CSE167: Introduction to Computer Graphics

Lecture #4

Jürgen Schulze
University of California, San Diego
Fall 2008
Project 1

- Due this Friday (Oct 10)
- Present during lab session
  - Friday 1:00p - 4:00p
  - EBU3B 250, look in different lab (240 first) if you can’t find Robert or me
  - List your name on the whiteboard once you get to the lab. Homework will be graded in this order.
- New: instructions for Linux lab machines on web site

In addition

Before your demonstration in the lab:
Zip/tar up your code (only .cpp and .h files needed) and email to jschulze AT ucsd.edu
Questions?

• Problems with base code?
Today

- Review
- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation
Review

- Change of coordinates
- Object, world, camera coordinates
- Camera matrix
Change of coordinates

\[ \mathbf{p}_{\text{xyz}} = p_x \mathbf{x} + p_y \mathbf{y} + p_z \mathbf{z} + \mathbf{o} \]
Change of coordinates

\[ p_{xyz} = p_x x + p_y y + p_z z + o \]

Coordinates of \( p \) w.r.t. \( uvwq \) frame?
Change of coordinates

\[ p_{xyz} = p_x \mathbf{x} + p_y \mathbf{y} + p_z \mathbf{z} + \mathbf{o} \]

Coordinates of \( \mathbf{xyzo} \) frame w.r.t. \( \mathbf{uvwq} \) frame

\[
\mathbf{x} = \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix} \quad \mathbf{o} = \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}
\]
Change of coordinates

\[ p_{xyz} = p_x x + p_y y + p_z z + o \]

Point \( p_{xyz} \) w.r.t. \( uvwq \) frame

\[
\begin{bmatrix}
    p_x \\
    p_y \\
    p_z \\
    1
\end{bmatrix}
= \begin{bmatrix}
    x_u & y_u & z_u & o_u \\
    x_v & y_v & z_v & o_v \\
    x_w & y_w & z_w & o_w \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    p_x \\
    p_y \\
    p_z \\
    1
\end{bmatrix}
\]
Change of coordinates

- Given coordinates of basis $x, y, z, o$ with respect to new frame $u, v, w, q$

\[
\begin{align*}
x &= \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix}, \quad y &= \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix}, \quad z &= \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix}, \quad o &= \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}
\end{align*}
\]

- Coordinates of point $p_{xyz}$ w.r.t. new frame

\[
p_{uvw} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ o \end{bmatrix}
\]

- Columns of transformation matrix are new coordinates ($uvw$) of old basis vectors ($xyzo$)
Change of coordinates

Inverse transformation

• Given point $p_{uvw}$ w.r.t. frame $u, v, w, q$
• Coordinates $p_{xyz}$ w.r.t. frame $x, y, z, o$

$$
\begin{bmatrix}
  x_u & y_u & z_u & o_u \\
  x_v & y_v & z_v & o_v \\
  x_w & y_w & z_w & o_w \\
  0 & 0 & 0 & 1
\end{bmatrix}^{-1}
\begin{bmatrix}
  p_u \\
  p_v \\
  p_w \\
  1
\end{bmatrix}
$$
Object, world, camera coords.
Object, world, camera coords.

Object-to-world transformation matrix $M$
- Columns contain basis vectors of object space in world coordinates

Camera-to-world transformation matrix $C$
- Columns contain basis vectors of camera space in world coordinates

Object-to-camera

$$\mathbf{p}_{\text{camera}} = C^{-1} M \mathbf{p}_{\text{object}}$$
Camera-to-world matrix

“Camera matrix”

- Construct from center of projection $e$, look at $d$, up-vector

![Diagram of camera and world coordinate systems](image-url)
Camera matrix

• z-axis

\[ z_c = \frac{e - d}{\|e - d\|} \]

• x-axis

\[ x_c = \frac{\text{up} \times z_c}{\|\text{up} \times z_c\|} \]

• y-axis

\[ y_c = z_c \times x_c \]

• Camera matrix

\[
C = \begin{bmatrix}
x_c & y_c & z_c & e \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Objects in camera coordinates

• We have things lined up the way we like them on screen
  – $x$ to the right
  – $y$ up
  – $z$ going into the screen
  - Objects to look at are in front of us, i.e. have negative $z$ values

• But objects are still in 3D

• Today: how to project them into 2D
Questions?
Today

- Review
- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation
Rendering pipeline

- Hardware & software that draws 3D scenes on the screen
- Consists of several stages
  - Simplified version here
- Most operations performed by specialized hardware (GPU)
- Access to hardware through low-level 3D API (OpenGL, DirectX)
- All scene data flows through the pipeline at least once for each frame
Rendering engine

- Additional software layer encapsulating low-level API
- More functionality
- Platform independent
- Layered software architecture common in industry
  - Game engines
Rendering pipeline

Scene data

Modeling and viewing transformation

- Textures, lights, etc.
- Geometry
  - Vertices and how they are connected
  - Triangles, lines, points, triangle strips
  - Attributes such as color

