Ray-Tracing

University of
British Columbia

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

[Pixar: Ray Tracing for the Movie 'Cars'
http://graphics.pixar.com/library/RayTracingCars/paper.pdf]

raytracing
+ inter-reflections
+ (better) shadows
+ refraction effects

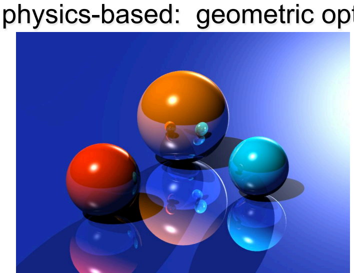
Compare:
Physics light → surface A → surface B → ey

Raytracing eye → surface B → surface A → ligh

Ray-tracing Overview

University of
British Columbia

- flexible and simple* algorithm
- well suited to transparent and specular objects
- global illumination*
- partly physics-based: geometric optics



[http://www.futuretech.blinkenlights.nl/c-ray.html]

Ray-Tracing

University of
British Columbia

```
raytrace( ray ) {
   find closest intersection
   cast shadow ray(s), compute colour_local
   colour_reflect = raytrace( reflected_ray )
   colour_refract = raytrace( refracted_ray )
   colour = k1*colour_local +
               k2*colour_reflect +
               k3*colour_refract
   return( colour )
}
```

R

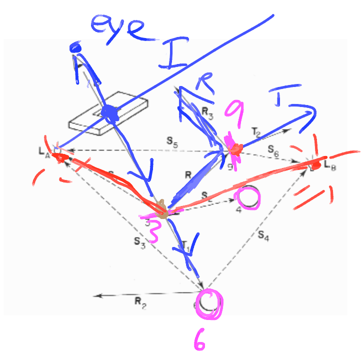
= transmitted ray T

combining (1) reflected colour (2) transmitted color (3) local colour

- "raycasting" : only cast first ray from eye

Ray-Tracing

University of
British Columbia



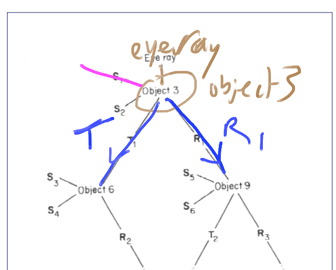
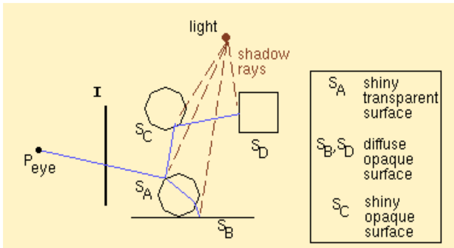


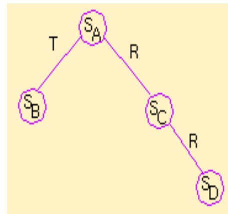
Fig. 12. The ray tree in schematic form.

Figure from Andrew S. Glassner, "An Overview of Ray Tracing" in An Introduction to Ray Tracing, Andrew Glassner, ed., Academic Press Limited, 1989.

Ray-Tracing

University of British Columbia





Ray-Tracing

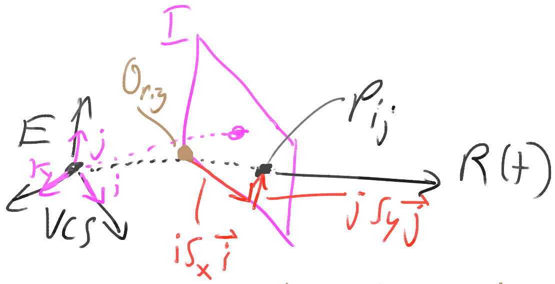
University of British Columbia

Ray Termination Criteria:

- diffuse surface
- ray exits scene
- maximal depth
- ray contribution below threshold

Generation of Rays

University of British Columbia



$$R(t) = E + t(\underline{P_{ij} - E})$$
$$= E + t\,\vec{V}_{ij}$$

where $P_{ij} = \text{Orig} + i\,\underbrace{s_x \vec{i}} + j\,s_y \vec{j}$

$$s_x = \frac{\text{right} - \text{left}}{\text{width in pixels}}$$

$$s_y = \frac{\text{top} - \text{bot}}{\text{height in pixels}}$$

Generation of Rays

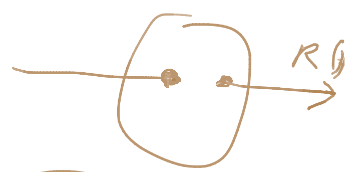
University of British Columbia

Ray Intersections

University of British Columbia

Ray-Sphere Intersections

University of British Columbia



Ray : $R(t) = E + t\vec{V}$

$x(t) = e_x + t v_x$

$y(t) = e_y + t v_y$

$z(t) = e_z + t v_z$

Sphere : $F(x,y,z) = (x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 - r^2 = 0$

$\Rightarrow$ quadratic in t

0 solutions : miss sphere

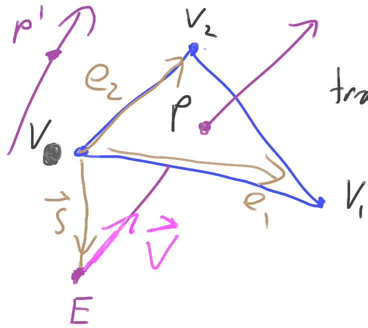
1 solution : touch the sphere

2 solutions : two intersections   take first one

smallest value of t

University of British Columbia

Ray-Triangle Intersections



$\alpha+\beta+\gamma=1$

$P(\alpha,\beta,\gamma) = \alpha V_0 + \beta V_1 + \gamma V_2 \quad 0 \le \alpha,\beta,\gamma \le 1$

triangle: $P(\beta,\gamma) = (1-\beta-\gamma)V_0 + \beta V_1 + \gamma V_2$

line: $P(t) = E + t\vec{V}$

$E + t\vec{V} = (1-\beta-\gamma)V_0 + \beta V_1 + \gamma V_2$

$\underbrace{E - V_0}_{\vec{S}} = -t\vec{V} + \beta\underbrace{(V_1 - V_0)}_{e_1} + \gamma\underbrace{(V_2 - V_0)}_{e_2}$

solve using determinants, etc. $\Rightarrow$

$\begin{bmatrix} \vec{S} \end{bmatrix} = \begin{bmatrix} -\vec{V} & \vec{e_1} & \vec{e_2} \end{bmatrix} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix}$ check $0 \le \beta,\gamma \le 1$

University of British Columbia

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!)
  - *Higher order functions (e.g. tori and other quartic functions)*
    - root-finding more difficult
    - resort to numerical methods

Ray-Tracing – Geometric Transformations

University of British Columbia

Transformations:

- Transform all rays into object coordinates
  - *Transform camera point and ray direction by inverse of model/view matrix*
- Shading has to be done in world coordinates (where light sources are given)
  - *Transform object space intersection point to world coordinates*
  - *Thus have to keep both world and object-space ray*