Ray-Tracing

Ray-tracing Overview

• handles multiple inter-reflections of light
• partly physics-based: geometric optics
• well suited to transparent and reflective objects

Trace light path from the eye backwards(!) into the scene; recursively apply to reflected and refracted rays.

Figure 1: Reflection test: (left) with environment map. (right) with environment map and ray-traced interreflections.

[Pixar: Ray Tracing for the Movie ‘Cars’
Ray-Tracing

```
raytrace( ray ) {
    find closest intersection: P
    colour_local = (0,0,0);
    if visible(P,L) // cast shadow ray
        colour_local = Phong(N,L,rayDir)
    colour_reflect = raytrace( reflected_ray ) // if reflective
    colour_refract = raytrace( refracted_ray ) // if refractive
    colour = k1*colour_local +
        k2*colour_reflect +
        k3*colour_refract
    return( colour )
}
```

• “raycasting” : only cast first ray from eye
Ray termination

- ray hits a diffuse object
- ray exits the scene
- when exceeding max recursion depth
- when final contribution will be too small

Generation of Rays

- distance to image plane: \(d\)
- image resolution (in pixels): \(N_x, N_y\)
- image plane dimensions: \(left, right, top, bot\)
- pixel \(i, j\)

\[
P_{0,0} = C + d \vec{w} + left \vec{u} + bot \vec{v}
\]

\[
P_{i,j} = P_{0,0} + i\Delta u \vec{u} + j\Delta v \vec{v}
\]

where

\[
\Delta u = (right - left) / N_x
\]

\[
\Delta v = (top - bot) / N_y
\]
Ray-Sphere Intersections

Ray
\[ R_{i,j}(t) = C + t \cdot (P_{i,j} - C) \]
\[ = C + t \cdot v_{i,j} \]
\[ x(t) = C_x + V_x t \]
\[ y(t) = C_y + V_y t \]
\[ z(t) = C_z + V_z t \]

Sphere
\[ F(x, y, z) = r^2 - (x - S_x)^2 - (y - S_y)^2 - (z - S_z)^2 \]

Ray-Triangle Intersections
Ray-Tracing: Optimizations

- process rays in parallel (multi-core, GPU, …)
- efficient ray-object culling
  - hierarchical bounding volumes