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

algorithm	mem. for	shadows easy?	analytic antialiasing?	trans- parency?	**adv**antages and **dis**advantages
Painter's	image	N	partial	Y	ADV: easy implementation if no sorting required. DIS: sorting & intersecting polygons tricky.
z-buffer	image & z-buf	N	N	N	ADV: easy impl., general, draw in arb. order, good for complex scenes, parallelizable.
scanline	scene & active edge list	N	Y	Y	ADV: low mem. req. if simple scene. DIS: requires sorting, polys only, slow for complex scene.
ray casting	scene & spatial datastruc	Y	N	Y	ADV: very general. DIS: slow without spatial data structures.

Theoretical Comparison of Rendering Algorithms

s = #surfaces (e.g. polygons) t_s = time per surface (transforming, ...)
p = #pixels t_p = time per pixel (writing, incrementing, ...)
ℓ = #lights t_ℓ = time to light surface point w.r.t. one light
$a = \Sigma$ screen areas of surfs t_i = 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 = $s(t_s + \ell t_\ell) + at_p$ $\approx at_p$ if polygons big

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

worst case cost = $st_s + a(\ell t_\ell + t_p)$ $\approx a\ell t_\ell$ if polygons big

Ray casting with no shadows, no spatial data structures

worst case cost = $p(st_i + \ell t_\ell + t_p)$ $\approx pst_i$ if many surfaces

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 ℓ shadow rays

worst case cost = $p(2^d-1)[(\ell+1)st_i + \ell t_\ell]$ $\approx 2^d p\ell st_i$ 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×area×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×(N·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

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
    - color = (1-transparency) × (fog color) + (transparency) × (background color)
  - Fog (absorption of light in translucent material)
    - transparency = e^(-alpha × 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)*

TECHNIQUE	SHADING PARAMETER AFFECTED
surface color mapping	surface color
bump mapping	normal vector
environment mapping	incident light color
specularity mapping	coefficient of specular reflection
transparency mapping	transparency

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)

EFFECT	DISTRIBUTE RAYS OVER
spatial antialiasing	pixel
motion blur	frame time
penumbras	area light source, when shooting shadow rays
depth of field	camera aperture
rough specular reflection	specular reflection angle
diffuse reflection	hemisphere

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.