Computer Graphics

-Ray Tracing-

Oliver Bimber

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

NEXT ICG LAB TALK:
10. MAY 2016, 4:00PM

Institute of Computer Graphics

Dr. Bettina Heise

Johannes Kepler University Linz

(Low) Coherent Imaging: Current State, Trends and Perspectives

Science Park 2

Room S2 059

JKU

For more information about our talks visit
http://www.cg.jku.at/talks/invited

Local vs. Global Shading

- What we did so far:
  - We rendered (triangle) primitives
  - We computed shading parameters (e.g., color, lighting, texture, shadows, etc.) for each vertex
  - We interpolated these parameters during rasterization to estimate the appearance of each pixel on the screen
- If the shading parameters for each surface point are computed directly only with respect to the light sources (not considering indirect illumination), then this is called local shading
- What we haven't done so far:
  - We did not consider indirect lighting phenomena, such as inter-reflection, scattering, caustics, refraction, etc.
- Techniques that do consider such effects are called global shading techniques (e.g., ray tracing or radiosity)
- Shadow mapping, environment mapping, etc. are only approximations for real-time applications (their quality is limited)

local shading example

example for high-end ray traced graphics

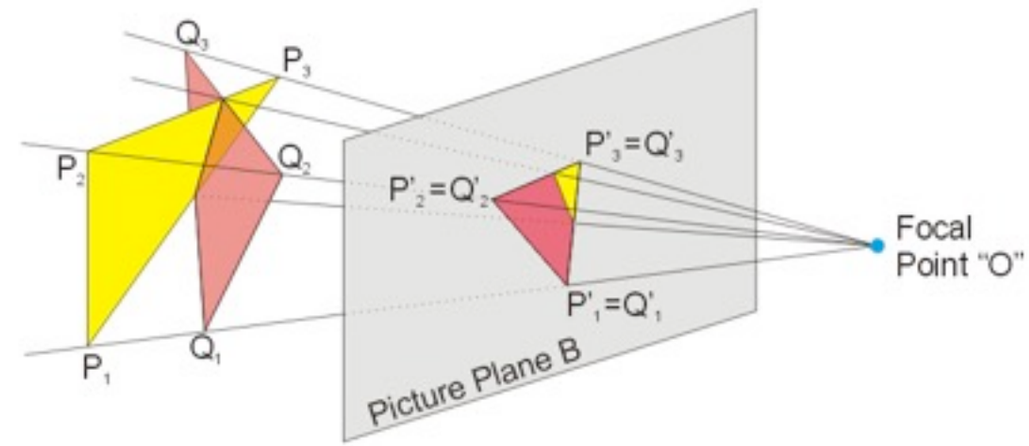
Ray Tracing Examples

Ray Tracing Examples

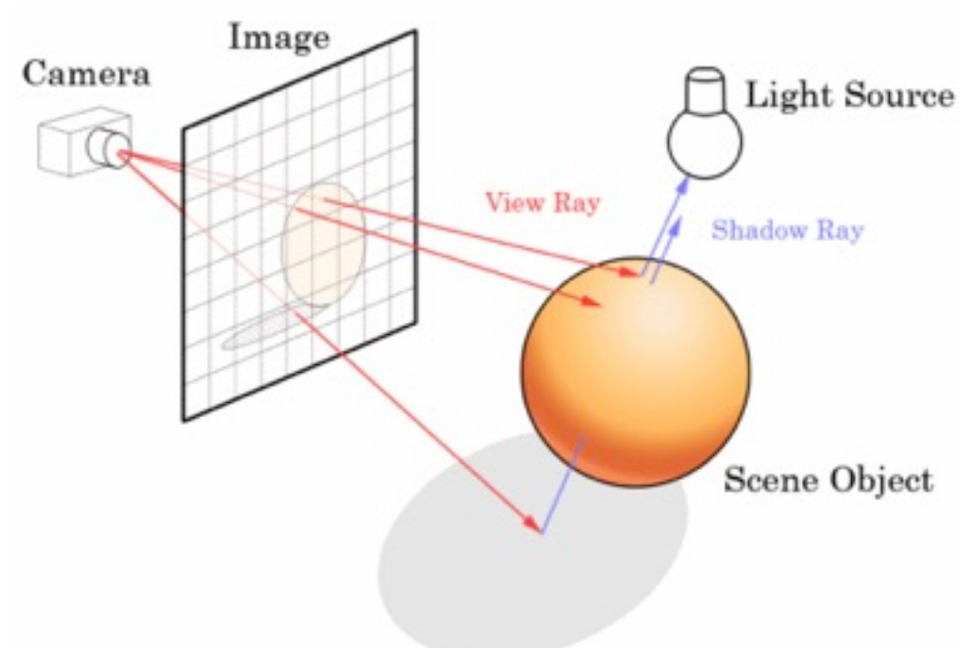
Ray Tracing Examples

Ray Tracing

- Ray tracing can be used for creating (photo-) realistic images
- It is very common in the movie industry (i.e., in contrast to games, special effects for movies do not require real-time rendering)
- Compared to what we have seen before, ray tracing is carried out pixel-by-pixel, rather than primitive-by-primitive
- For each pixel on the screen, a ray is traced through the scene, and its modulation with different objects is accumulated
- It is very easy to parallelize ray tracing - thus rendering can be done simultaneously on multiple computers (e.g., render farms)



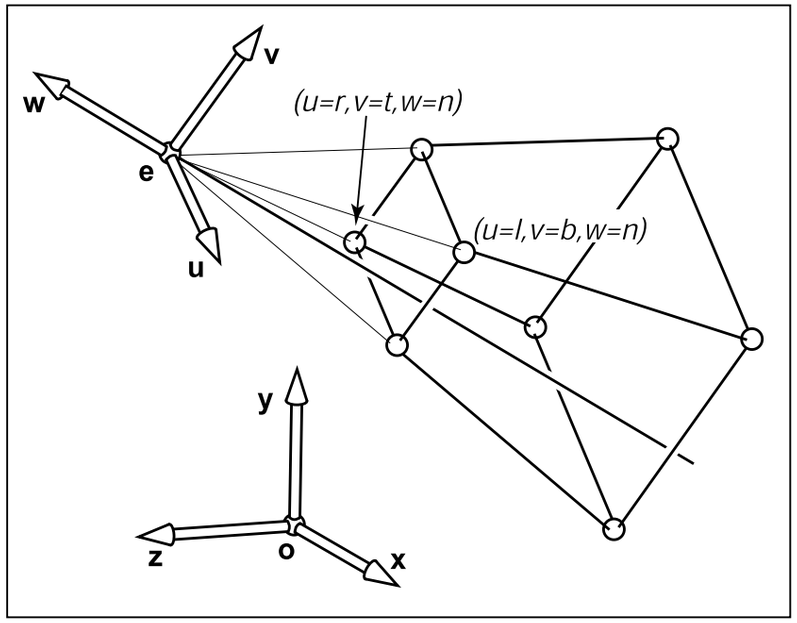
primitive-by-primitive projection and rasterization on screen



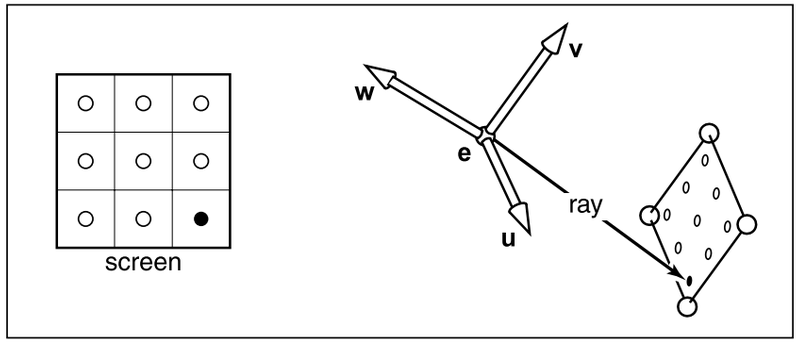
pixel-by-pixel ray tracing and accumulation of light modulation within scene

Ray Tracing Basics

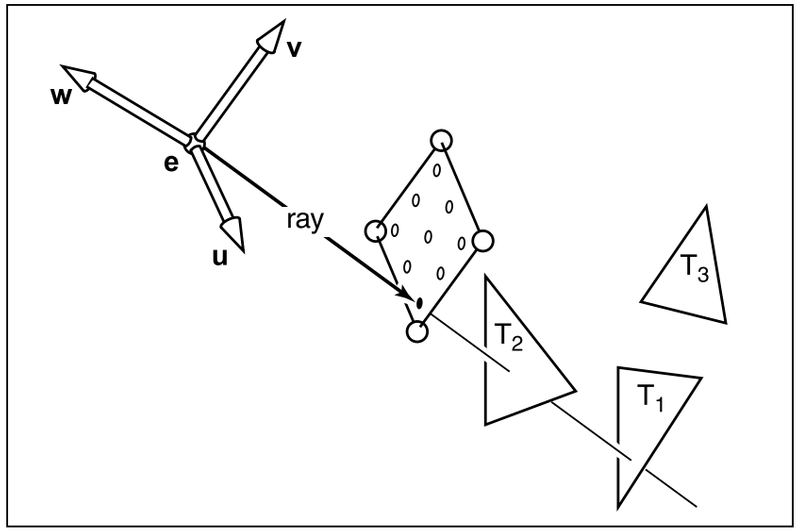
- Basically, we set up a CG camera as before (i.e., we define view transformation and projection matrices) - see transformations and projections
- But then, we cast rays from the camera center (e) through locations on the viewing plane (near clipping plane) that correspond to pixel-center projections (i.e., we compute viewing rays that origin at e and pass through corresponding pixel centers if we assume that viewing plane = screen plane)
- We compute the intersections of each ray with the first (triangle) primitive that is hit
- Properties of this triangle (like color, normal, transparency, etc.) influence the properties of the corresponding ray
- Based on these properties, a ray can be traced further (e.g., through reflection, refraction, scattering, etc), and we compute the intersection with the next triangle on the ray's new path



the viewing geometry is defined as before



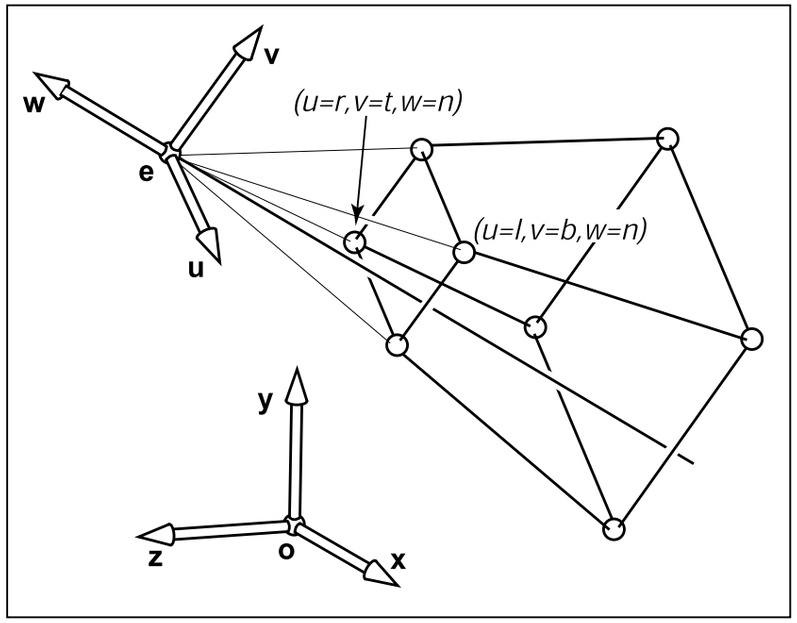
for each pixel center, a ray is cast through the viewing geometry



