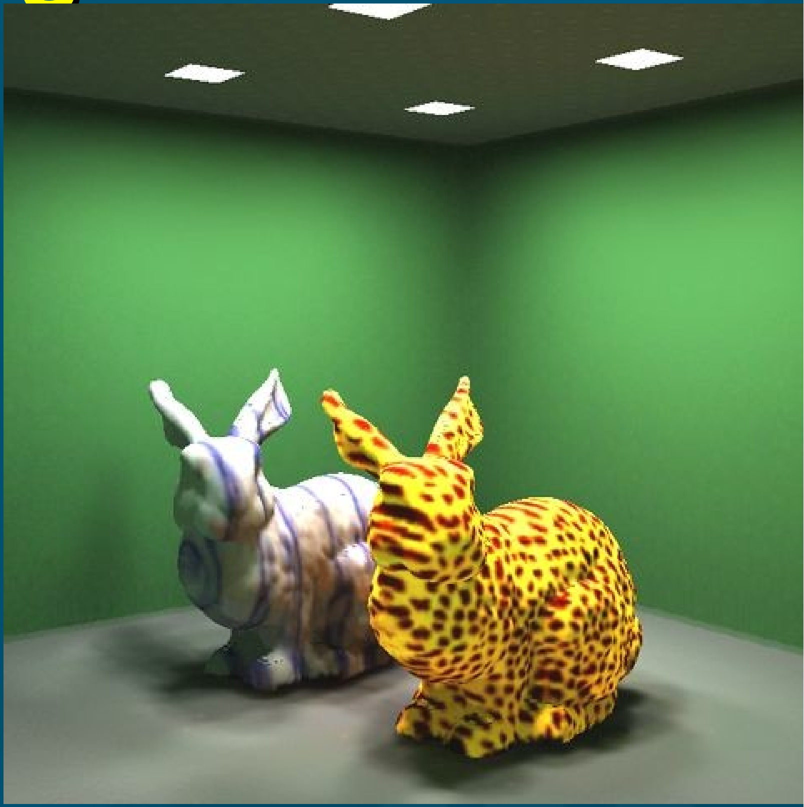
# Ray Tracing

3 April 2007
CMPT370
Dr. Sean Ho
Trinity Western University

# Review last time

- Using Bezier evaluators in OpenGL
- deCasteljau algorithm to compute Beziers
- B-splines
  - $C^2$ continuity
  - Knot spacing: uniform, open, non-uniform
  - NURBS

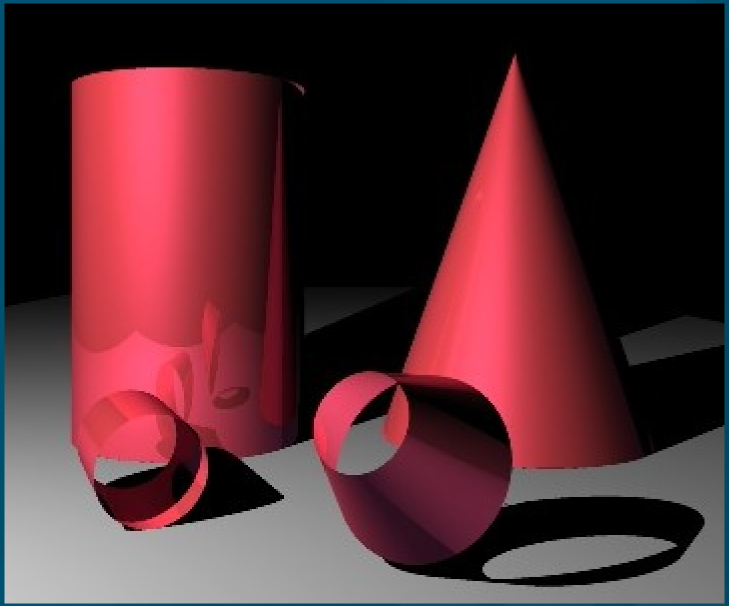
# Local vs. global rendering



- Local rendering:
  - Each object is rendered independently
  - Real-time OpenGL pipeline
- Global rendering:
  - Light scatters between objects
  - Approximates more of the global rendering equation
  - Usually computed off-line
    - Ray tracing: highlights, reflection, refraction
    - Radiosity: surface scattering

