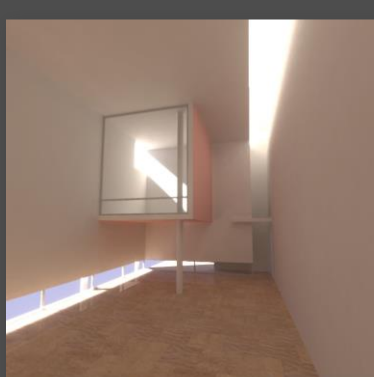
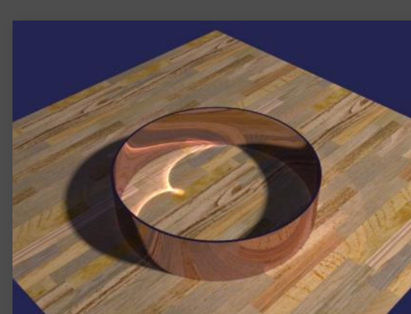
Ray-Tracing

Photorealism

Objective: To generate images that are as close as possible to those perceived by the human eye.

Method: Accurate simulation of optical phenomena.





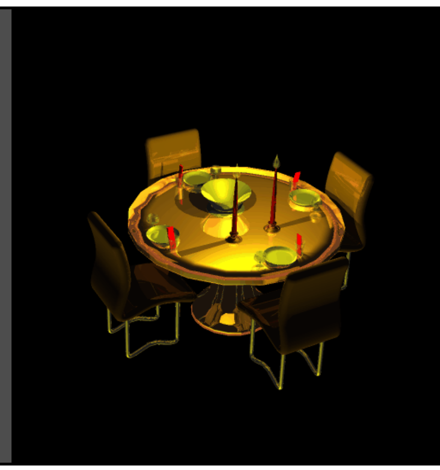
Illumination Models

Local: Depends only on the model, the light sources and the viewer. Easy to simulate.

Examples: Diffuse and specular illumination models. Flat, Gourard and Phong shading of polyhedra.

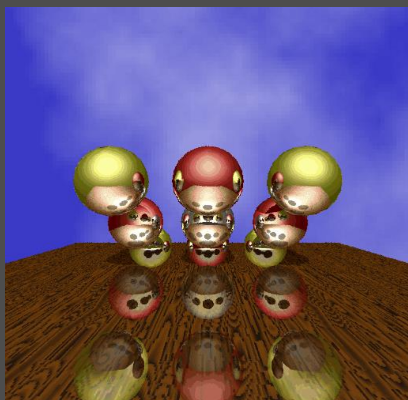
Global: Depends on the *entire* scene, the light sources and the viewer. Complex to simulate.

Examples: Shadow algorithms, ray-tracing, radiosity methods.

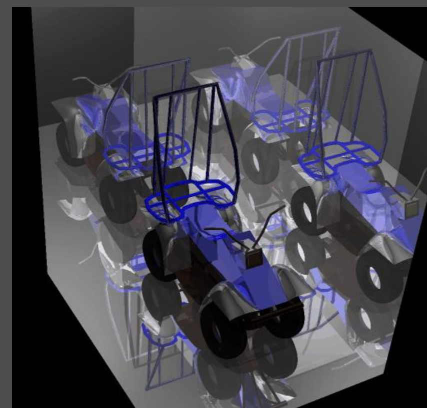


11

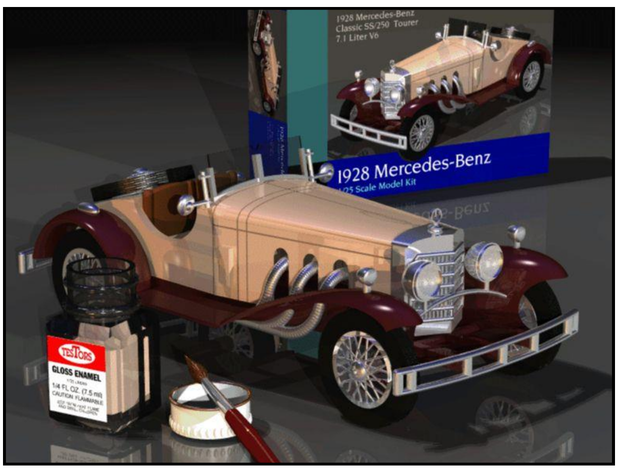
12



13



14

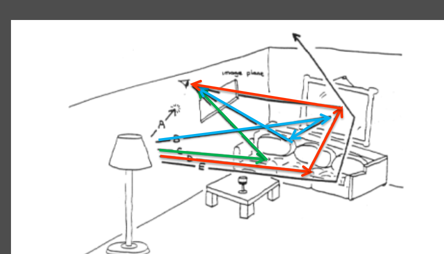


16

17

The Physics
Basic Illumination Model

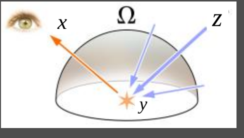
Light rays are emitted from light sources and bounce (reflect) in the scene until they reach the eye