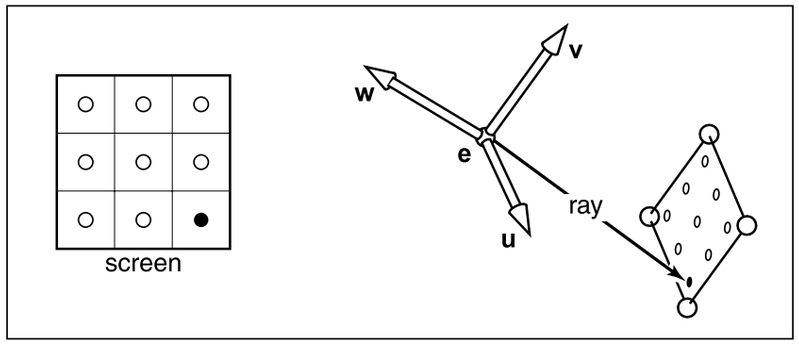
rays are intersected with scene geometry, traced and modulation properties are accumulated

Ray Tracing Basics

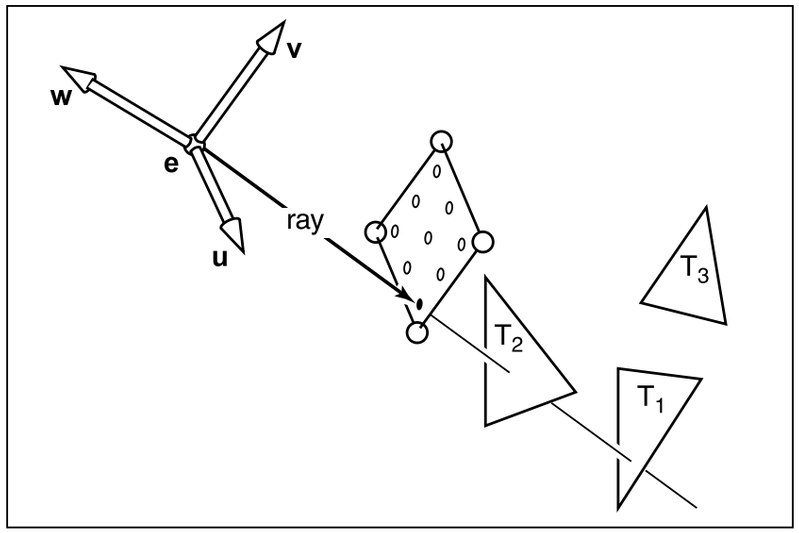
- We keep on doing this until we have reached an exit condition (e.g., a maximum path depth, or a ray does not intersect a scene object anymore)
- On the way through the scene, we also keep on accumulating modulation information and update the ray's properties
- The final properties of the ray are used to derive the color of the pixel to which it belongs
- We do this for all pixels (this can be done in parallel, since tracing rays for pixels are totally independent from each other)
- Having collected the information for all pixels gives us the final image
- The first rays that are cast through the pixel centers are called primary rays (or viewing rays)
- A ray that is traced further is called secondary (...) ray



the viewing geometry is defined as before



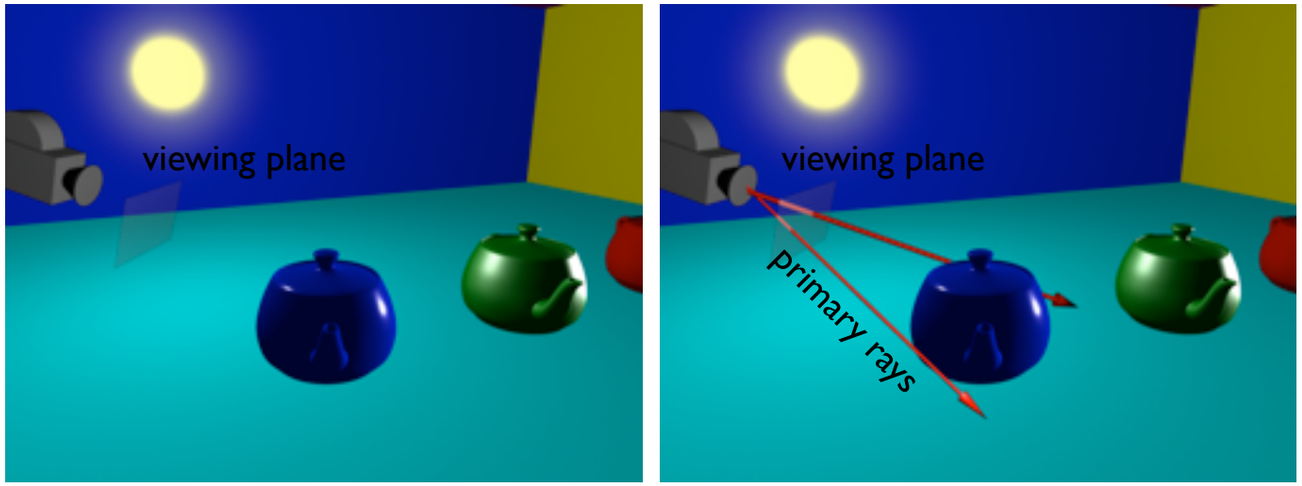
for each pixel center, a ray is cast through the viewing geometry



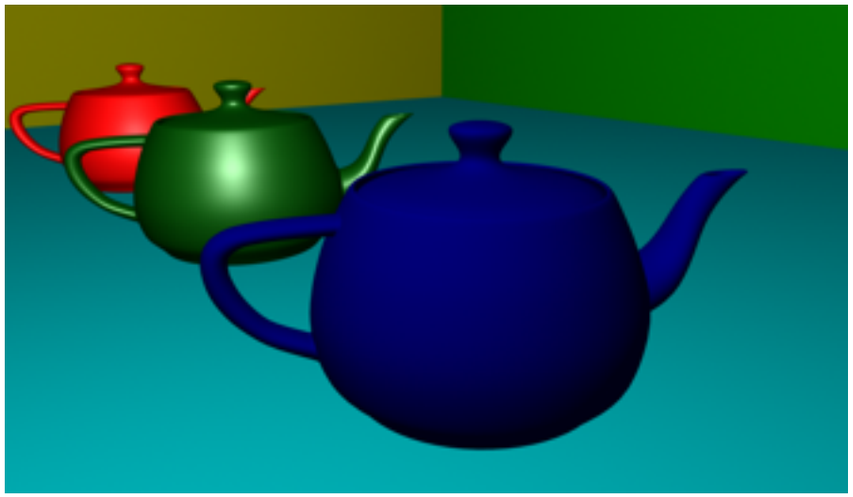
rays are intersected with scene geometry, traced and modulation properties are accumulated

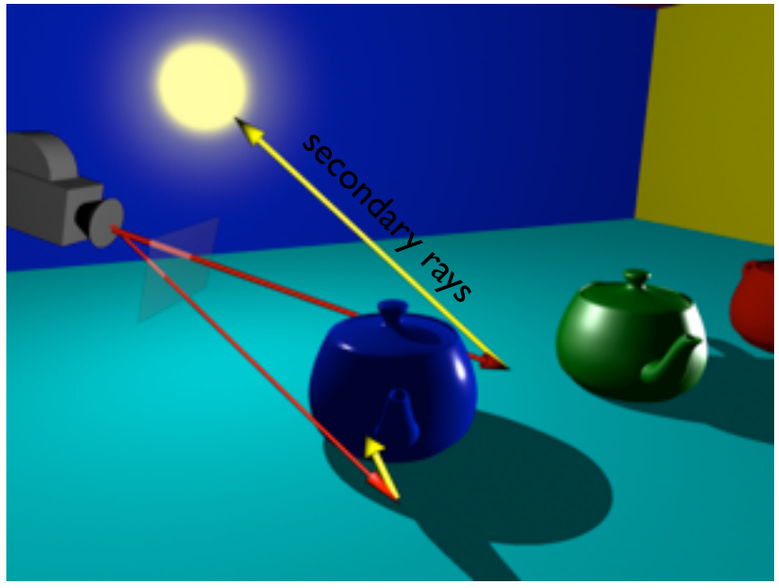
Ray Tracing vs. Ray Casting

- If only primary rays are computed and a local shading model is applied to surface points at ray intersections, this is called ray casting
- Ray casting does not require other depth handling (such as Z-buffer) or rasterization techniques
- Ray casting was an early rendering technique (Appel68), but it is still being used in some cases
- If rays are really traced (i.e., secondary rays are also considered), then this is referred to as ray tracing
- This allows computing modulation effects, such as reflections, shadows, refraction, etc (which is not possible with primary rays only)



