University of British Columbia

# Ray-Tracing

CPSC 414

Michiel van de Panne

## Ray-Tracing

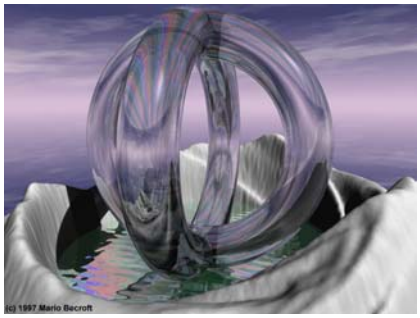
University of British Columbia

CAD Raytraced Image of Audi R8C



## Raytracing

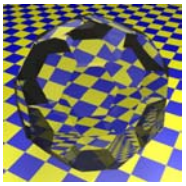
University of British Columbia

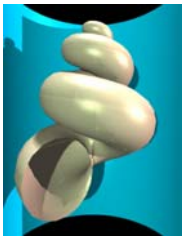





## Raytracing

University of British Columbia







## Overview

University of British Columbia

***So far***

- projective rendering (hardware)
- radiosity

***Ray-Tracing***

- simple algorithm for software rendering
- extremely flexible
- well suited to transparent and specular objects
- global illumination (*)
- partly physics-based: geometric optics



## Ray-Tracing

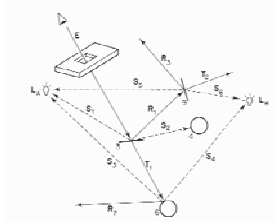
University of British Columbia

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

- "raycasting" : only cast first ray from eye

## Ray-Tracing

University of British Columbia



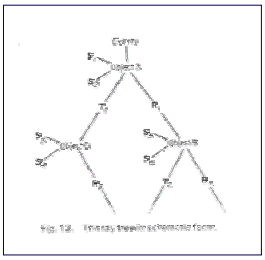
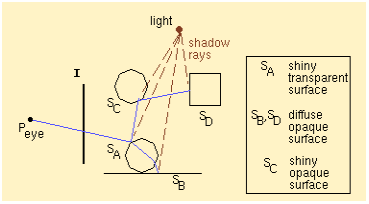


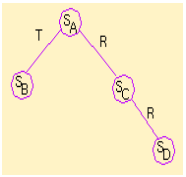
Figure from Andrew S. Glassner, "An Overview of Ray Tracing" in An Introduction to Ray Tracing, Andrew Glassner, ed., Academic Press Limited, 1989.



## Ray-Tracing

University of British Columbia






## Ray-Tracing

University of British Columbia

***Ray Termination Criteria:***

- **ray hits a diffuse surface**
- **ray exits the scene**
- **threshold on contrib. towards final pixel colour**
- **maximum recursion depth**

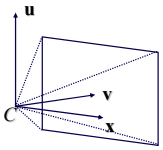


## Ray-Tracing – Generation of Rays

University of British Columbia

***Camera Coordinate System***

- Origin: C (camera position)
- Viewing direction: **v**
- Up vector: u
- x direction: **x**= **v**×**u**




***Note:***

- Corresponds to viewing transformation in rendering pipeline!
- See gluLookAt…

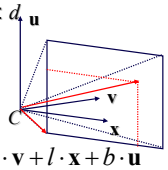


## Ray-Tracing – Generation of Rays

University of British Columbia

***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: $O = C + d \cdot \mathbf{v} + l \cdot \mathbf{x} + b \cdot \mathbf{u}$
- Pixel at position $i, j$ ($i$=0..$w$-1, $j$=0..$h$-1):

$$P_{i,j} = O + i \cdot \frac{r-l}{w-1} \cdot \mathbf{x} - j \cdot \frac{t-b}{h-1} \cdot \mathbf{u}$$

$$= O + i \cdot \Delta x \cdot \mathbf{x} - j \cdot \Delta y \cdot \mathbf{y}$$



## Ray-Tracing – Generation of Rays

University of British Columbia

***Ray in 3D Space:***

$$\mathrm{R}_{i,j}(t) = C + t \cdot (P_{i,j} - C) = C + t \cdot \mathbf{v}_{i,j}$$

where $t = 0 \ldots \infty$

## Ray Intersections


***Task:***

- Given an object o, find ray parameter $t$, such that $\mathbf{R}_{i,j}(t)$ is a point on the object
  - *Such a value for $t$ may not exist*
- Intersection test depends on geometric primitive



## Ray Intersections


***Spheres at origin:***

- Implicit function:

$$S(x,y,z): x^2 + y^2 + z^2 = r^2$$

- Ray equation:

$$\mathrm{R}_{i,j}(t) = C + t \cdot \mathbf{v}_{i,j} = \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} + t \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} c_x + t \cdot v_x \\ c_y + t \cdot v_y \\ c_z + t \cdot v_z \end{pmatrix}$$



## Ray Intersections


***To determine intersection:***

- Insert ray $\mathbf{R}_{i,j}(t)$ into $S(x,y,z)$:

$$(c_x + t \cdot v_x)^2 + (c_y + t \cdot v_y)^2 + (c_z + t \cdot v_z)^2 = r^2$$

- Solve for $t$ (find roots)
  - *Simple quadratic equation*



## 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 Intersections


***Other Primitives (cont)***

- 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 in on the right side of every boundary edge
    - Similar to computation of outcodes in line clipping



## Ray-Tracing – Geometric Transformations


***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*
    - This problem is fixed in rendering pipeline by restriction to polygons (affine invariance!)
  - *Ray-Tracing has different solution:*
    - The ray itself is always affine invariant!
    - Thus: transform ray into object coordinates!

# Ray-Tracing – Geometric Transformations

University of British Columbia

## *Ray Transformation:*

- For intersection test, it is only important that ray is in same coordinate system as object representation
- 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*