Shading

Projection

Rasterization, visibility

- Specified in object coordinates
- Processed by the rendering pipeline one-by-one

Image
Rendering pipeline

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

- Transform object to camera coordinates
- Specified by GL_MODELVIEW matrix in OpenGL
- User computes GL_MODELVIEW matrix as discussed

\[ \mathbf{p}_{\text{camera}} = \mathbf{C}^{-1} \mathbf{M} \mathbf{p}_{\text{object}} \]
Rendering pipeline

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

- Look up light sources
- Compute color for each vertex
- Later in the course
Rendering pipeline

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

- Project 3D vertices to 2D image positions
- GL_PROJECTION matrix
- This lecture
Rendering pipeline

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

- Draw primitives (triangles, lines, etc.)
- Determine what is visible
- Next lecture
Rendering pipeline

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

- Pixel colors
Questions?
Today

- Review
- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation
Projections

- Given 3D points (vertices) in camera coordinates, determine corresponding image coordinates

Orthographic projection (=parallel projection)

- Simply ignore $z$-coordinate
- Use camera space $xy$ coordinates as image coordinates
Orthographic projection

- Project points to $x$-$y$ plane along parallel lines
- Graphical illustrations, architecture
Perspective projection

• Most common for computer graphics

• Simplified model of human eye, or camera lens (pinhole camera)

• Things farther away seem smaller

• Discovery/formalization attributed to Filippo Brunelleschi (Italian architect) in the early 1400’s
Perspective projection

- Project along rays that converge in center of projection
Perspective projection

Parallel lines no longer parallel, converge at one point

Earliest example
La Trinitá (1427) by Masaccio
Perspective projection

The math: simplified case

\[
\frac{y'}{d} = \frac{y_1}{z_1}
\]

\[
y' = \frac{y_1 d}{z_1}
\]

\[
x' = \frac{x_1 d}{z_1}
\]

\[
z' = d
\]
Perspective projection

The math: simplified case

\[ x' = \frac{x_1 d}{z_1} \]

\[ y' = \frac{y_1 d}{z_1} \]

\[ z' = d \]

- Can express this using homogeneous coordinates, 4x4 matrices
Perspective projection

The math: simplified case

\[ x' = \frac{x_1 d}{z_1} \]
\[ y' = \frac{y_1 d}{z_1} \]
\[ z' = d \]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0 \\
0 & 0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z \\
zd \\
1 \\
\end{bmatrix}
\]

Projection matrix

Homogeneous division
Perspective projection

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\begin{bmatrix}
x \\y \\z \\1
\end{bmatrix}
= 
\begin{bmatrix}
x \\y \\z \\z/d
\end{bmatrix}
\mapsto
\begin{bmatrix}
xd/z \\yd/z \\d \\1
\end{bmatrix}
\]

Projection matrix  Homogeneous division

- Using projection matrix, homogeneous division seems more complicated than just multiplying all coordinates by \( d/z \), so why do it?
- Will allow us to
  - handle different types of projections in a unified way
  - define arbitrary view volumes
Perspective projection

- The math behind the general case can be complicated
  - Projective transformations, 3D projective space

In practice

- We normally use homogeneous matrices
- **Modeling & viewing transformations use affine matrices**
  - for points $w$ is always 1
  - no need to divide by $w$ when doing modeling operations or transforming into camera space

- **Projection transform uses perspective matrices**
  - $w$ not always 1

- Divide by $w$ (perspective division, homogeneous division) after performing projection transform
  - Graphics hardware does this
Realistic image generation

More than just perspective projection

- Example: lens effects

Focus, depth of field  Fish-eye lens
Realistic image generation

- Chromatic aberration
- Motion blur

- Often too complicated for hardware rendering pipeline
Today

- Review
- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation
View volumes

- Define 3D volume seen by camera

Perspective view volume
Camera coordinates

Orthographic view volume
Camera coordinates

World coordinates
Perspective view volume

General view volume

- Defined by 6 parameters, in camera coordinates
  - Left, right, top, bottom boundaries
  - Near, far clipping planes
- Clipping planes to avoid numerical problems
  - Divide by zero
  - Low precision for distant objects
- Usually symmetric, i.e., left=-right, top=-bottom
Perspective view volume

Symmetric view volume

• Only 4 parameters
  - Vertical field of view (FOV)
  - Image aspect ratio (width/height)
  - Near, far clipping planes

\[
\text{aspect ratio} = \frac{\text{right} - \text{left}}{\text{top} - \text{bottom}} = \frac{\text{right}}{\text{top}}
\]

\[
\tan(\frac{\text{FOV}}{2}) = \frac{\text{top}}{\text{near}}
\]
Orthographic view volume

- Parametrized by 6 parameters
  - Right, left, top, bottom, near, far
- If symmetric
  - Width, height, near, far
Clipping

- Need to identify objects outside view volume
  - Avoid division by zero
  - Efficiency, don’t draw objects outside view volume (view frustum culling)
- Performed by hardware