ray casting consideres only primary rays

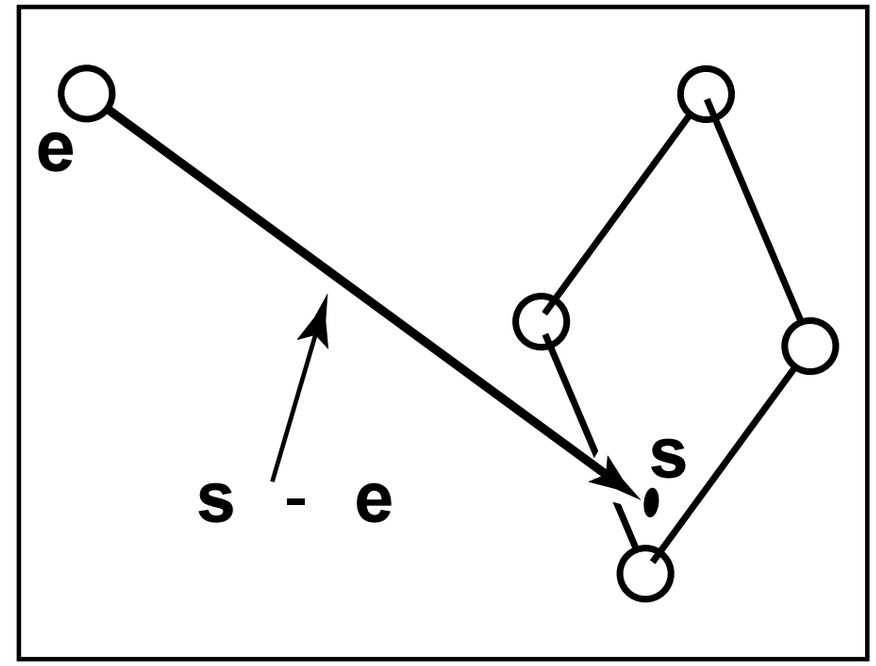




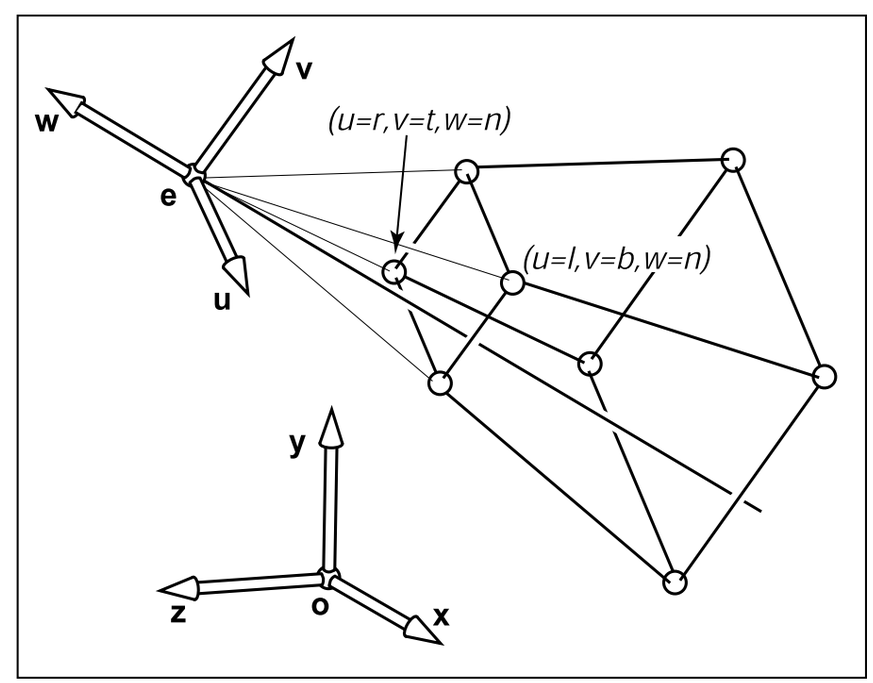
ray tracing traces rays further to compute indirect shading effects

Computing Viewing Rays

- Rays can be expressed with parametric line equation
- Thereby, e is the common origin and s is a point on the screen (viewing plane)
- p(0)=e and p(1)=s, with t<0 then p is behind e
- e is given and s can be computed by applying our transformation matrices
  - Either: define s in camera coordinate system (assuming that no view transformation exist, and e=0,0,0), then apply inverse view transformation to compute s,e in world coordinate system
  - Or: apply view transformation to scene first (as we defined it) and then ray trace in camera coordinate system (i.e., with e=0,0,0)



$$p(t) = e + t(s - e)$$



Ray-Object Intersection

- We want to find intersections with objects only at t>0
- We can intersect the ray with an implicit surface f(p)=0, such as a sphere
- Ray intersections are easy to compute if an implicit surface equation is given
- Spheres are handy when being used as bonding spheres (for making pre-intersection tests), but not all scene objects are spheres
- Thus, we need to compute ray triangle intersections, in the most general case
- But how?

$$f(p) = 0 = f(e + td), d = (s - e)$$

for a sphere with center at c=(c_x, c_y, c_z) with radius R:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - R^2 = 0$$

in vector form:

$$(p - c)(p - c) - R^2 = 0$$

plugging p(t) in and solving for t (quadratic equation):

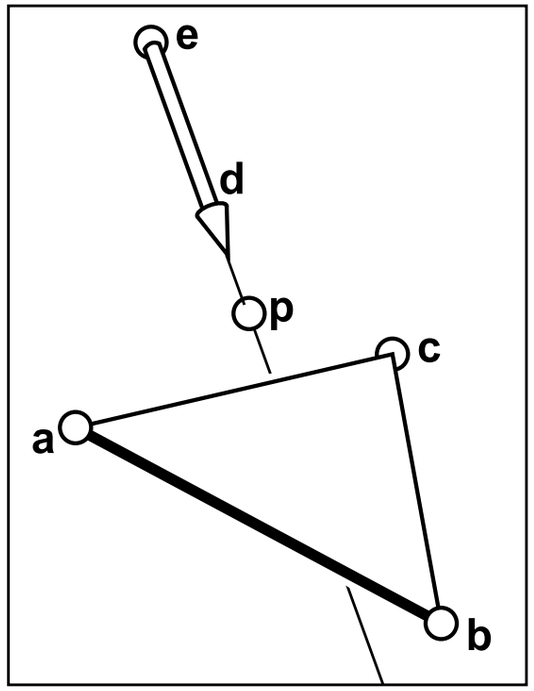
$$t = \frac{-d \cdot (e - c) \pm \sqrt{(d \cdot (e - c))^2 - (d \cdot d)((e - c)(e - c) - R^2)}}{(d \cdot d)}$$

given t, the intersection point p(t) and the unit normal at this point (computed from the gradient) are:

$$p(t) = e + td, n(t) = (p - c) / R$$

Ray-Object Intersection

- Several possibilities exist (we will use our barycentric coordinates and parametric plane equations, as discussed in depth handling and rasterization)
- We can equalize both parametric equations (ray and plane) and solve for the common intersection point
- There are three unknowns: t and the two barycentric coordinates
- However, we have also three equations: in x, y, and z!



given the three vertices (a,b,c) of a triangle, all points can be parameterized in the triangle's plane via barycentric coordinates:

$$p = a + \beta(b-a) + \gamma(c-a)$$

equalizing plane with ray:

$$e + td = a + \beta(b-a) + \gamma(c-a)$$

gives us a lin. equation system with 3 equations in 3 unknowns:

$$e_x + td_x = a_x + \beta(b_x - a_x) + \gamma(c_x - a_x)$$
$$e_y + td_y = a_y + \beta(b_y - a_y) + \gamma(c_y - a_y)$$
$$e_z + td_z = a_z + \beta(b_z - a_z) + \gamma(c_z - a_z)$$

Ray-Object Intersection

- Several possibilities exist (we will use our barycentric coordinates and parametric plane equations, as discussed in depth handling and rasterization)
- We can equalize both parametric equations (ray and plane) and solve for the common intersection point
- There are three unknowns: t and the two barycentric coordinates
- However, we have also three equations: in x, y, and z!
- The fastes way to solve this linear equation system is Cramer's rule

$$\begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - e_x \\ a_y - e_y \\ a_z - e_z \end{bmatrix}$$

solving for 3 unknowns via Cramer's rule:

$$\beta = \frac{\begin{vmatrix} a_x - e_x & a_x - c_x & d_x \\ a_y - e_y & a_y - c_y & d_y \\ a_z - e_z & a_z - c_z & d_z \end{vmatrix}}{|A|}, \gamma = \frac{\begin{vmatrix} a_x - b_x & a_x - e_x & d_x \\ a_y - b_y & a_y - e_y & d_y \\ a_z - b_z & a_z - e_z & d_z \end{vmatrix}}{|A|}$$

$$t = \frac{\begin{vmatrix} a_x - b_x & a_x - c_x & a_x - e_x \\ a_y - b_y & a_y - c_y & a_y - e_y \\ a_z - b_z & a_z - c_z & a_z - e_z \end{vmatrix}}{|A|}$$

with:

$$A = \begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix}$$

and |A| is determinant of A

Ray-Object Intersection

- Several possibilities exist (we will use our barycentric coordinates and parametric plane equations, as discussed in depth handling and rasterization)
- We can equalize both parametric equations (ray and plane) and solve for the common intersection point
- There are three unknowns: t and the two barycentric coordinates
- However, we have also three equations: in x, y, and z!
- The fastes way to solve this linear equation system is Cramer's rule
- Intersection point p can be computed again from t, but we have to check if p is inside the triangle (possibly combined with early termination strategies)

is ray within predefined range (e.g., 0..inf)

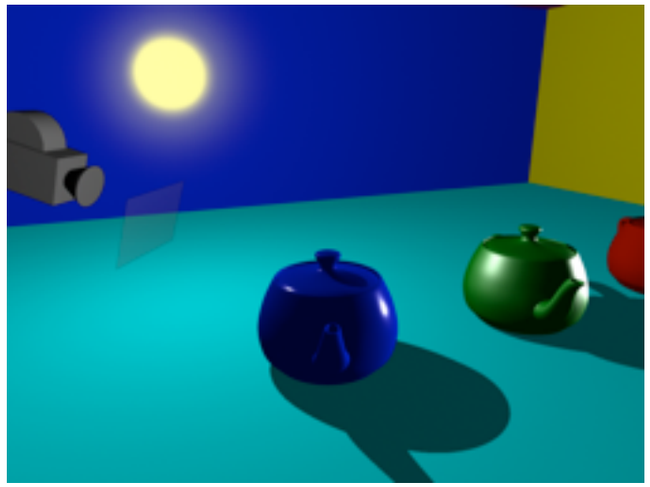
if ($t<t_0$) or ($t>t_1$) **then**
  return false
compute

