Review of Rendering

OUTLINE:

Visibility:
painter’s, z-buffer, ray casting

Optics:
light, color, reflection, shadows, transparency, fog

Shading:
Gouraud/Phong shading, ray tracing (spatial data structures, antialiasing, generalizations), texture mapping, radiosity

Volume Rendering

Practical Comparison of Visibility Algorithms

<table>
<thead>
<tr>
<th>algorithm</th>
<th>mem.</th>
<th>shadows</th>
<th>analytic</th>
<th>transparency</th>
<th>adv. and disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Painter’s</td>
<td>image</td>
<td>N</td>
<td>partial</td>
<td>Y</td>
<td>ADV: easy implementation if no sorting required. DIS: sorting &amp; intersecting polygons tricky.</td>
</tr>
<tr>
<td>z-buffer</td>
<td>image &amp; z-buf</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>ADV: easy impl., general, draw in arb. order, good for complex scenes, parallelizable.</td>
</tr>
<tr>
<td>scanline</td>
<td>scene &amp; active edge list</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>ADV: low mem. req. if simple scene. DIS: requires sorting, polys only, slow for complex scene.</td>
</tr>
<tr>
<td>ray casting</td>
<td>scene &amp; spatial datastruc</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>ADV: very general. DIS: slow without spatial data structures.</td>
</tr>
</tbody>
</table>
Theoretical Comparison of Rendering Algorithms

\[ s = \#\text{surfaces (e.g. polygons)} \quad t_s = \text{time per surface (transforming, ...)} \]
\[ p = \#\text{pixels} \quad t_p = \text{time per pixel (writing, incrementing, ...)} \]
\[ \ell = \#\text{lights} \quad t_\ell = \text{time to light surface point w.r.t. one light} \]
\[ a = \sum \text{screen areas of surfs} \quad t_a = \text{time for one ray/surface intersection test} \]

**Painter’s or Z-buffer algorithm, with flat shading**

*(assuming no sorting in painter’s algorithm)*

worst case cost = \( s(t_s + \ell t_\ell) + a t_p \) = \( a t_p \) if polygons big

**Painter’s or Z-buffer algorithm, with per pixel shading (e.g. Phong)**

worst case cost = \( st_i + a(\ell t_\ell + t_p) \) = \( a t_\ell \) if polygons big

**Ray casting with no shadows, no spatial data structures**

worst case cost = \( p st_i + \ell t_\ell + t_p \)

**Ray tracing to max depth \( d \) with shadows, refl\&tran, no spat. DS, no supersampling**

\[ 2^{d-1} \] intersections/pixel, for each of which there are \( \ell \) shadow rays

worst case cost = \( p(2^{d-1})[(\ell + 1)st_i + \ell t_\ell] \) = \( 2^d p st_i \) if many surfaces

*Note: time constants vary, e.g. \( t_p \) is larger for z-buffer than for painter’s.*

Optics 1

Light is electromagnetic radiation visible to humans.

Light intensity is a function of position, direction, wavelength, and time.
Radiance = energy/(time \cdot \text{area} \cdot \text{solid angle})

Color is humans’ perception of light.

3-D because we have 3 sets of cones in our eyes.
Hence 3 primary colors suffice (e.g. R,G,B)
Light is additive; pigments are subtractive.

Illumination Models

Materials can absorb, emit, and scatter light.

Surface scattering:

Surface Reflection
- diffuse (Lambertian): radiance = \( k \times (N \cdot L) \)
- radiance is independent of direction (view-independent)
- specular (mirror-like)

Phong Illumination model = diffuse + specular
General: reflectance is a fn. of incoming & outgoing directions

Notes 19, Computer Graphics 2, 15-463
Optics 2

Illumination Models (cont.)

Surface scattering (cont.)

Transmission

Similar to reflection, but for the opposite hemisphere.

Refraction is due to difference of density of materials (index of refraction), obeys Snell’s law.