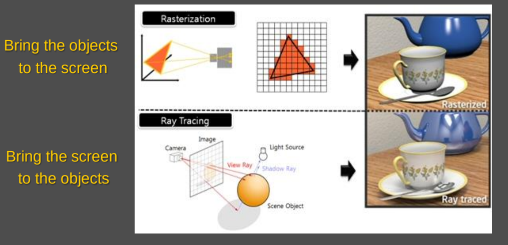
18

Rendering Equation

$$I(x,y) = g(x,y)[e(x,y) + \int_{Surf} s(x,y,z)I(y,z)dz]$$

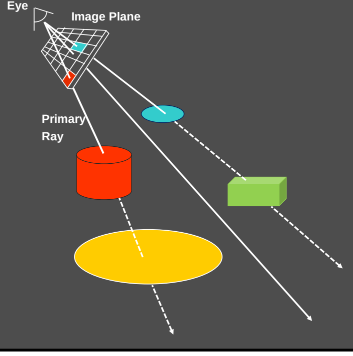


19



20

Ray-Casting Algorithm



21

A Basic Ray-Casting Algorithm

```
RayCast (r, scene)

        <obj, p> := FirstIntersection(r, scene);
        if (no obj)
                return BackgroundColor;
        else
                return Shade(p, obj);
        end;
```

22

Ray-Object Intersection

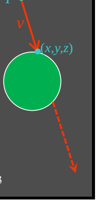
- In the kernel of every ray-tracer
- Ray-object intersections are computed millions of times for a single image, hence must be very efficient
- **Example**: Ray-Sphere intersection

ray: $x(t) = p_x + v_x t, \quad y(t) = p_y + v_y t, \quad z(t) = p_z + v_z t$

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

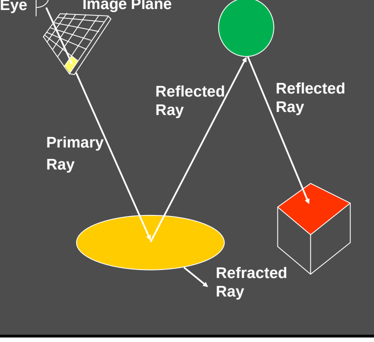
Solve a quadratic equation in t :

$$\begin{aligned} 0 &= (p_x + v_x t)^2 + (p_y + v_y t)^2 + (p_z + v_z t)^2 - 1 \\ &= t^2(v_x^2 + v_y^2 + v_z^2) + 2t(p_x v_x + p_y v_y + p_z v_z) \\ &\quad + (p_x^2 + p_y^2 + p_z^2) - 1 \end{aligned}$$