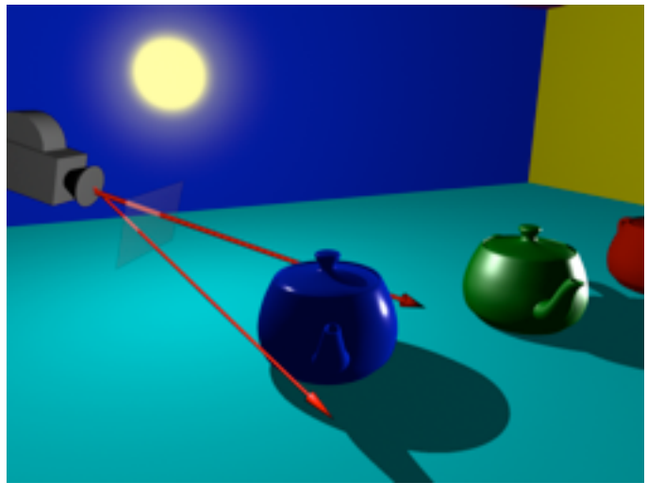
is intersection inside triangle

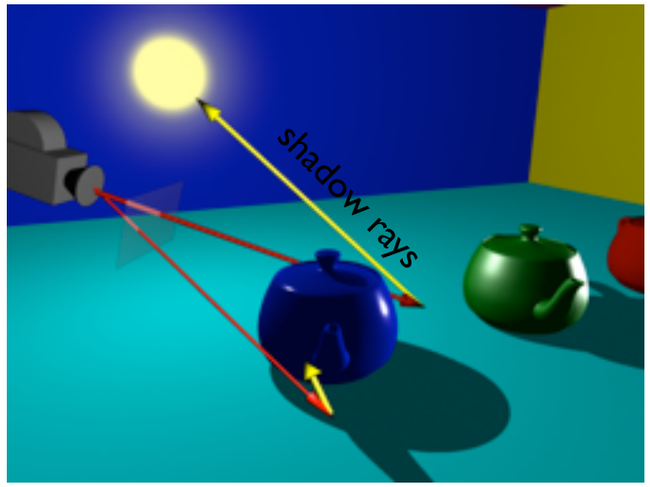
if ($\gamma<0$) **or** ($\gamma>1$) **then**
  return false
compute
if ($\beta<0$) **or** ($\beta<1-\gamma$) **then**
  return false
return true

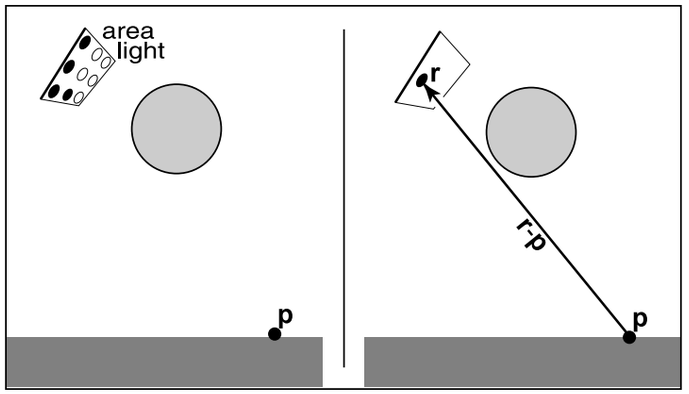
Shadows

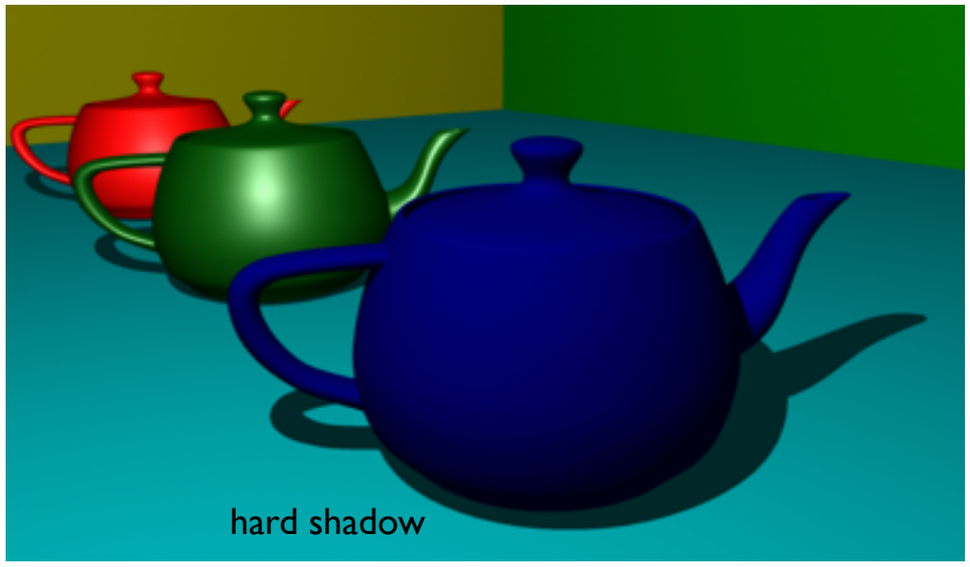
- Shadows can be rendered by finding out whether or not a surface point lies in shadow (is illuminated by light source)
- This can be achieved by shooting a secondary ray back to light source and see if it is blocked by (i.e., it intersects) another scene object
  - If so: intersection point is in shadow
  - If not: intersection point is illuminated
- If only one secondary ray (which is now called shadow ray) is cast per intersection point towards a point light sources, then this will lead to what is called hard shadows (this is also what you get with conventional shadow mapping)