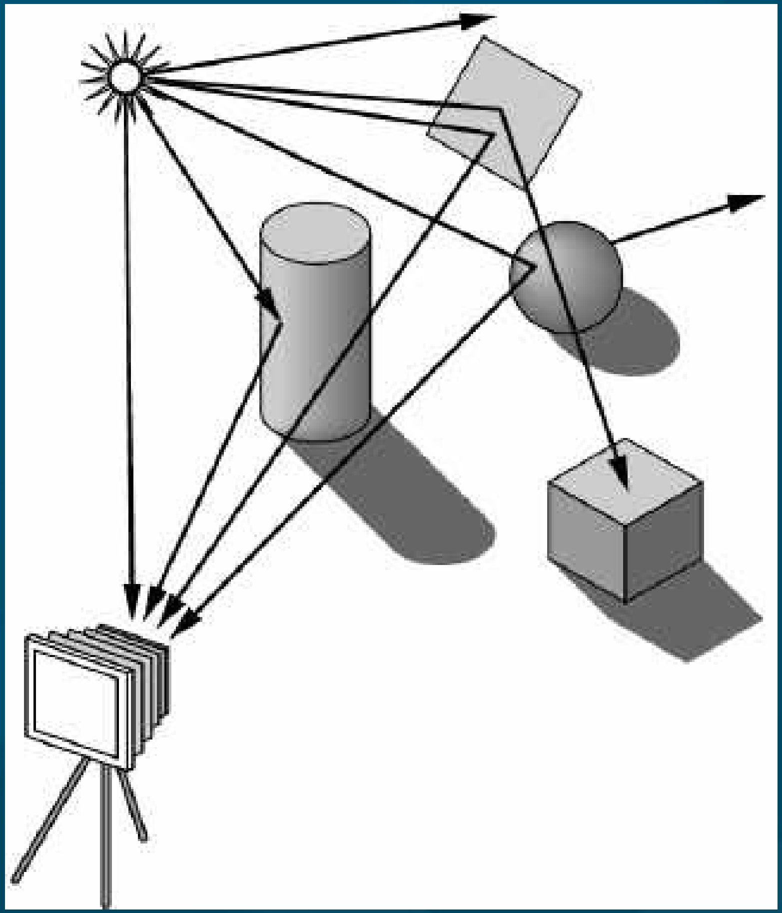
# Object-space vs. image-space

- OpenGL pipeline: object-space
  - Render one object at a time
  - Fast, easy, but doesn't model object-object interactions
- Ray tracing: image-space
  - Render one pixel at a time
  - Pixel-level parallelism
- Radiosity: surface patches
  - Find diffuse inter-reflections between each pair of surface patches

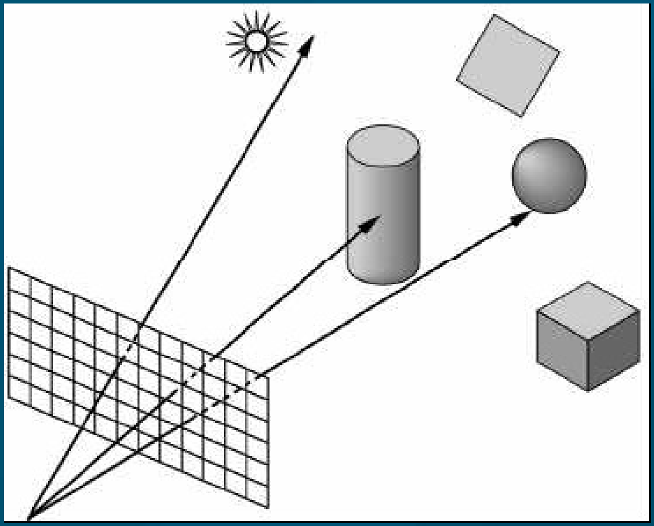


# Forward ray tracing

- Physically-based modelling
- Each light source emits photons
- Follow photons as they bounce around the scene and eventually the camera
- Problem: most photons won't contribute to the image!
  - Waste of computation



