Ray-Tracing
Global Illumination Models

- Simple, scan-conversion based, shading methods simulate local illumination models
  - Little object-object interaction

- To simulate global illumination models, one needs more sophisticated and probably more computation-intensive algorithms

- Ray-tracing can properly deals with
  - Reflections
  - Refractions and Transparency
  - Shadows
  - Caustics (to a limited extent)
Reflection and Refraction

Snell law reflects the connection between different transparent materials, $c_i$, and the angular deviation of the ray:

\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}
\]
A Basic Ray-Tracing Algorithm

```plaintext
RayTrace(r, scene)
obj := FirstIntersection(r, scene)
if (no obj) return BackgroundColor;
else begin
    if (Reflect(obj)) then
        ReflectColor := RayTrace(ReflectRay(r, obj), scene);
    else
        ReflectColor := Black;
    if (Transparent(obj)) then
        RefractColor := RayTrace(RefractRay(r, obj), scene);
    else
        RefractColor := Black;
    return Shade(ReflectColor, RefractColor, obj);
end;
```
Sub-Routines

- ReflectRay\(r,\) obj – computes a reflected ray (use obj normal at the intersection point)

- RefractRay\(r,\) obj - computes a refracted ray, following Snell’s law
  - Note: ray is inside obj

- Shade(ReflectColor, RefractColor, obj) – compute the illumination of obj at the point of intersection with ray, taking into account ReflectColor, RefractColor and the shaping properties of obj

Question: So what is so special in ray-tracing?
What is the most difficult task in this basic ray tracing process?
Ray-Object Intersections

- In the Kernel of every ray-tracing process
- Ray-Object intersections are computed millions of times for a single image and hence must be highly efficient
- **Example**: Ray-Sphere intersection

**ray**: \( x(t) = p_x + v_xt, \quad y(t) = p_y + v_yt, \quad z(t) = p_z + v_zt \)

(unit) **sphere**: \( x^2 + y^2 + z^2 = 1 \)

and the intersections points are at the solution of a quadratic equation in \( t \):

\[
0 = (p_x + v_xt)^2 + (p_y + v_yt)^2 + (p_z + v_zt)^2 - 1
\]

\[
= t^2(v_x^2 + v_y^2 + v_z^2) + 2t(p_xv_x + p_yv_y + p_zv_z)
\]

\[
+ (p_x^2 + p_y^2 + p_z^2) - 1
\]
Ray-Object Intersections

- Efficient algorithms exist to compute ray-object intersections for:
  - Primitives – Box, Sphere, Cone, Cylinder, Torus, etc..
  - Quadrics – $Ax^2 + By^2 + Cz^2 + Dxy + Exz$
    + $Fyz + Gx + Hy + Iz + J = 0$
  - Polygons
  - Volumetric Data

- Freeform surface are typically approximated by large sets of polygons

- Direct ray-surface intersection is not robust enough and is subject for contemporary research.

- **Question**: How many intersections a ray can have with a Quadrics? A Torus?
More About Ray-Tracing

- The basic ray-tracer above has a BUG: It simply never terminates
- Possible termination Criteria
  - No intersection
  - The real world has no ideal mirrors. Each reflection/refraction has a lesser affect on the original pixel
    - Contribution of secondary reflected and/or refracted ray is ignored below a prescribe threshold
  - Some maximal depth has been reached
Optimized Ray-Tracing

Basic algorithm is simple but VERY expensive. For each pixel, the number of rays grows exponentially with the depth of the recursion tree.

Optimized ray-tracing is a whole different game
- Reduce number of rays traced
- Reduce number of ray-object intersection calculations

Methods
- Bounding Boxes
- Object Hierarchies
- Spatial Subdivision (Octrees/BSP)
- Tree Pruning (Randomized)
Simulating Shadows

- Trace ray from each ray-object intersection point to light source(s)
  - If the ray intersects an object in between ⇒ point is shadowed from the light source
  - Otherwise, the light source illuminates the point

- A shadow computation routine should be added

```plaintext
Shadow = RayTrace(LightRay(obj,r,light), scene);
only to be included in the final shading:
return Shade(Shadow, ReflectColor, RefractColor, obj);
```
Ray-Tracing With Shadows

Eye → Image Plane → Light Source

- Reflected Ray
- Refracted Ray
Advanced Phenomena

- Ray tracers can (not always efficiently) simulate
  - Soft Shadows
  - Fog
  - Frequency Dependent Light. Snell law is different for different wave-lengths
  - Barely handle S*DS*
    - S – Specular
    - D – diffuse
  - Radiosity is a global scheme complementing ray-tracing that can aid in handling S*DS*