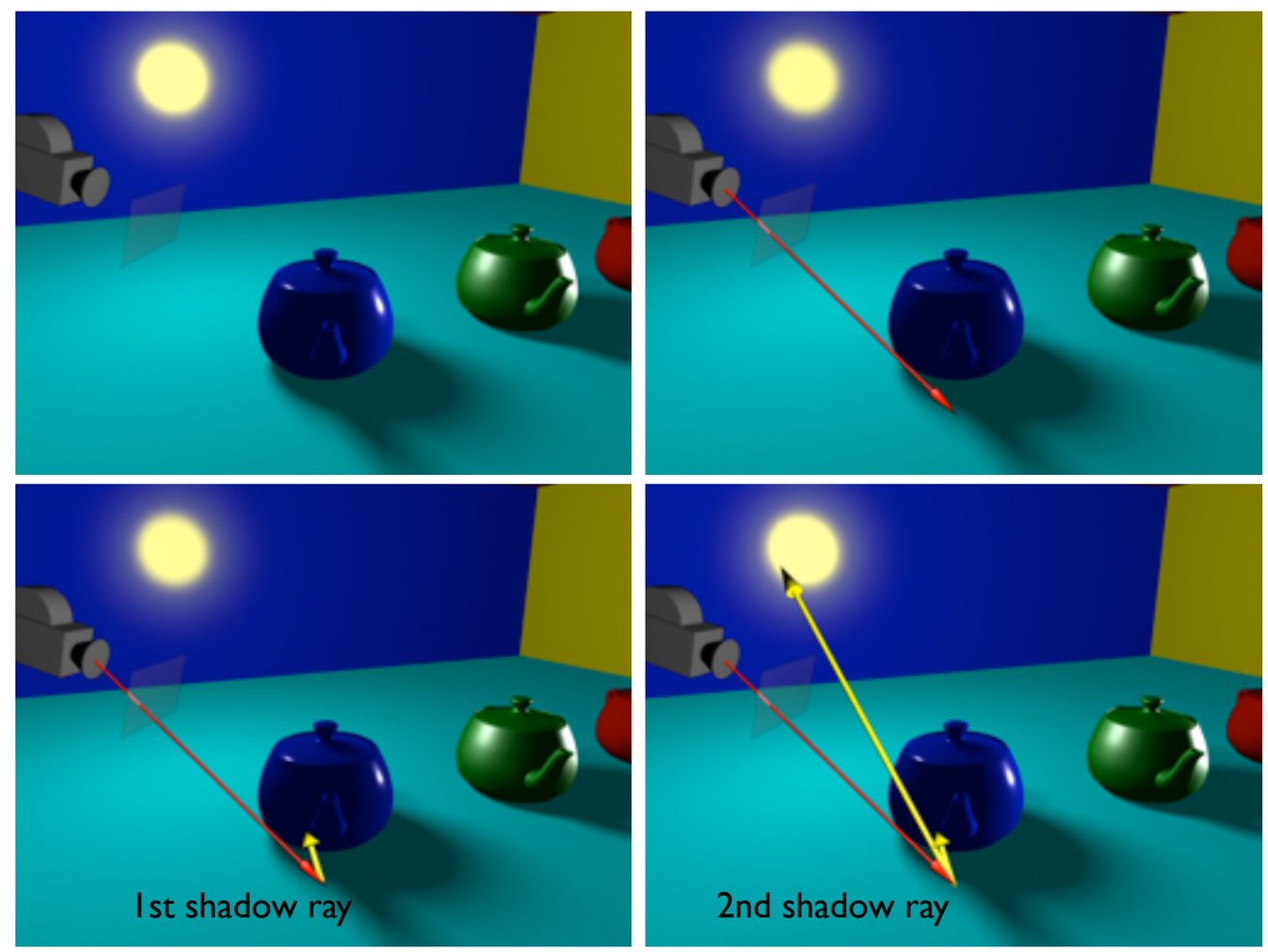






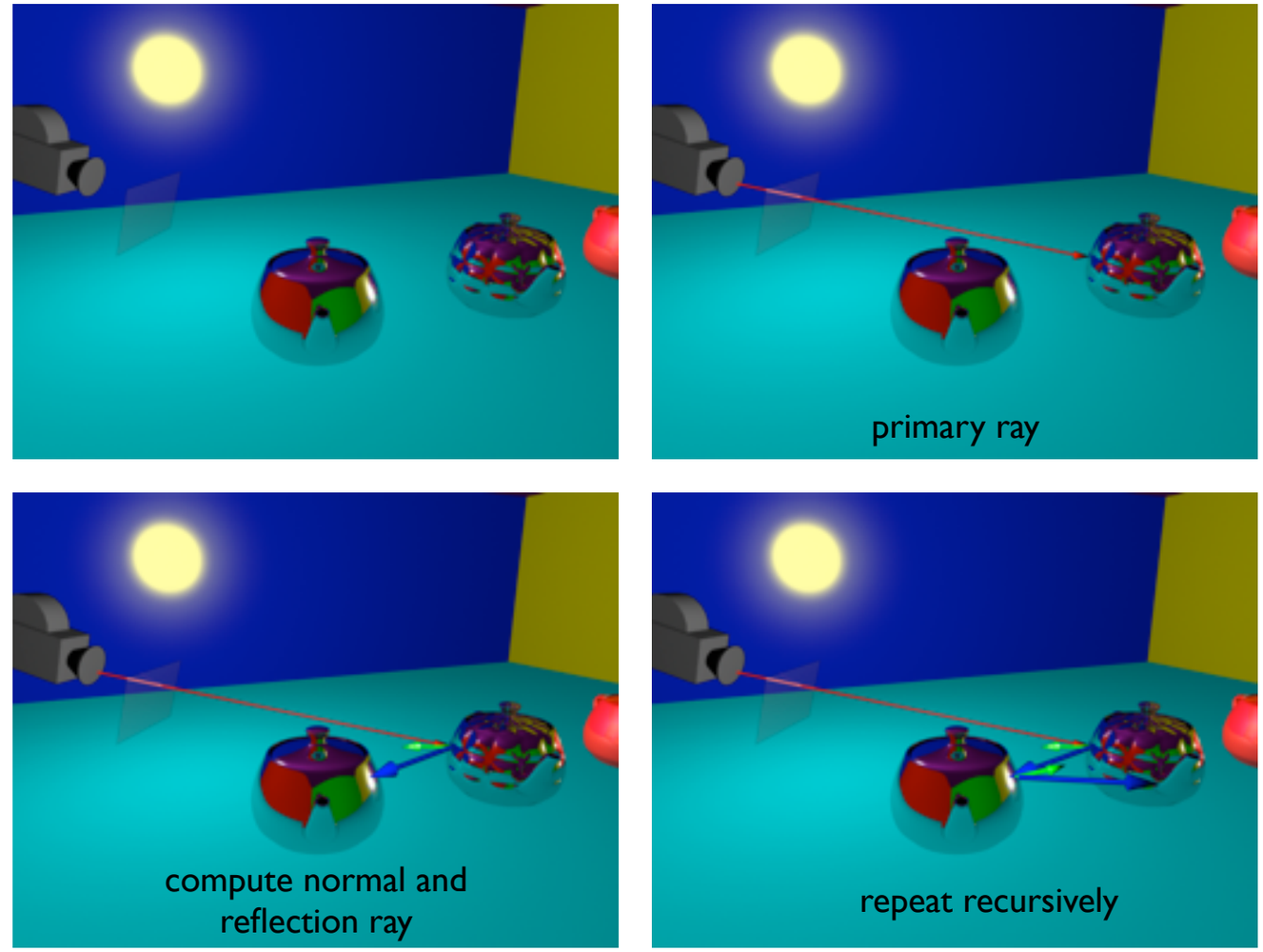
Shadows

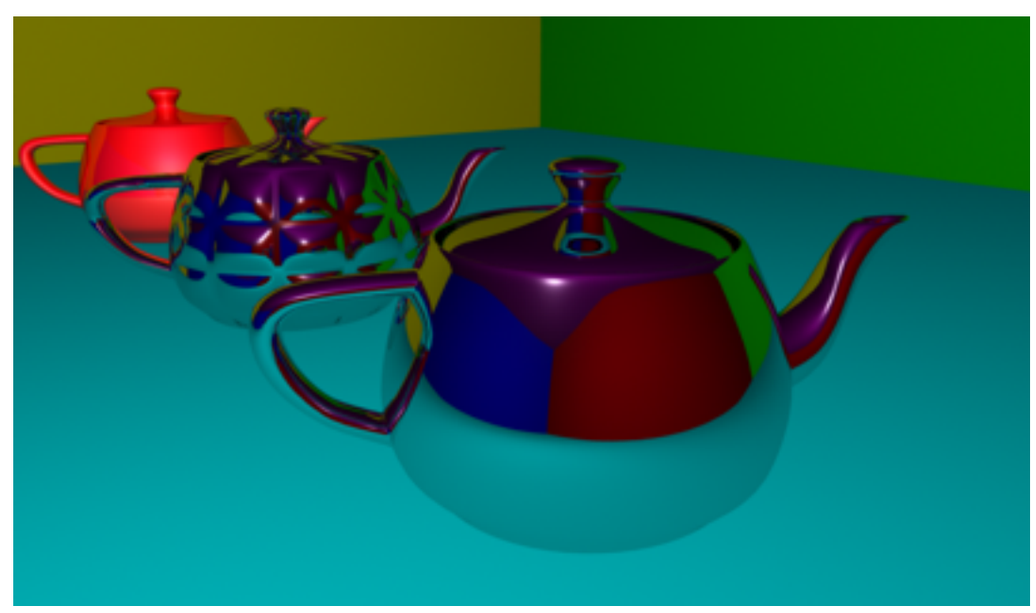
- However, we might also be interested in producing more realistic shadow representations (called soft shadows)
- They are created when objects are lit by area light sources, creating an umbra (core) shadow and a penumbra region
- This can be done by shooting multiple shadow rays in random directions towards area light source
  - Some will lead to occlusions, while other won't
  - Their feedback is accumulated to determine shadows intensity
- Shadow rays from core shadow points will always be intersected
- Shadow rays from points not in shadow, will never be intersected
- Shadow rays from points in soft penumbra region will be intersected more of less often - this is proportional to their their correct shadow intensity

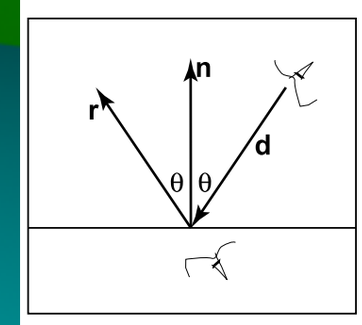


Reflection

- If primary ray is intersected at reflecting point (by definition), then we have to compute a reflected secondary ray (called reflection ray)
- This can be done by determining the normal vector at the intersection point, and then computing the reflected ray at it (i.e., incoming angle = outgoing angle)
- This process can be recursively repeated to compute multi-reflections (i.e., reflections of reflections of reflections of ...)
- Remember that cube mapping / environment mapping is only correct for one single point (center of environment/cube map) for real-time - ray tracing produces correct reflections!

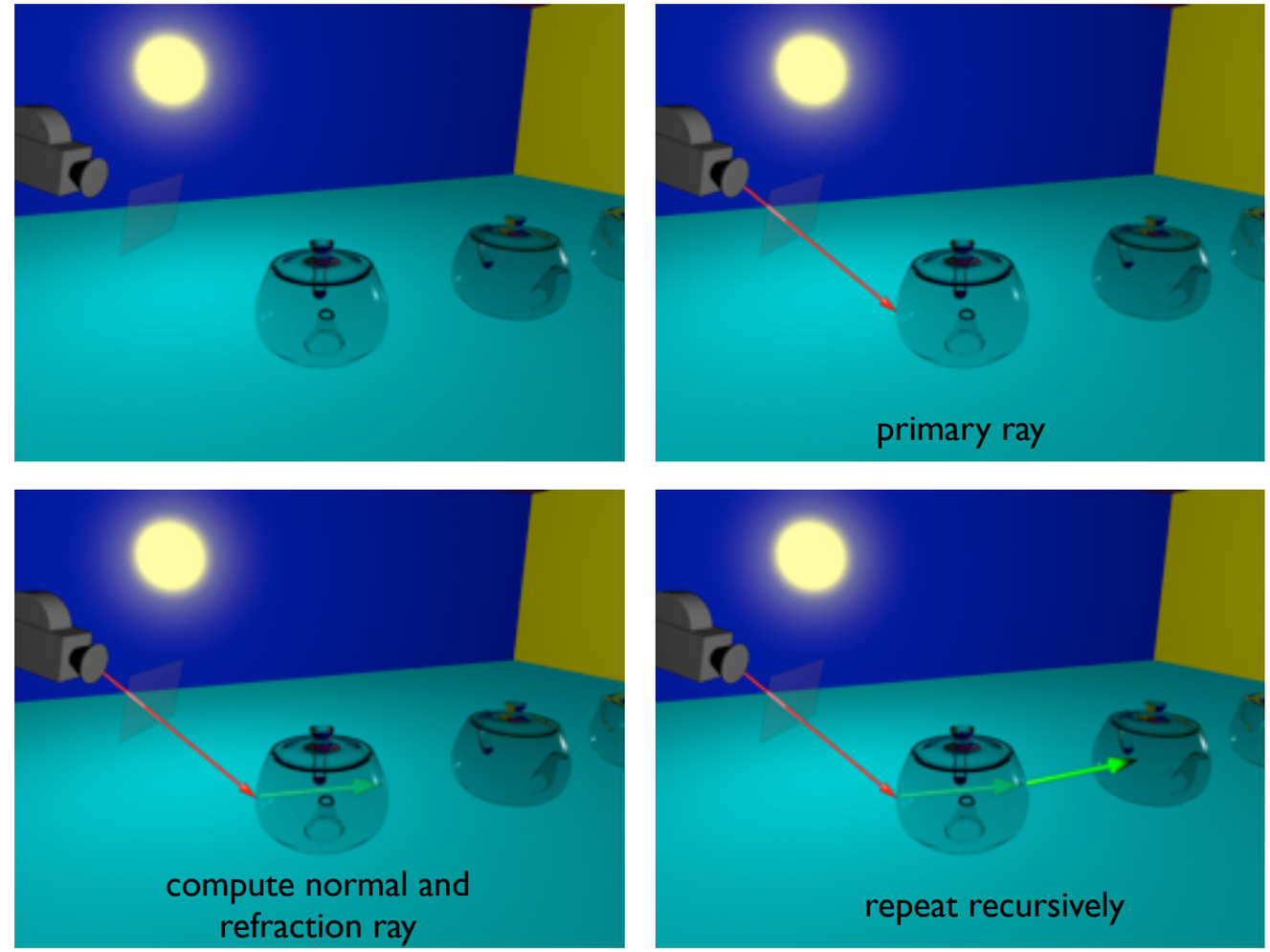


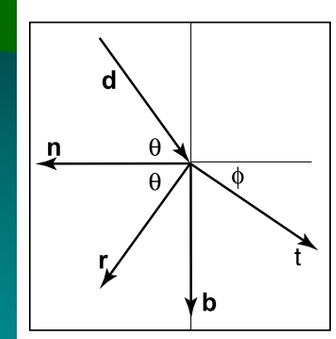




Refraction

- If primary ray is intersected at transparent point (by definition), then we have to compute a refracted secondary ray (called refraction ray)
- This can be done by determining the normal vector at the intersection point, knowing the material's refraction index (Snell's law)
- This process can be recursively repeated to compute multi-refractions (i.e., refraction after refraction after refraction after...)
- Can also consider wavelength (light is refracted / slowed down differently depending on its wavelength) - this leads to chromatic aberration - in this case, refraction rays have to be traced for each color individually





Distribution Ray Tracing

- Shoot multiple, randomly selected rays to calculate final pixel values is called distribution ray tracing