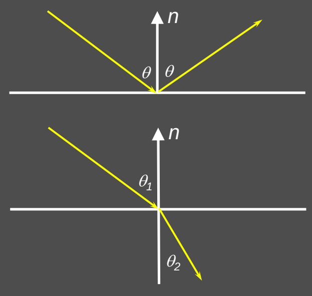
23

Ray-Tracing Algorithm



24

Reflection and Refraction



Snell's law

$$\frac{\sin \theta_1}{\sin \theta_2} = \frac{c_1}{c_2}$$

25

A Basic Ray-Tracing Algorithm

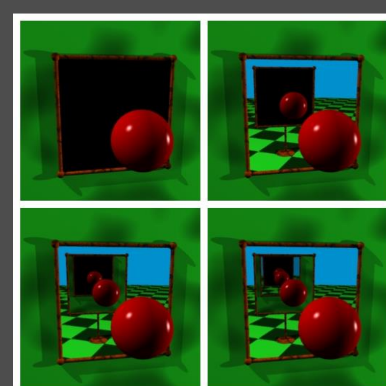
```
RayTrace(r, scene)
<obj, p> := FirstIntersection(r, scene);
if (no obj)  return BackgroundColor;
else begin
   if ( Reflect(obj) ) then
      ReflectColor := RayTrace(ReflectRay(r, p, obj));
   else
      ReflectColor := Black;
   if ( Transparent(obj) ) then
      RefractColor := RayTrace(RefractRay(r,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

26

Termination in Ray-Tracing

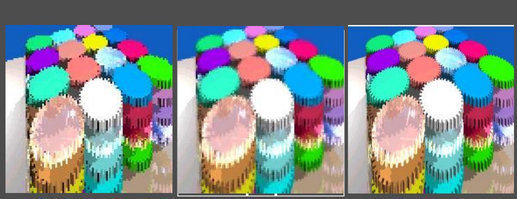
- Possible termination criteria:
  - No intersection
  - Contribution of secondary ray attenuated below a threshold
  - Maximal depth

27



28

Ray-Tracing as a Sampling Process

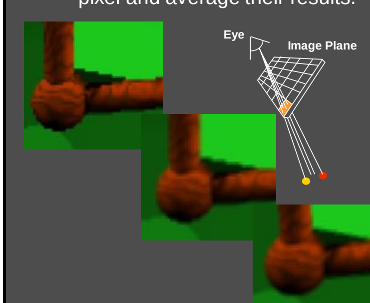


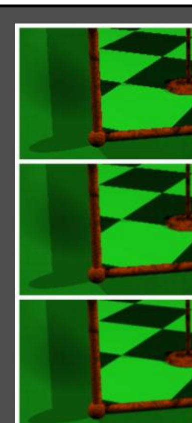
1 ray/pixel   16 rays/pixel   64 rays/pixel

29

Supersampling

Trace multiple primary rays per pixel and average their results.





30

JACOBS TECHNION-CORNELL INSTITUTE AT CORNELL TECH

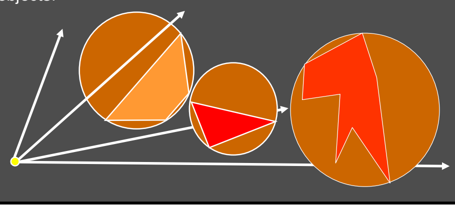
Optimized Ray-Tracing

- Basic algorithm is simple but VERY expensive.
- Optimized ray-tracing is critical
  - Reduce number of rays traced
  - Reduce number of ray-object intersection calculations
- Methods
  - Bounding Boxes
  - Object Hierarchies
  - Spatial Subdivision (Octrees/BSP)
  - Tree Pruning (Randomized)

31

Bounding Volumes

- Bound each scene object by a simple volume (e.g. sphere). This enables *fast reject* of non-intersections. More work is performed when there *is* an intersection (or near intersection).
- Since, on the average, a typical ray will not intersect the vast majority of the scene objects, this results in a significant speedup.
- The time complexity is still linear in the number of scene objects.



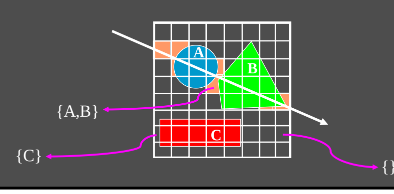
Choosing Bounding Volumes

Bounding volume should be:

- As tight-fitting as possible
- As simple as possible

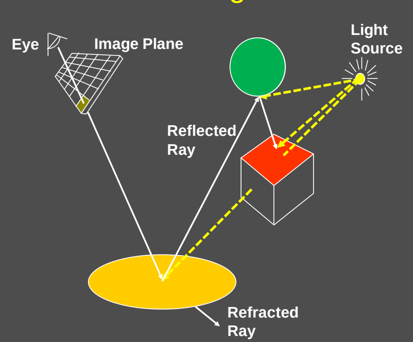
Uniform Spatial Partition

- 3D space is divided into voxels of identical size. Each voxel contains a list of objects it intersects. A tradeoff exists between voxel size and list length.
- **Disadvantage:** The subdivision is totally independent of the scene structure.
- **Advantages:**
  - Simple !
  - The voxels pierced by a ray may be accessed very efficiently by incremental calculation. A 3D version of Bresenham's algorithm is used.



34

Simulating Shadows



35

Simulating Shadows

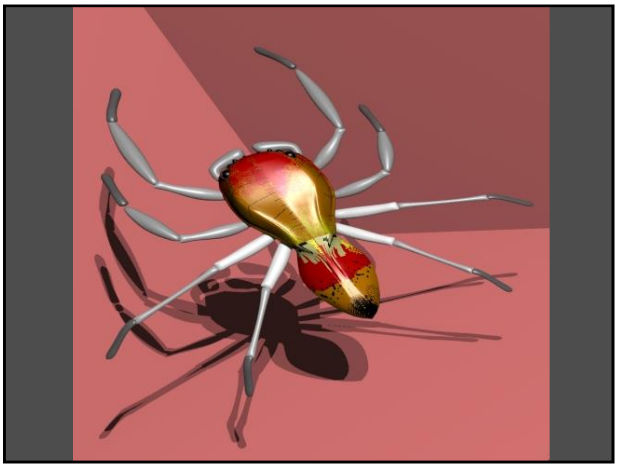
- Trace ray from each ray-object intersection point to light source(s)
  - If no line-of-sight ⇒ point is shadowed
- Shadow computation routine:

shadow = **RayTrace**(**LightRay**(p,obj,light));

to be included in the final shading:

return **Shade**(shadow, ReflectColor, RefractColor, p, obj);

36




mental images
"OFFICE FURNITURE SERIES "YPSILON""
Customer: Rosenthal AG
Design: Cini Boeri
Visualization: buenck+fehse, Berlin
Image rendered with mental ray.