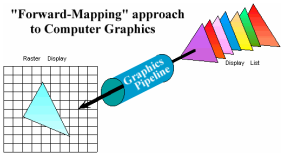
Raytracing

Chapter 10

Ray-tracing

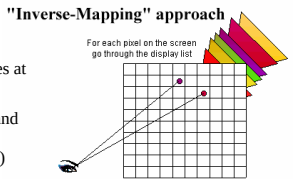
- ”Traditional” Computer Graphics



- Primitives are processed one at a time
- Sampling occurs last

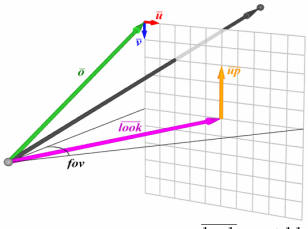
Ray-tracing

- One alternative is to invert the problem, lets from each pixel at the sceen shot a ray into the scene and return a color value depending on what the rat hits!



- Go through all primitives at each pixel
- Calculate intersection, and normal (to be used in illumination calculation)

Calculating the ray



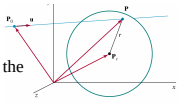
$$\bar{d} = \begin{bmatrix} u_x & v_x & o_x \\ u_y & v_y & o_y \\ u_z & v_z & o_z \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

$$\bar{u} = \frac{\overline{look} \times \overline{up}}{\left\| \overline{look} \times \overline{up} \right\|}$$

$$\bar{v} = \frac{\overline{look} \times \bar{u}}{\left\| \overline{look} \times \bar{u} \right\|}$$

$$\bar{o} = \frac{\overline{look}}{\left\| \overline{look} \right\|} \frac{width}{2\tan\left(\frac{fov}{2}\right)} - \frac{width}{2}\bar{u} - \frac{height}{2}\bar{v}$$

Object intersection

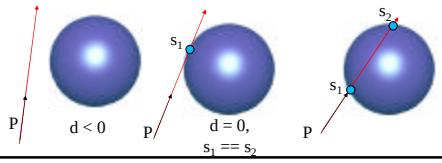


- We have to test for intersection of all objects in the scene
- Sphere/ray
- Any poiny P on the surface of the sphere satisfies the equation: $\left|\bar{P} - \bar{P}_c\right|^2 - r^2 = 0$
- Inserting the ray equation $\bar{P} = \bar{P}_0 + s\bar{u}$ results in: $\left|\bar{P}_0 - s\bar{u} - \bar{P}_c\right|^2 - r^2 = 0$
- Substituting P_c-P_0 as ? P and expanding the dot product results in a quadric equation: $s^2 - 2\left(\bar{u} \bullet \Delta\bar{P}\right)s + \left(\left|\Delta\bar{P}\right|^2 - r^2\right) = 0$

$$s = \bar{u} \bullet \Delta\bar{P} \pm \sqrt{\left(\bar{u} \bullet \Delta\bar{P}\right)^2 - \left|\Delta\bar{P}\right|^2 + r^2}$$

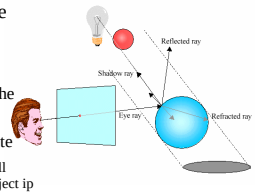
Object intersection

- Rejection check
  - If the discriminant d= $\left(\bar{u} \bullet \Delta\bar{P}\right)^2 - \left|\Delta\bar{P}\right|^2 + r^2$ is < 0 then the ray doesnt intersect the sphere, or the ray starts behind the sphere, in either case, we can ignore that sphere.



Global illumination

- With raytracing we can achieve global illumination
  - Recursive raytracing.
  - Cast a ray through a pixel in to the scene
  - At intersection point (ip) calculate
    - Shadow ray, shoot a ray towards all lights, if it intersects an opaque object ip is in shadow
    - If object is transparent, shoot a refraction ray through the object
    - Shoot a reflection ray in the direction of the reflection



Ray tracing

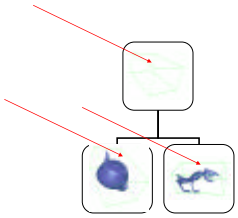
- **Advantages of Ray Tracing:**
- Improved realism over the graphics pipeline
  - Shadows
  - Reflections
  - Transparent surfaces with refraction
- Higher level rendering primitives
- Very simple design

Disadvantages:

- Very slow per pixel calculations
- Only approximates full global illumination
- Hard to accelerate with special-purpose H/W

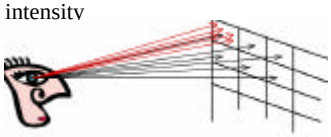
Acceleration

- Intersection tests can be as much as 95% of processing time of a ray-tracer
- Using trivial rejects
  - Bounding hiearchy
    - bounding boxes, bounding spheres
    - Spatial subdivision
  - First intersect the bounding volume, if the ray intersects the BV, then proceed and test the children of the BV.



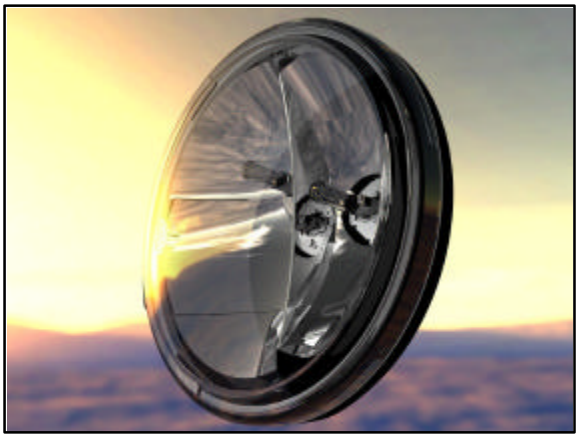
Antialiasing

- Well now we are there again!
- Sampling will always result in aliasing effects
- We could cast several rays per pixel, and jitter the pixels using a noise function and calculate the medium intensity



What about real-time?

- OpenRT a project for doing just that, ray tracing with global illumination in real-time
- So if you have a 20 node cluster at home, please go ahead!



What about games?