Ray Tracing

- simple basic algorithm
- well-suited for software rendering
- flexible, easy to incorporate new effects
  - Turner Whitted, 1990

Simple Ray Tracing

- view dependent method
- cast a ray from viewer’s eye through each pixel
- compute intersection of ray with first object in scene
- cast ray from intersection point on object to light sources

Recursive Ray Tracing

- ray tracing can handle
  - reflection (chrome/mirror)
  - refraction (glass)
  - shadows
  - spawn secondary rays
    - reflection, refraction
    - if another object is hit, recurse to find its color
  - shadow
    - cast ray from intersection point to light source, check if intersects another object

Basic Algorithm

```plaintext
RayTrace(r, scene)
  obj := FirstIntersection(r, scene)
  if (obj is null)
    return BackgroundColor;
  else
    refract_color := Black;
    return Shade(reflect_color, refract_color, obj);
```
Ray Generation
- camera coordinate system
  - origin: C (camera position)
  - viewing direction: v
  - up vector: u
  - x direction: x = v x u
- note:
  - corresponds to viewing transformation in rendering pipeline
    - like gluLookAt

Ray Generation
- other parameters:
  - distance of camera from image plane: d
  - image resolution (in pixels): w, h
  - left, right, top, bottom boundaries in image plane: l, r, t, b
- then:
  - lower left corner of image: O = C + d · v + l · x + b · u
  - pixel at position i, j (i=0..w-1, j=0..h-1):
    \[ P_{ij} = O + i \cdot \Delta x \cdot x + j \cdot \Delta y \cdot y \]

Ray Intersections: Spheres
- spheres at origin
- implicit function
- ray equation
- to determine intersection:
  - insert ray \( R(t) \) into \( S(x,y,z) \):
  - solve for \( t \) (find roots)
  - simple quadratic equation
- local surface information (normal...)
  - for implicit surfaces \( F(x,y,z)=0 \): normal \( n(x,y,z) \)
  - needs to be normalized!

Ray Intersections: Other Primitives
- implicit functions
  - spheres at arbitrary positions
  - same thing
  - conic sections (hyperboloids, ellipsoids, paraboloids, cones, cylinders)
  - same thing (all are quadratic functions!)
- polygons
  - first intersect ray with plane
  - linear implicit function
  - then test whether point is inside or outside of polygon (2D test)
  - for convex polygons
  - suffices to test whether point \( p \) is on the correct side of every boundary edge
  - similar to computation of outdated in line clipping (upcoming)

Ray Tracing
- issues:
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
  - efficient data structures so we don’t have to test intersection with every object

Geometric Transformations
- similar goal as in rendering pipeline:
  - modeling scenes more convenient using different coordinate systems for individual objects
  - problem
    - not all object representations are easy to transform
    - problem is fixed in rendering pipeline by restriction to polygons, which are affine invariant
    - ray tracing has different solution
    - ray itself is always affine invariant
    - thus: transform ray into object coordinates!

Local Lighting
- local surface information (normal...)
  - for implicit surfaces \( F(x,y,z)=0 \): normal \( n(x,y,z) \)
  - can be easily computed at every intersection point using the gradient
  - lightiing and shading
  - efficient data structures so we don’t have to test intersection with every object
  - example:
    \[ n(x,y,z) = \begin{bmatrix} \frac{\partial F}{\partial x} \\ \frac{\partial F}{\partial y} \\ \frac{\partial F}{\partial z} \end{bmatrix} \]
  - needs to be normalized!

Local Lighting
- local surface information
  - alternatively: can interpolate per-vertex information for triangles/meshes as in rendering pipeline
  - now easy to use Phong shading!
    - as discussed for rendering pipeline
  - difference with rendering pipeline:
    - interpolation cannot be done incrementally
    - have to compute barycentric coordinates for every intersection point (e.g plane equation for triangles)
Global Shadows

- approach
  - to test whether point is in shadow, send out shadow rays to all light sources
  - if ray hits another object, the point lies in shadow

Global Reflections/Refractions

- approach
  - send rays out in reflected and refracted direction to gather incoming light
  - that light is multiplied by local surface color and added to result of local shading

Total Internal Reflection

- heat/light source
- energy
- packets
- view-independent technique
- calculate solution for entire scene offline
- browse from any viewpoint in realtime

Ray Tracing

- issues:
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
  - efficient data structures so we don’t have to test intersection with every object

Optimized Ray-Tracing

- basic algorithm simple but very expensive
- optimize by reducing:
  - number of rays traced
  - number of ray-object intersection calculations
- methods
  - bounding volumes: boxes, spheres
  - spatial subdivision
  - uniform
  - BSP trees
- (more on this later with collision)

Example Images

Radiosity

- radiosity definition
- rate at which energy emitted or reflected by a surface
- radiosity methods
  - capture diffuse-diffuse bouncing of light
  - indirect effects difficult to handle with raytracing

Ray Tracing

- illumination as radiative heat transfer
- conserve light energy in a volume
- model light transport as packet flow until convergence
- solution captures diffuse-diffuse bouncing of light
- view-independent technique
- calculate solution for entire scene offline
- browse from any viewpoint in realtime

Better Global Illumination

- ray-tracing: great specular, approx. diffuse
- view dependent
- radiosity: great diffuse, specular ignored
- view independent, mostly-enclosed volumes
- photon mapping: superset of raytrace and radiosity
- view dependent, handles both diffuse and specular well
- photon mapping

Subsurface Scattering: Translucency

- light enters and leaves at different locations on the surface
  - bounces around inside
- technical Academy Award, 2003
- Jensen, Marschner, Hanrahan

Subsurface Scattering: Marble

- illumination as radiative heat transfer
- conserve light energy in a volume
- model light transport as packet flow until convergence
- solution captures diffuse-diffuse bouncing of light
- view-independent technique
- calculate solution for entire scene offline
- browse from any viewpoint in realtime

Subsurface Scattering: Milk vs. Paint

- simulate look of hand-drawn sketches or paintings, using digital models

Subsurface Scattering: Skin

- simulate look of hand-drawn sketches or paintings, using digital models

Non-Photorealistic Rendering