Today

- Review & Schedule
- Ray Casting / Tracing vs. Scan Conversion
- The Graphics Pipeline
- Projective Transformations

Last Week:
- Animation & Quaternions
- Finite Element Simulations
  - collisions, fracture, & deformation

Schedule

- Final Project
  - Post your ideas on the web page
  - Meet with staff to talk about project ideas
    - sign up for an appointment on Friday
  - Proposal due on Monday October 27th
- Friday October 24th: Assignment 5 due
- Office Hours this week:
  - Tuesday after class (Rob – student center)
  - Wednesday 7-9 (Patrick – student center)
  - Thursday after class (Fredo – student center)
  - Friday 3-5, student center (Barb – student center)

XForms Forms Library

- GUI (graphical user interface) for Linux
- buttons, scrollbars, dialog boxes, menus, etc.
- fdesign for interactive layout

Questions?
Today

- Review & Schedule
- Ray Casting / Tracing vs. Scan Conversion
- The Graphics Pipeline
- Projective Transformations

What have we done so far?

- Ray Casting / Tracing
  - ray/primitive intersections
  - transformations
  - local shading (diffuse, ambient, \rightarrow BRDFs)
  - global effects (shadows, transparency, caustics, ...)

Ray Casting / Tracing

for every pixel, construct a ray from the eye
for every object in the scene
intersect ray with object
find closest intersection with the ray
compute normal at point of intersection
compute color for pixel (shoot secondary rays)

"Inverse-Mapping" approach

Can we render things interactively?

- Of course! games, 3D modeling packages, architectural walkthroughs, assignment 5, etc.

How do we render interactively?

- Use the graphics hardware (the graphics pipeline), via OpenGL, MesaGL, or DirectX
- Most global effects available in ray tracing will be sacrificed, but some can be approximated.
Scan Conversion – Graphics Pipeline

- Primitives are processed one at a time
- Early stages involve analytic processing
- Sampling occurs late in the pipeline
- Minimal state required

```c
glBegin(GL_TRIANGLES)
glNormal3f(...)
glVertex3f(...)
glVertex3f(...)
glVertex3f(...)
glEnd();
```

Scan Conversion

- Given the primitive's vertices & the illumination at each vertex:
- Figure out which pixels to "turn on" to render the primitive
- Interpolate the illumination values to "fill in" the primitive
- At each pixel, keep track of the closest primitive (z-buffer)

Limitations of Scan Conversion

- Restricted to scan-convertible primitives
  - Object polygonization
- Faceting, shading artifacts
- Effective resolution is hardware dependent
- No handling of shadows, reflection, transparency
- Problem of overdraw (high depth complexity)
- What if there are more triangles than pixels?

Questions?

Today

- Review & Schedule
- Ray Casting / Tracing vs. Scan Conversion
- The Graphics Pipeline
- Projective Transformations
The Graphics Pipeline

- **Input:**
  - Geometric model: Description of all object, surface, and light source geometry and transformations
  - Lighting model: Computational description of object and light properties, interaction (reflection)
  - Eye position and viewing frustum
  - Host Viewport: Pixel grid onto which image plane is mapped

- **Output:**
  - Colors/Intensities suitable for framebuffer display (For example: 24-bit RGB value at each pixel)

Modeling Transformations

- 3D models defined in their own coordinate system (object space)
- Modeling transforms orient the models within a common coordinate frame (world space)

Illumination (Shading) (Lighting)

- Vertices lit (shaded) according to material properties, surface properties (normal) and light sources
- Local lighting model (Diffuse, Ambient, Phong, etc.)

Viewing Transformation

- Maps world space to eye space
- Viewing position is transformed to origin & direction is oriented along some axis (usually z)

Clipping

- Transform to Normalized Device Coordinates (NDC)
- Portions of the object outside the view volume (view frustum) are removed

- Primitives are processed in a series of stages
- Each stage forwards its result on to the next stage
- The pipeline can be drawn and implemented in different ways
- Some stages may be in hardware, others in software
- Optimizations & additional programmability are available at some stages
**Projection**
- The objects are projected to the 2D image place (screen space)

**Scan Conversion (Rasterization)**
- Rasterizes objects into pixels
- Interpolate values as we go (color, depth, etc.)

**Visibility / Display**
- Each pixel remembers the closest object (depth buffer)
- Almost every step in the graphics pipeline involves a change of coordinate system. Transformations are central to understanding 3D computer graphics.

