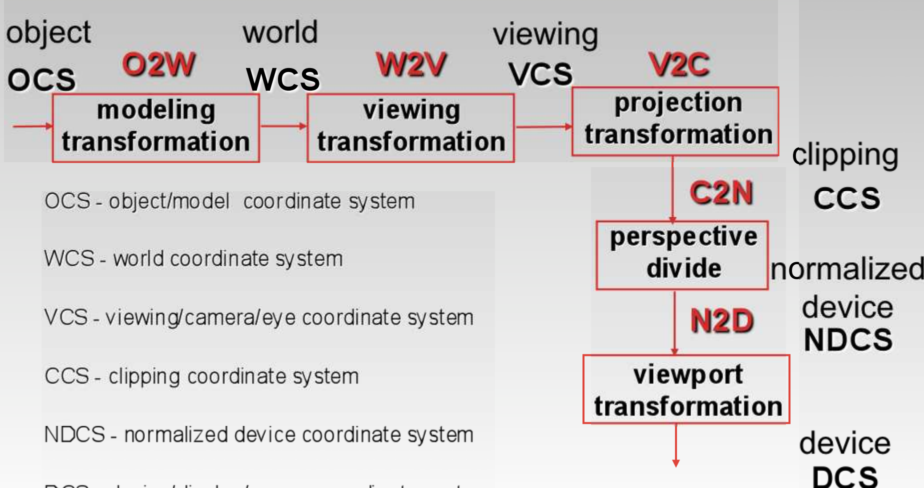
- ***Convention***
  - Viewing frustum mapped to specific parallelepiped
    - *Normalized Device Coordinates (NDC)*
    - *Same as clipping coords*
  - Only objects inside the parallelepiped get rendered
  - Which parallelepiped?
    - *Depends on rendering system*

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*

Wolfgang Heidrich

Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

CCS - clipping coordinate system

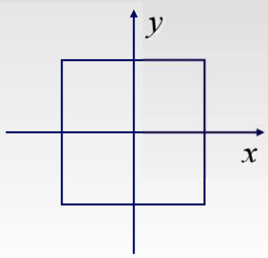
NDCS - normalized device coordinate system

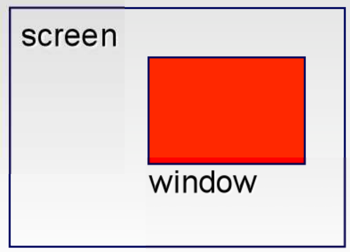
DCS - device/display/screen coordinate system

Wolfgang Heidrich

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





Wolfgang Heidrich

Coming Up:

Wednesday:

- More on perspective projection

Friday/Next Week

- Lighting/shading

Wolfgang Heidrich