Transparency

\[
\text{color} = (1-\text{transparency}) \times (\text{fog color}) + (\text{transparency}) \times (\text{background color})
\]

Fog (absorption of light in translucent material)

\[
\text{transparency} = e^{(-\alpha \times \text{thickness})}
\]

Shadows

point light sources have sharp shadows

area light sources have soft shadows (with umbra & penumbra)

Interreflection (Global Illumination)

light comes not just from light sources, but from all surfaces (or volumes!)

To simulate, need to approximate integral of radiances coming from all surfaces.

Shading 1

polygon shading methods

faceted (shade each polygon, interpolate nothing)

smooth shading:

Gouraud shading (using vertex normals, shade each vertex, interpolate shade)

Phong shading (using vertex normals, interpolate normals, shade each pixel)

Texture Mapping

Texture can be represented as an array or procedure, \(texture(u,v)\),

or as a solid texture, \(texture(x,y,z)\)

<table>
<thead>
<tr>
<th>TECHNIQUE</th>
<th>SHADING PARAMETER AFFECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>surface color mapping</td>
<td>surface color</td>
</tr>
<tr>
<td>bump mapping</td>
<td>normal vector</td>
</tr>
<tr>
<td>environment mapping</td>
<td>incident light color</td>
</tr>
<tr>
<td>specularity mapping</td>
<td>coefficient of specular reflection</td>
</tr>
<tr>
<td>transparency mapping</td>
<td>transparency</td>
</tr>
</tbody>
</table>
Shading 2

Ray Tracing = recursive ray casting

Follow paths of photons in reverse, from eye.
Recurse to simulate specular reflection and specular transmission.

To antialias ray tracing
use supersampling (multiple rays per pixel), adaptive and/or stochastic

To speed ray tracing
spatial data structures can be used to reduce # of ray-object intersection tests
hierarchical bounding volumes (need to compute good hierarchy)
uniform grid (poor if scene is inhomogeneous)
octree (more code than uniform grid, but works better on inhomog. scenes)

Distribution Ray Tracing (a.k.a. “distributed” ray tracing)

<table>
<thead>
<tr>
<th>EFFECT</th>
<th>DISTRIBUTES RAYS OVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>spatial antialiasing</td>
<td>pixel</td>
</tr>
<tr>
<td>motion blur</td>
<td>frame time</td>
</tr>
<tr>
<td>penumbra</td>
<td>area light source, when shooting shadow rays</td>
</tr>
<tr>
<td>depth of field</td>
<td>camera aperture</td>
</tr>
<tr>
<td>rough specular reflection</td>
<td>specular reflection angle</td>
</tr>
<tr>
<td>diffuse reflection</td>
<td>hemisphere</td>
</tr>
</tbody>
</table>

Shading 3

Radiosity

Simulates interreflection in diffuse scenes.
Typically important for indoor scenes, but less important for outdoor.
Radiosity computes shading on surfaces, since they’re diffuse and view-independent, you can then move the camera without re-shading.

Steps:
subdivide polygons into elements
compute form factors by computing visibility and doing approximate integration
solve system of equations (explicitly or implicitly) for radiosities of each element
display view of scene
Hardware

Simple, brute force methods are easiest to parallelize, pipeline.
Z-buffer good for real-time graphics (SGI, HP workstations).

Extended z-buffer algorithm can do
  Gouraud-shaded polygons
  texture mapping with pyramid filtering
  antialiasing by supersampling (16 samples per pixel)

Volume Rendering

Painter’s algorithm
  Draw voxels in back-to-front order.
  Requires careful reconstruction or rastering results.

Z-buffer with linked lists
  Draw objects in arbitrary order.
  Massive memory requirements - not widely used.

Ray Casting
  Scan screen space, find voxels affecting each pixel.
  Quite general.

Convert to Surface Model (a.k.a. “marching cubes”)
  Test values at cube (or tetrahedron) corners, polygonize contour surface in voxels through which surface passes.
  Allows conventional surface renderers to be used, but sometimes introduces undesirable artifacts.