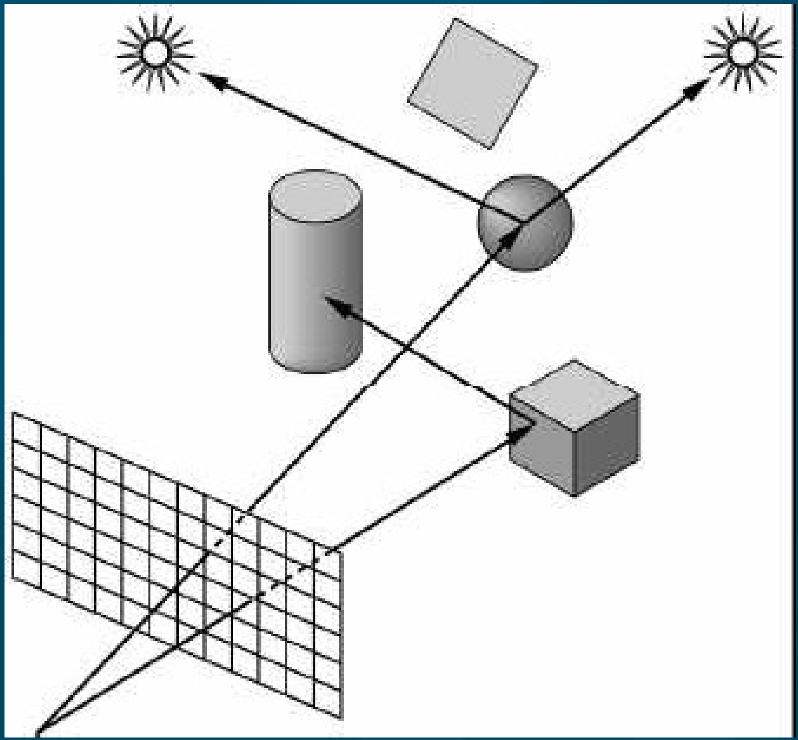
# Backward ray tracing



- First step is ray casting:
  - Fire rays from center of projection through each pixel in image plane
- When ray strikes an object, compute illumination:
  - Local illumination model (as in OpenGL)
  - Shadow rays
  - Reflection rays
  - Refraction rays
- Reflection/refraction: use recursion
- Critical operations: intersections, illumination

# Shadow rays

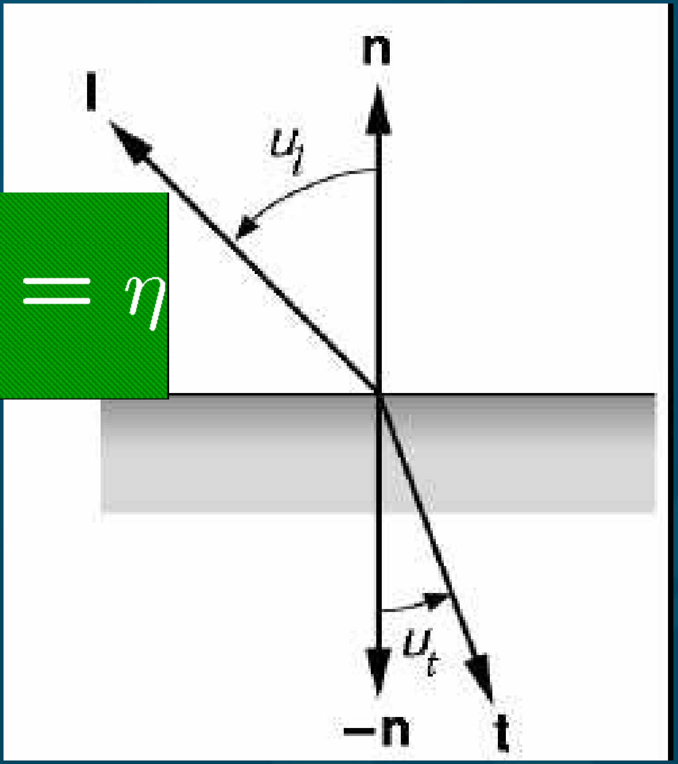
- An optimization to speed up local illumination
- Trace a shadow ray from the surface to each light
- If the shadow ray intersects any opaque object, then
  - That light does not contribute to the local illumination of this surface patch
  - Don't need to compute the illumination from that light
- Often over 90% of rays cast are shadow rays!



# Reflection and refraction rays

- Reflection ray
  - Replaces specular part of local illumination model
  - Compute backward reflection ray
- Refraction (transmission) ray
  - Models glass/water
  - Refractive index
- Recurse until either:
  - Ray exits scene (no intersect)
  - Contribution too dim
  - Fixed recursion depth

$$\frac{\sin(\theta_l)}{\sin(\theta_t)} = \frac{\eta_t}{\eta_l} = \eta$$



# Finding ray-surface intersections

- Easier to work with specialized kinds of objects:
  - Spheres
  - Planes
  - Polygons
  - Generalized parametric surfaces
  - Generalized implicit surfaces
  - Constructive solid geometry (CSG)
- We don't want to decompose a surface into lots of little triangles!