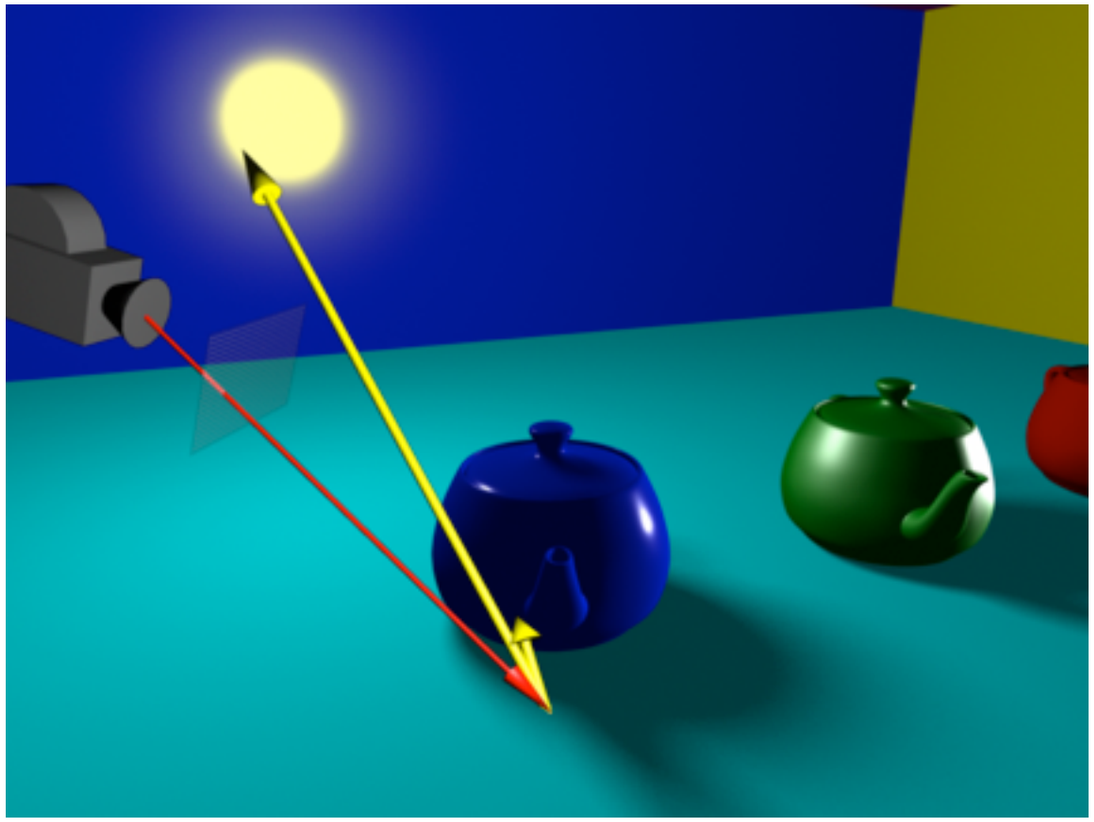
- Distribution ray tracing creates more realistic images with better quality - it supports:

  - Soft shadows via multiple shadow rays towards area light

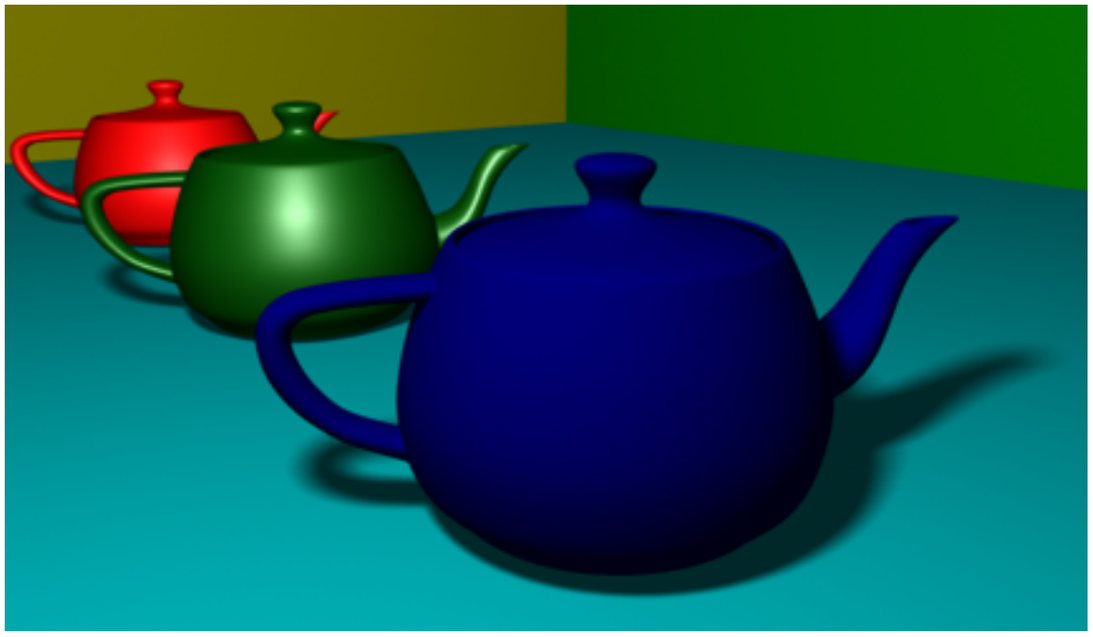
  - Glossy reflections via multiple permuted reflection rays

  - Depth of field effects via shooting multiple viewing rays through focal plane instead of image plane and selecting their origins randomly from small eye area

  - Antialiasing via shooting multiple viewing rays



example soft shadows



Distribution Ray Tracing

- Shoot multiple, randomly selected rays to calculate final pixel values is called distribution ray tracing

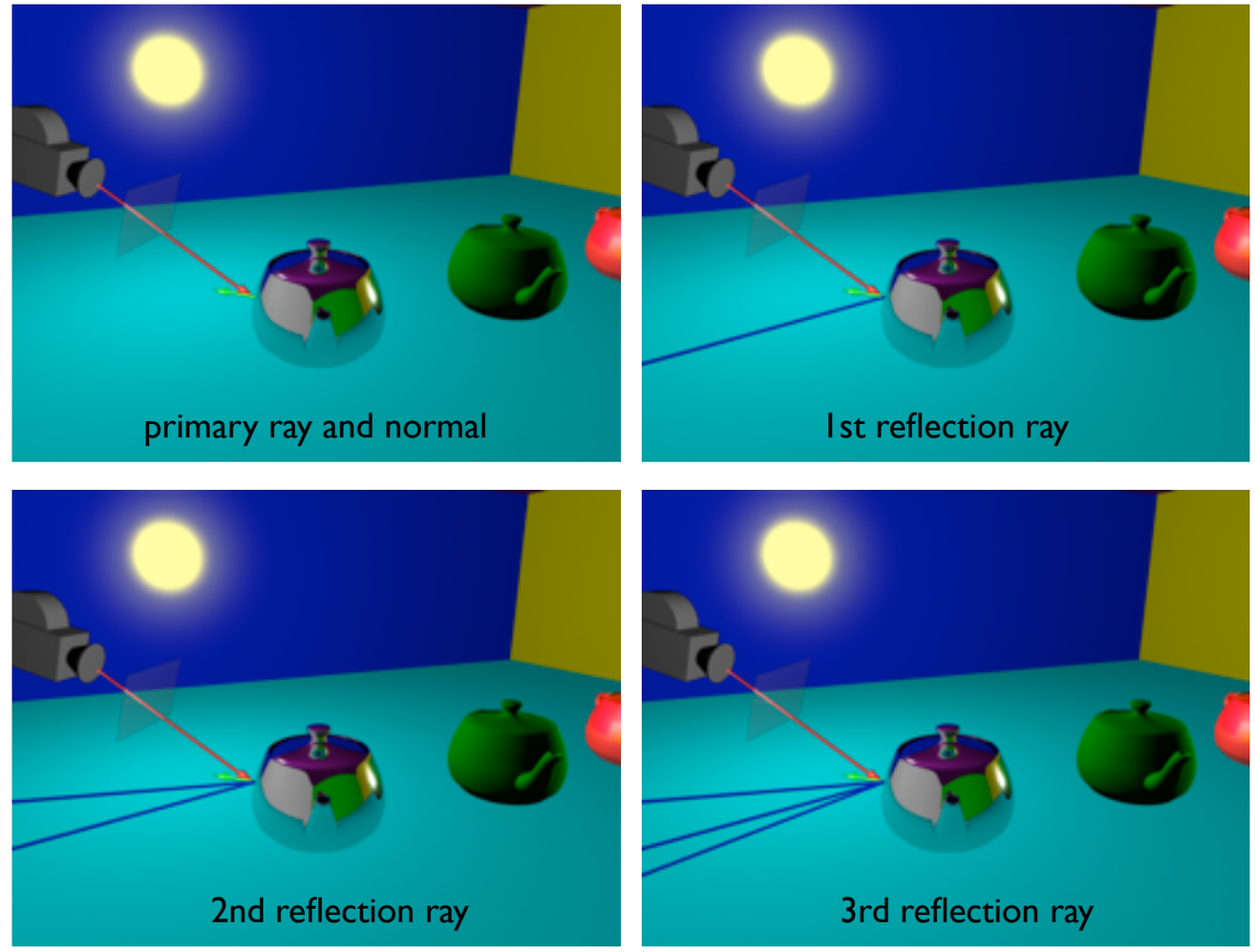
- Distribution ray tracing creates more realistic images with better quality - it supports:

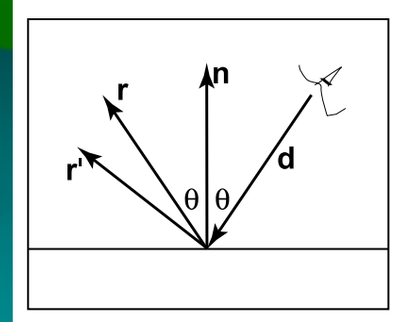
  - Soft shadows via multiple shadow rays towards area light

  - Glossy reflections via multiple permuted reflection rays

  - Depth of field effects via shooting multiple viewing rays through focal plane instead of image plane and selecting their origins randomly from small eye area

