Graphics Pipeline: Projective Transformations

Last Time

- Shadows
  - cast ray to light
  - stop after first intersection
- Reflection & Refraction
  - compute direction of recursive ray
- Recursive Ray Tracing
  - maximum number of bounces OR
  - contribution < error threshold
- Epsilon…

Does Ray Tracing Simulate Physics?

- Ray Tracing is full of dirty tricks
- For example, shadows of transparent objects:
  - opaque?
  - multiply by transparency color?
    (ignores refraction & does not produce caustics)

Correct Transparent Shadow

---

Refraction and the Lifeguard Problem

- Running is faster than swimming

Does Ray Tracing Simulate Physics?

- Photons go from the light to the eye, not the other way
- What we do is *backward ray tracing*
### Forward Ray Tracing
- Start from the light source
  - But low probability to reach the eye
- What can we do about it?
  - Always send a ray to the eye…. still not efficient

### The Rendering Equation
- Clean mathematical framework for light-transport simulation
- At each point, outgoing light in one direction is the integral of incoming light in all directions multiplied by reflectance property
  - *We’ll see this later…*

### Questions?

### Today
- Ray Casting / Tracing vs. Scan Conversion
  - advantages & disadvantages
  - when is each appropriate?
- The Graphics Pipeline
- Projective Transformations
- Introduction to Clipping

### Ray Casting / Tracing
- Advantages?
  - Smooth variation of normal, silhouettes
  - Generality: can render anything that can be intersected with a ray
  - Atomic operation, allows recursion
- Disadvantages?
  - Time complexity (N objects, R pixels)
  - Usually too slow for interactive applications
  - Hard to implement in hardware (lacks computation coherence, must fit entire scene in memory)

### How Do We Render Interactively?
- Use 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

- Given a 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)

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

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 many more triangles than pixels?

Ray Casting vs. Rendering Pipeline

<table>
<thead>
<tr>
<th>Ray Casting</th>
<th>Rendering Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each pixel</td>
<td>For each triangle</td>
</tr>
<tr>
<td>For each object</td>
<td>For each pixel</td>
</tr>
<tr>
<td>Send pixels to the scene</td>
<td>Project scene to the pixels</td>
</tr>
<tr>
<td>Discretize first</td>
<td>Discretize last</td>
</tr>
</tbody>
</table>

Ray Casting

- Whole scene must be in memory
- Depth complexity: no computation for hidden parts
- Atomic computation
- More general, more flexible
  - Primitives, lighting effects, adaptive antialiasing

Rendering Pipeline

- Primitives processed one at a time
- Coherence: geometric transforms for vertices only
- Early stages involve analytic processing
- Computation increases with depth of the pipeline
  - Good bandwidth/computation ratio
- Sampling occurs late in the pipeline
- Minimal state required

Movies

both pipeline and ray tracing

Games

pipeline
Simulation pipeline (painter for a long time)

CAD-CAM & Design pipeline during design, anything for final image

Architecture ray-tracing, pipeline with preprocessing for complex lighting

Virtual Reality pipeline

Visualization mostly pipeline, ray-tracing for high-quality eye candy, interactive ray-tracing is starting

Medical Imaging same as visualization
Questions?

Today

- Ray Casting / Tracing vs. Scan Conversion
- The Graphics Pipeline
- Projective Transformations
- Introduction to Clipping

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)
- Synthetic viewpoint or Camera:
  - Eye position and viewing frustum
  - Camera 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

Projection

- The objects are projected to the 2D image place (screen space)

Scan Conversion (Rasterization)

- Rasterizes objects into pixels
- Interpolate objects 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

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

Questions?

Today
- Ray Casting / Tracing vs. Scan Conversion
- The Graphics Pipeline
- Projective Transformations
  - Transformations & Homogeneous Coordinates
  - Orthographic & Perspective Projections
  - Coordinate Systems & Projections in the Pipeline
  - Canonical View Volume
- Introduction to Clipping

Remember Transformations?

Homogeneous Coordinates
- Most of the time $w = 1$, and we can ignore it

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- If we multiply a homogeneous coordinate by an affine matrix, $w$ is unchanged

Homogeneous Visualization
- Most of the time $w = 1$, and we can ignore it

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- If we multiply a homogeneous coordinate by an affine matrix, $w$ is unchanged

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$
Orthographic vs. Perspective

- Orthographic

- Perspective

Simple Orthographic Projection

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

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

Simple Perspective Projection

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

$$
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
= \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1/d & 0 \\
  0 & 0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
$$

Alternate Perspective Projection

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

$$
\begin{bmatrix}
  x \\
  y \\
  0
\end{bmatrix}
= \begin{bmatrix}
  x \\
  y \\
  (z + d)/d
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1/d & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
$$

In the limit, as $d \to \infty$

the perspective projection matrix...

...is simply an orthogonal projection

$$
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1/d & 1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 1
\end{bmatrix}
$$

Where are projections in the pipeline?

1. Modeling
2. Transformations
3. Illumination (Shading)
4. Viewing Transformation (Perspective / Orthographic)
5. Clipping
6. Projection (to Screen Space)
7. Scan Conversion (Rasterization)
8. Visibility / Display

Eye Space / Camera Space
Clip Space (NDC)
Screen Space
Positioning the camera
Translation + Change of orthonormal basis

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

\[
\begin{align*}
\begin{bmatrix}
U \\
V \\
N \\
\end{bmatrix} &= 
\begin{bmatrix}
U_x & U_y & U_z \\
V_x & V_y & V_z \\
N_x & N_y & N_z \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
\end{bmatrix} \\
\end{align*}
\]

where:
\(U_x = x \cdot u\)
\(U_y = y \cdot u\)

etc.

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

Canonical Orthographic Projection

Orthographic
\[
\begin{bmatrix}
x' & 0 & 0 & 0 \\
y' & 0 & 0 & 0 \\
z' & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
(right - left) & 0 & 0 \\
(bottom - top) & 0 & 0 \\
(far - near) & 0 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Questions?
Today

• Ray Casting / Tracing vs. Scan Conversion
• The Graphics Pipeline
• Projective Transformations
• Introduction to Clipping
  – Projecting to the Image Plane
  – Why Clip?
  – Clipping Strategies

What if the $p_z$ is $> e_{ey_z}$?

What if the $p_z$ is $< e_{ey_z}$?

What if the $p_z = e_{ey_z}$?

Clipping

• Eliminate portions of objects outside the viewing frustum
• View Frustum
  – boundaries of the image plane projected in 3D
  – a near & far clipping plane
• User may define additional clipping planes
Why Clip?

- Avoid degeneracies
  - Don’t draw stuff behind the eye
  - Avoid division by 0 and overflow
- Efficiency
  - Don’t waste time on objects outside the image boundary
- Other graphics applications (often non-convex)
  - Hidden surface removal, Shadows, Picking, Binning, CSG (Boolean) operations (2D & 3D)

Clipping Strategies

- Don’t clip (and hope for the best)
- Clip on-the-fly during rasterization
- Analytical clipping: alter input geometry

Next Time:
Clipping & Line Rasterization