CSE167: Introduction to Computer Graphics

Lecture #4

Jürgen Schulze
University of California, San Diego
Fall 2009
Announcements

- Visual C++ and GLUT are being installed on all Windows machines -> suggested to use Linux for this project
- Freeglut should be installed on all Linux machines. Previous GLUT version will remain in place also.
- TA lab sessions this week:
  - Daniel: Tue+Thu 11am-2pm
  - Jason: Mon 2-4pm, Wed 2-5pm
Homework Grading

- Homework project due this Friday (Oct 9)
- Can be presented to TAs or instructor during any lab session this week
- Latest presentation opportunity before late penalty: Friday Oct 9, 1:00p - 5:00p
- EBU3B 250, look in other labs if you can’t find Daniel, Jason and me
- List your name on the whiteboard in the grading section once you get to the lab. Homework will be graded in this order.
- We will also have a help section on the whiteboard. List your name there to get help. We will give priority to the grading list!

In addition to grading:

- Zip/tar up your source code (only .cpp and .h files required) and email to jschulze@ucsd.edu by 5pm!
- If you don’t send your source code in by the end of the grading slot (5pm) we may not be able to change your grade in case of a later dispute.
Today

- Common coordinate systems
- Rendering pipeline
- Projections
- View volumes, clipping
Common coordinate systems

• Camera, world, and object coordinates
Object coordinates

- Coordinates the object is defined with
- Often origin is in middle, base, or corner of object
- No right answer, whatever was convenient for the creator of the object
World coordinates

- “World space”
- Common reference frame for all objects in the scene
- Chosen for convenience, no right answer
  - If there is a ground plane, usually x-y is horizontal and z points up
World coordinates

- Transformation from object to world space is different for each object
- Defines placement of object in scene
- Given by “model matrix” (model-to-world transform) $M$
Camera coordinate system

- "Camera space"
- Origin defines center of projection of camera
- x-y plane is parallel to image plane
- z-axis is perpendicular to image plane
Camera coordinate system

• “Camera matrix” defines transformation from camera to world coordinates
  - Placement of camera in world
• Transformation from object to camera coordinates

\[ p_{\text{camera}} = C^{-1}Mp_{\text{object}} \]
Camera matrix

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

![Diagram showing camera coordinates and world coordinates with vectors $e$, $up$, and $d$.]
**Camera matrix**

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

![Diagram of camera and world coordinates]

- Camera coordinates
- World coordinates
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

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

- Think about: What does it mean to compute

\[
p' = Cp
\]

\[
q' = C^{-1}q
\]
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

• Problem: project them into 2D
Questions?
Today

- Common coordinate systems
- Rendering pipeline
- Projections
- View volumes, clipping
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
Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

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

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

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

MODEL VIEW matrix
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
CSE 167 Rendering engine

- Additional software layer encapsulating low-level API
- Higher level functionality than OpenGL
- Platform independent
- Layered software architecture common in industry
  - Game engines
Questions?
Today

• Common coordinate systems
• Rendering pipeline
• Projections
• View volumes, clipping
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 look smaller
- Discovery 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

\[
\begin{align*}
y' &= \frac{y_1}{d} \\
y' &= \frac{y_1 d}{z_1} \\
x' &= \frac{x_1 d}{z_1} \\
z' &= d
\end{align*}
\]
Perspective projection

The math: simplified case

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

- We can express this using homogeneous coordinates and 4x4 matrices!
Perspective projection

The math: simplified case

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

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

\[ z' = d \]

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

Projection matrix

Homogeneous division
Using projection matrix, homogeneous division seems more complicated than just multiplying all coordinates by \( d/z \), so why do it?

- It will allow us to
  - handle different types of projections in a unified way
  - define arbitrary view volumes
- 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

• Common coordinate systems
• Rendering pipeline
• Projections
• View volumes, clipping
View volumes

- Define 3D volume seen by camera

Perspective view volume
- Camera coordinates
- World 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

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

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

• Projection matrix is set such that
  - User defined view volume is transformed into canonical view volume, i.e., cube [-1,1]x[-1,1]x[-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

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

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

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

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

Camera coordinates

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

\[
\begin{bmatrix}
\frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\
0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\
0 & 0 & \frac{2}{far - near} & -\frac{far + near}{far - near} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[ P_{ortho}(width, height, near, far) = \]

\[
\begin{bmatrix}
\frac{2}{width} & 0 & 0 & 0 \\
0 & \frac{2}{height} & 0 & 0 \\
0 & 0 & \frac{2}{far - near} & \frac{far + near}{far - near} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Questions?
Next Lecture

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