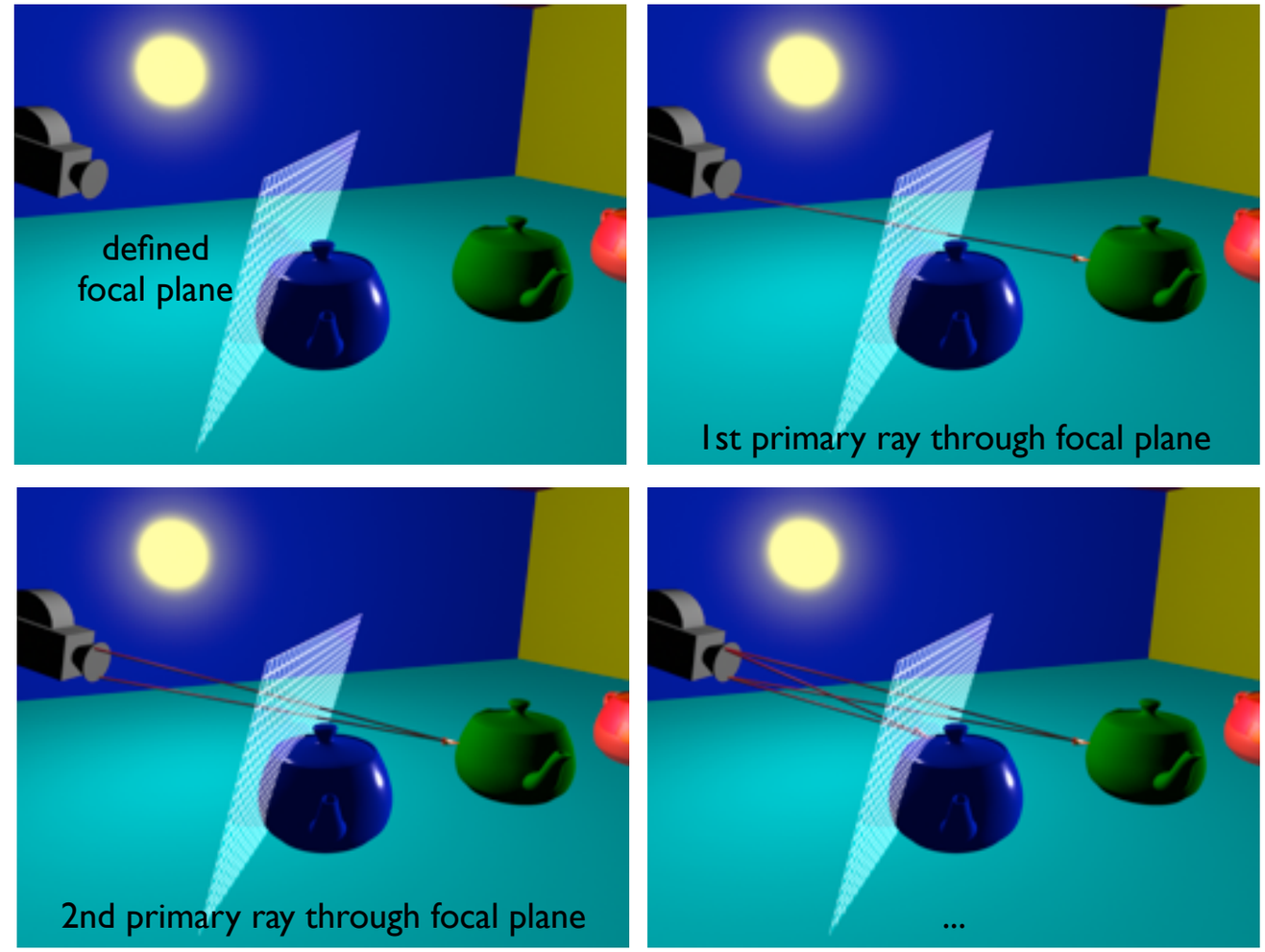
  - Antialiasing via shooting multiple viewing rays

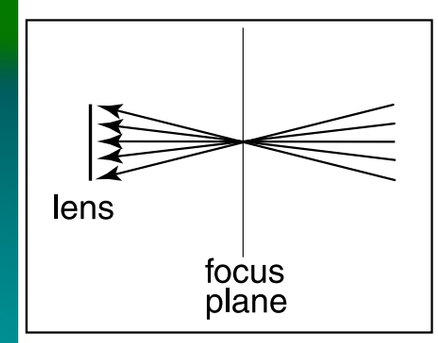




Distribution Ray Tracing

- Shoot multiple, randomly selected rays to calculate final pixel values is called distribution ray tracing
- Distribution ray tracing creates more realistic images with better quality - it supports:
  - Soft shadows via multiple shadow rays towards area light
  - Glossy reflections via multiple permuted reflection rays
  - Depth of field effects via shooting multiple viewing rays through focal plane instead of image plane and selecting their origins randomly from small eye area
  - Antialiasing via shooting multiple viewing rays





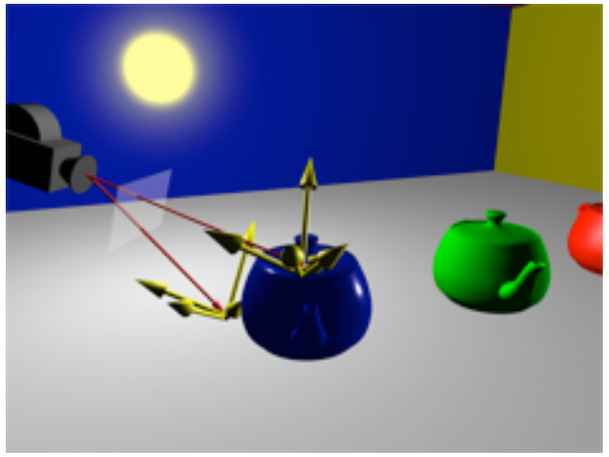
(Bidirectional) Path Tracing

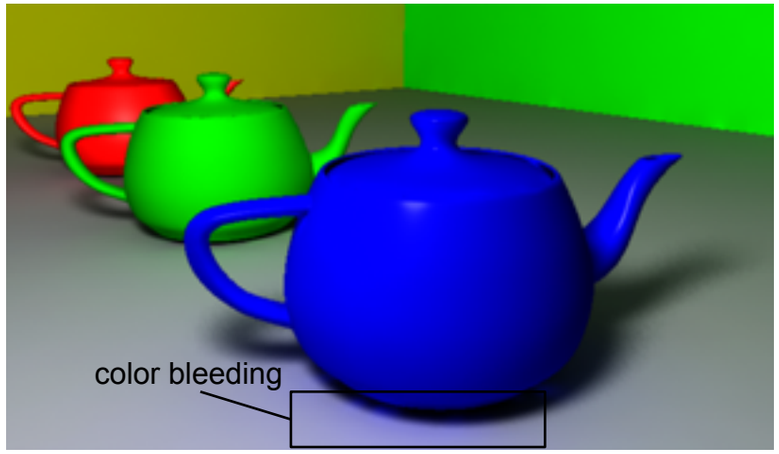
- Generate random secondary rays also at diffuse surfaces to simulate other global illumination effects such as color bleeding and caustics - this is called path tracing

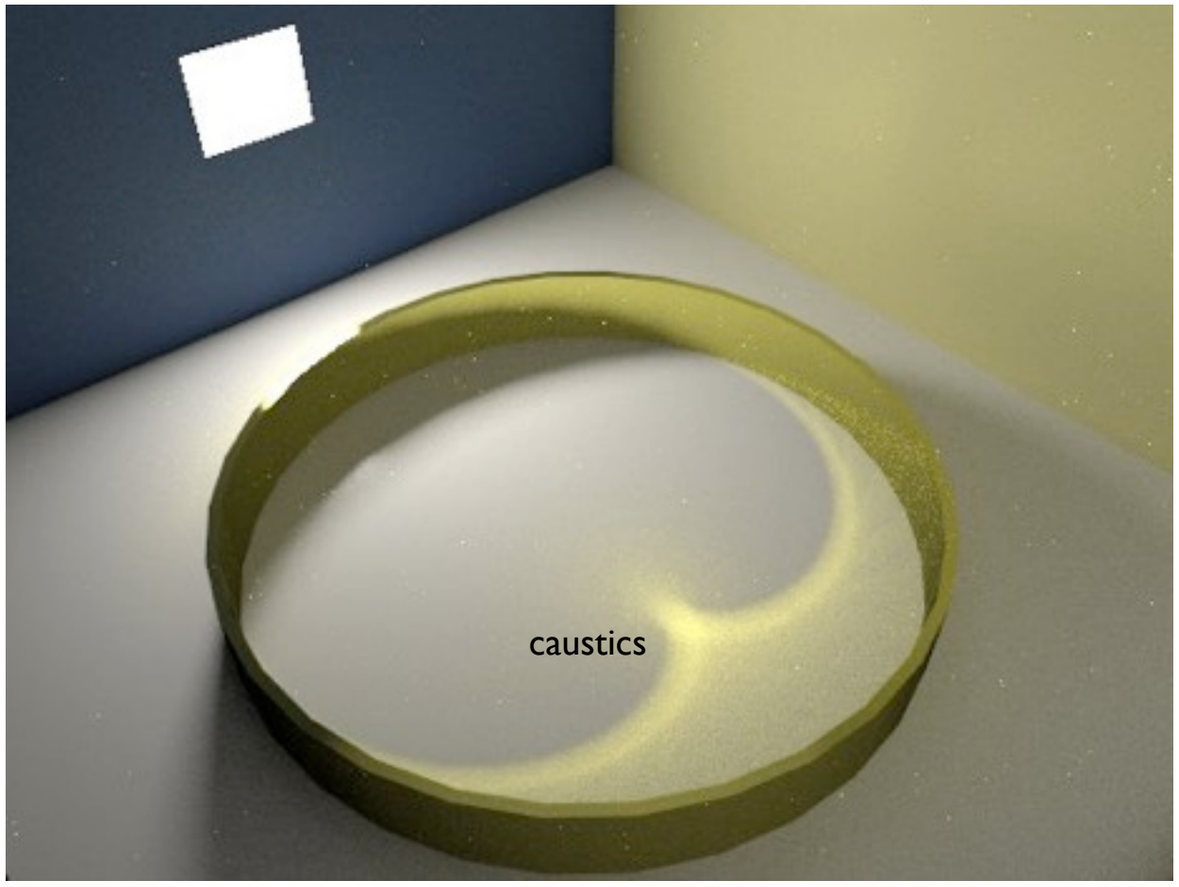
- Bidirectional Path Tracing
  - Shooting multiple rays from the eye as well as from the light source into the scene
  - Faster and more accurate than (ordinary) path tracing
  - Photon Mapping is a similar technique

- In general, ray tracing is not very efficient in generating global effects at diffuse surfaces, like scattering and color bleeding (we need many rays for producing realistic effects)

- Better techniques exist for this, such as radiosity







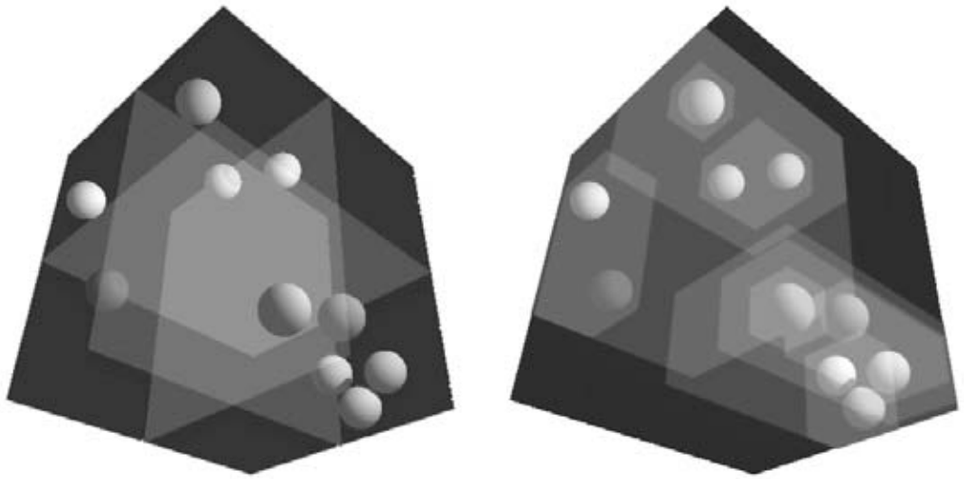
Speeding Up Ray Tracing

- Parallel processing
  - RPUs, specialized hardware (Cell processors, FPGA)
  - Multi-core CPUs
  - PC clusters (render farms)
  - GPUs
- Optimized data structures (eg., reduce intersection tests)
  - Object partitioning (bounding volume hierarchy, BVH)
  - Space partitioning (BSP-trees, Kd-trees)
  - Mixed (bounding interval hierarchy, BIH)