**Common Coordinate Systems**
- Object space – local to each object
- World space – common to all objects
- Eye space / Camera space – derived from view frustum
- Clip space / Normalized Device Coordinates (NDC) – $[-1,-1,-1] \rightarrow [1,1,1]$
- Screen space – indexed according to hardware attributes

**Coordinate Systems in the Pipeline**
- Object space
- World space
- Eye Space / Camera Space
- Clip Space (NDC)
- Screen Space
Today

- Review & Schedule
- Ray Casting / Tracing vs. Scan Conversion
- The Graphics Pipeline
- Projective Transformations
  - Transformations & Homogeneous Coordinates
  - Orthographic & Perspective Projections
  - Canonical View Volume

Remember Transformations?

Homogeneous Coordinates

- Most of the time $w = 1$, and we can ignore it
- If we multiply a homogeneous coordinate by an affine matrix, $w$ is unchanged

Homogeneous Visualization

- Divide by $w$ to normalize (homogenize)
- $W = 0$? Point at infinity (direction)

Orthographic vs. Perspective

- Orthographic

- Perspective

Simple Orthographic Projection

- Project all points along the $z$ axis to the $z = 0$ plane
Simple Perspective Projection

- Project all points along the z axis to the z = d plane, eyepoint at the origin:

\[
\begin{align*}
    x_p &= \frac{d \cdot x}{z} \\
    y_p &= \frac{d \cdot y}{z} \\
    z_p &= d
\end{align*}
\]

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

Alternate Perspective Projection

- Project all points along the z axis to the z = 0 plane, eyepoint at the (0,0,-d):

\[
\begin{align*}
    x_p &= \frac{(z + d) \cdot x}{d} \\
    y_p &= \frac{(z + d) \cdot y}{d} \\
    z_p &= 0
\end{align*}
\]

\[
\begin{pmatrix}
    x \cdot d / (z + d) \\
    y \cdot d / (z + d) \\
    0
\end{pmatrix}
= \begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 1/d & 1
\end{pmatrix}
\]

In the limit, as \( d \to \infty \)

...is simply an orthographic projection

\[
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 1/d & 1
\end{pmatrix}
\]

What if the \( p_z \) is \( \leq \text{eye}_z \)?

\[
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 1/d & 1
\end{pmatrix}
\]

What if the \( p_z \) is \( \leq \text{eye}_z \)?

\[
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 \\
    0 & 0 & 1/d & 1
\end{pmatrix}
\]

MIT EECS 6.837, Durand and Cutler
What if the $p_z$ is $\leq eyez$?

"clip" geometry to view frustum

Where are projections in the pipeline?

Modeling Transformations
Illumination (Shading)
Viewing Transformation: Perspective / Orthographic
Clipping
Projection: (to Screen Space)
Scan Conversion (Rasterization)
Visibility / Display

World Space $\rightarrow$ Eye Space

Positioning the camera
Translation + Change of orthonormal basis (Lecture 4)

- Given: coordinate frames $xyz$ & $uvn$, and point $p = (x,y,z)$
- Find: $p = (u,v,n)$

Change of Orthonormal Basis

$\begin{bmatrix} u \\ v \\ n \end{bmatrix} = \begin{bmatrix} ux & uy & uz \\ vx & vy & vz \\ nx & ny & nz \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

where:
- $ux = x \cdot u$
- $uy = y \cdot u$
- etc.

Normalized Device Coordinates

- Clipping is more efficient in a rectangular, axis-aligned volume: $(-1,-1,-1) \rightarrow (1,1,1)$ OR $(0,0,0) \rightarrow (1,1,1)$

Eye Space

Normalized Device Coordinates

(left-handed; $z$ moves into display)

(left-handed; $z$ moves into display)
Canonical Orthographic Projection

Orthographic

Canonical Perspective Projection

Perspective

Questions?

Next Time:
Line Rasterization

MIT EECS 6.837, Durand and Cutler
MIT EECS 6.837, Durand and Cutler