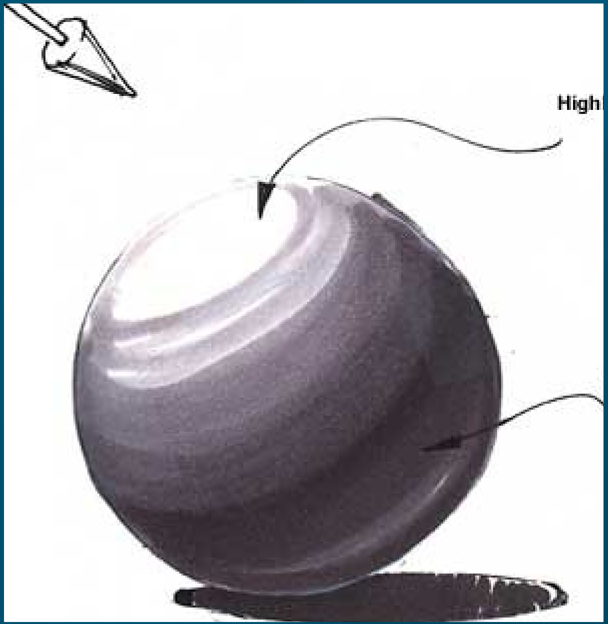
# Intersect ray with parametric surface

- Express the ray in parametric form:
  - Origin: $\mathbf{p_0} = [\ x_0, y_0, z_0, 1]^T$
  - Direction: $\mathbf{d} = [\ x_d, y_d, z_d, 0]^T$ (assume normalized)
  - The ray is: $\mathbf{p_0} + \mathbf{d}\ t$, for all $t > 0$.
- Express surface in parametric form:
  - $\mathbf{p}(u,v)$, for (u,v) in some bounds
- Solve (not easy for general surfaces):
  - $\mathbf{p_0} + \mathbf{d}\ t = \mathbf{p}(u,v)$
  - 3 equations, 3 unknowns (t,u,v)

# Intersect ray with implicit surface

- Express surface in implicit form:
  - Let $f: R^3 \rightarrow R^1$ be a function on 3D-space
  - The surface is the set of points **p** where $f(\mathbf{p}) = 0$
- Solve:
  - $f(\mathbf{p}_0 + \mathbf{d}\, t) = \mathbf{0}$
  - Solve for t
  - Check that $t>0$ (intersection in front of ray)
  - Also hard for general surfaces, but numerical approximation algorithms exist for root-finding

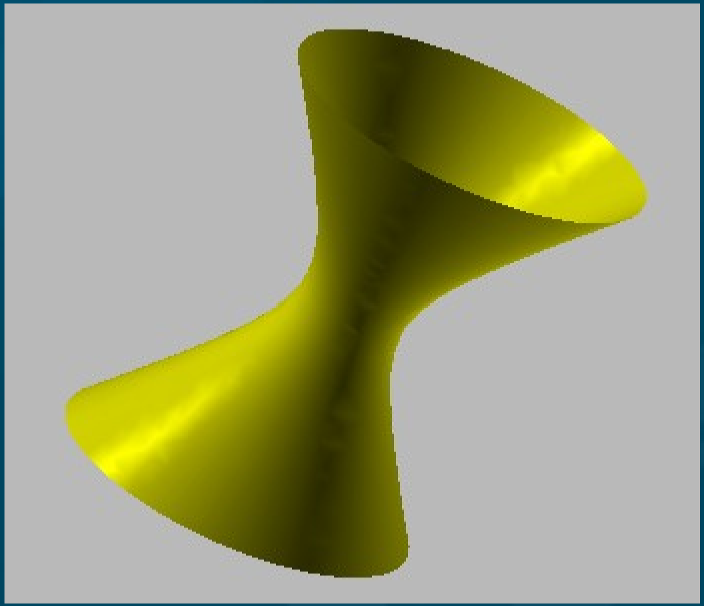
# Intersect ray with sphere



- Express the sphere in implicit form:
  - Centre: $\mathbf{c} = [\, x_c, y_c, z_c, 1]^T$
  - Radius: $r$
  - The sphere is:
    - $f(x,y,z) = (x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 - r^2 = 0$
- Solve: plug in ray equations for the point (x,y,z):
  - $(x_0 + x_d t - x_c)^2 + (y_0 + y_d t - y_c)^2 + (z_0 + z_d t - z_c)^2 - r^2 = 0$
  - Quadratic in t: solve using quadratic formula
- Also calculate normal vector on the sphere

# Intersect ray with quadric

- A quadric is any surface with an implicit form
    - $f(x,y,z) = 0$
  - Where f is a polynomial of order 2 (quadratic)
  - Examples: sphere, ellipsoid, cylinder, cone, etc.
- Solving for ray-quadric intersection:
  - Closed-form quadratic formula to solve for t
- Intersecting with one quadric faster than decomposing into polygons and testing each polygon



# Intersect ray with polygon

- First intersect ray with the plane containing the polygon
  - Implicit form of the plane:
    - $f(x,y,z) = ax + by + cz - h = 0$
  - Unit normal vector: $\mathbf{n} = [\ a\ b\ c\ 0\ ]^T$
  - Substitute ray equations and solve for t:
    - $t = -(\mathbf{n} * \mathbf{p}_0 + h) / \mathbf{n} * \mathbf{d}$ (use dot products)
- Check if intersection point lies within polygon:
  - Project onto 3 planes x=0, y=0, z=0
  - Check point-in-polygon in 2D