- Hardware always clips to the canonical view volume
- Cube $[-1..1] \times [-1..1] \times [-1..1]$ centered at origin
- Need to transform desired view frustum to canonical view frustum
Clipping

Primitives

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

Clip to view frustum
Canonic view volume

• Projection matrix is set such that
  - User defined view volume is transformed into canonical view volume, i.e., cube $[-1,1] \times [-1,1] \times [-1,1]$
  - Multiplying vertices of view volume by projection matrix and performing homogeneous divide yields canonical view volume
• Perspective and orthographic projection are treated exactly the same way
Projection matrix

Camera coordinates

Projection matrix

Canonical view volume

Clipping
Perspective projection matrix

- General view frustum

\[ P_{persp}(left, right, top, bottom, near, far) = \]

\[
\begin{bmatrix}
\frac{2\text{near}}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\
0 & \frac{2\text{near}}{top-bottom} & \frac{top-bottom}{top-bottom} & 0 \\
0 & 0 & \frac{-(far+near)}{far-near} & -1 \\
0 & 0 & \frac{-2far\cdot near}{far-near} & 0
\end{bmatrix}
\]
Perspective projection matrix

- Symmetric view frustum with field of view, aspect ratio, near and far clip planes

$$\mathbf{P}_{persp}(FOV, \text{aspect}, \text{near}, \text{far}) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
\frac{1}{\text{aspect} \cdot \tan(FOV/2)} & 0 & 0 & 0 \\
0 & \frac{1}{\tan(FOV/2)} & 0 & 0 \\
0 & 0 & \frac{\text{near} + \text{far}}{\text{near} - \text{far}} & \frac{2 \cdot \text{near} \cdot \text{far}}{\text{near} - \text{far}} \\
0 & 0 & \frac{\text{near} - \text{far}}{\text{near} - \text{far}} & -1 & 0
\end{bmatrix}$$
Orthographic projection matrix

\[
P_{\text{ortho}}(\text{right, left, top, bottom, near, far}) = \begin{bmatrix}
\frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
0 & \frac{2}{\text{top} - \text{bottom}} & 0 & -\frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\
0 & 0 & \frac{2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
P_{\text{ortho}}(\text{width, height, near, far}) = \begin{bmatrix}
\frac{2}{\text{width}} & 0 & 0 & 0 \\
0 & \frac{2}{\text{height}} & 0 & 0 \\
0 & 0 & \frac{2}{\text{far} - \text{near}} & \frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Questions?
Today

- Review
- Rendering pipeline
- Projections
- View volumes
- Viewport transformation
Viewport transformation

- After applying projection matrix, image points are in *normalized view coordinates*
  - Per definition range \([-1..1] \times [-1..1]\]
- Normalized view coordinates can be mapped to image (i.e., pixel) coordinates
  - User defined range \([x_0...x_1] \times [y_0...y_1]\]
- Scale and translation required:

\[
D(x_0,x_1,y_0,y_1) = \begin{bmatrix}
(x_1 - x_0)/2 & 0 & 0 & (x_0 + x_1)/2 \\
0 & (y_1 - y_0)/2 & 0 & (y_0 + y_1)/2 \\
0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates

- Object-to-world matrix $M$, camera matrix $C$, projection matrix $P$, viewport matrix $D$

$$p' = DPC^{-1}Mp$$

Object space
The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $M$, camera matrix $C$, projection matrix $P$, viewport matrix $D$

$$p' = DPC^{-1}M_p$$

Object space

World space
The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $M$, camera matrix $C$, projection matrix $P$, viewport matrix $D$

$$ p' = DPC^{-1}Mp $$

Object space

World space

Camera space
The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates

- Object-to-world matrix \( M \), camera matrix \( C \), projection matrix \( P \), viewport matrix \( D \)

\[
p' = DPC^{-1}Mp
\]

- Object space
- World space
- Camera space
- Canonical view volume
The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates

- Object-to-world matrix $M$, camera matrix $C$, projection matrix $P$, viewport matrix $D$

$$p' = DPC^{-1}M_p$$

Object space

World space

Camera space

Canonical view volume

Image space
The complete transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix $M$, camera matrix $C$, projection matrix $P$, viewport matrix $D$

$$p' = DPC^{-1}Mp$$

$$p' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

Pixel coordinates: $x'/w'$, $y'/w'$
• Object-to-world matrix $M$, camera matrix $C$, projection matrix $P$, viewport matrix $D$
OpenGL

- **GL_MODELVIEW, \( C^{-1}M \)**
  - Up to you to define

- **GL_PROJECTION, \( P \)**
  - Utility routines to set it by specifying view volume: `glFrustum()`, `glPerspective()`, `glOrtho()`
  - Do not use utility functions for project 2
  - You will implement a software renderer in project 3, which will not rely on any OpenGL

- **Viewport, \( D \)**
  - Specify implicitly via `glViewport()`
  - No direct access with equivalent to GL_MODELVIEW or GL_PROJECTION
Next time

- Drawing (rasterization)
- Visibility (z-buffering)