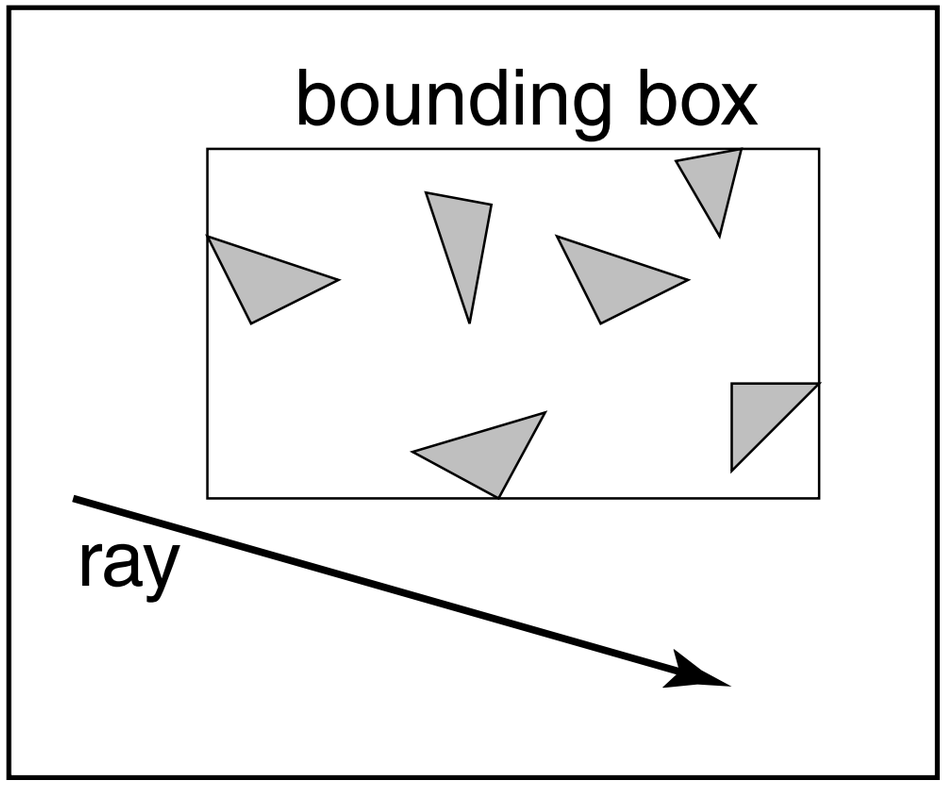
www.OpenRT.com

Sub-Linear Ray-Object Intersection

- What we have discussed so far, requires testing each ray against each triangle
- The complexity is O(N) for N triangles, which is rather inefficient
- To reduce the number of ray-triangle/object intersections, we can either partition the objects (called object partitioning) or the scene space (called space partitioning)
- This leads to a sub-linear complexity (i.e., we do not have to carry out a linear search)
- Enclosing objects (triangle sets) within bounding volumes (like bounding boxes) helps to reduce the intersection tests
- Rays are tested against all bounding boxes first - only the containing triangles are tested if ray intersects corresponding bounding box
- Bounding boxes can be axis-aligned (world-coordinate system) or oriented (aligned with the object boundaries)
- Oriented bounding boxes require additional transformations, but are in general more efficient than axis-aligned bounding boxes



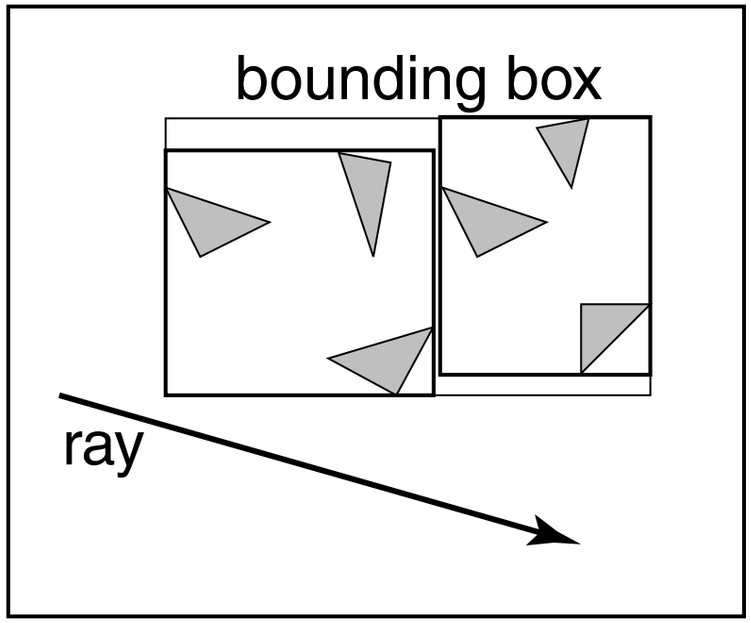
space partitioning vs. object partitioning

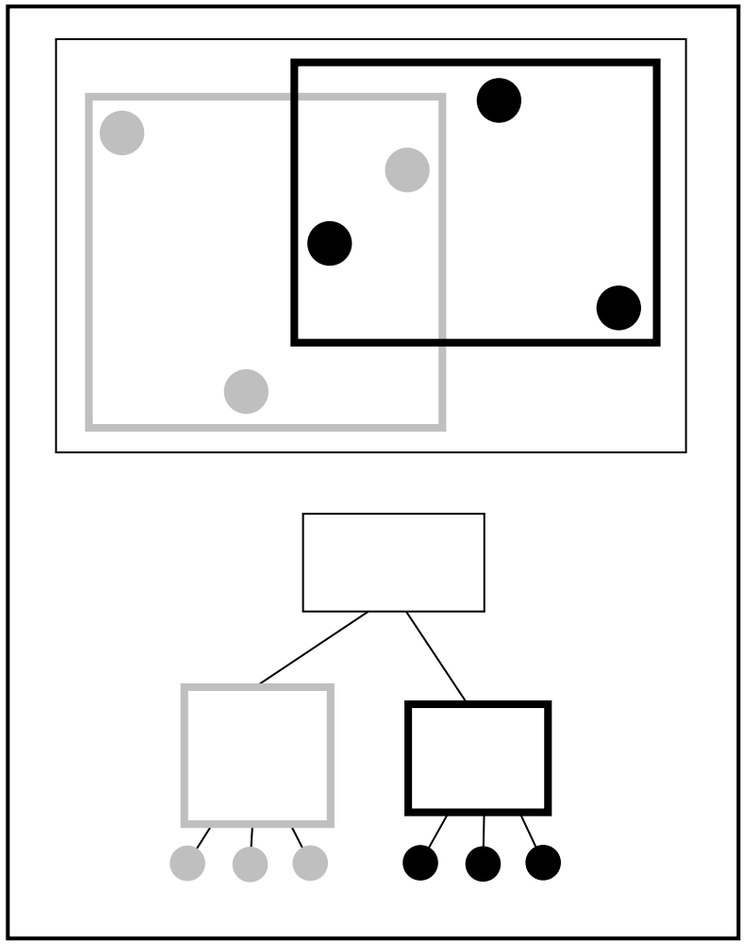


example for bounding box test for which to further tests for containing objects are required

Hierarchical Bounding Boxes

- This is an example for object partitioning
- Axes-aligned 3D bounding boxes are used - but more than one for each object
- Objects (and their bounding boxes) are hierarchically grouped
- This leads to tree structures with larger bounding boxes (encapsulating others) towards root, and smaller (more sub-group or object individual bounding boxes) towards leafs
- Each object is contained by exactly one branch, but multiple branches can cover overlapping scene space (this makes the traversal a bit more complex than traditional binary search)
- HBB trees are usually unbalanced - traversal is fast and building them is easy

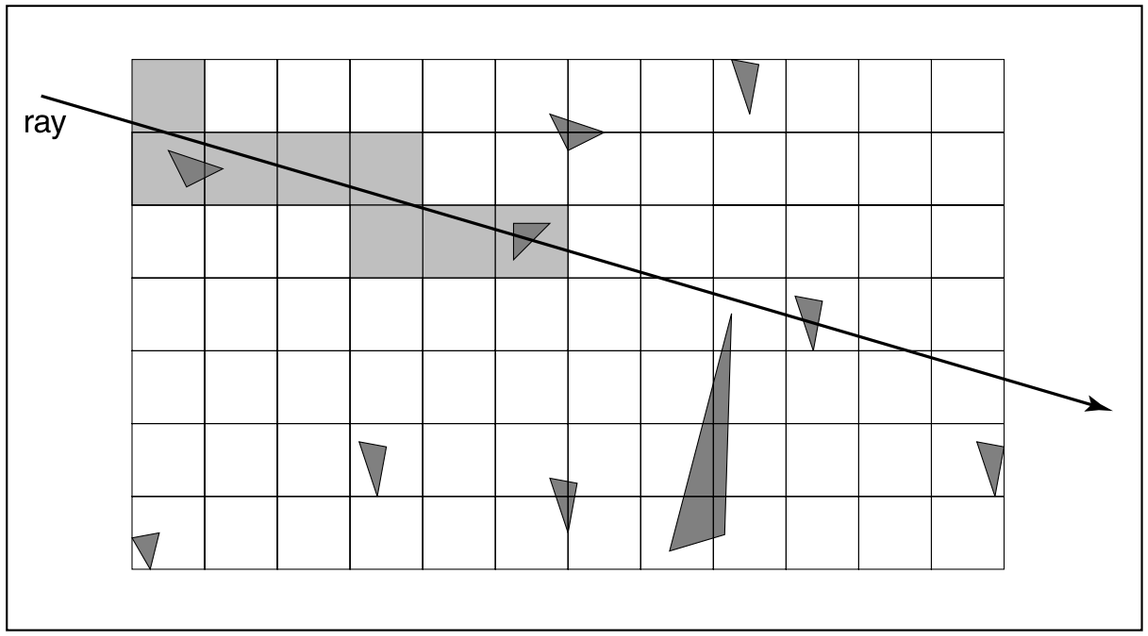




bounding box hierarchies are more efficient than simple per-object bounding boxe structures

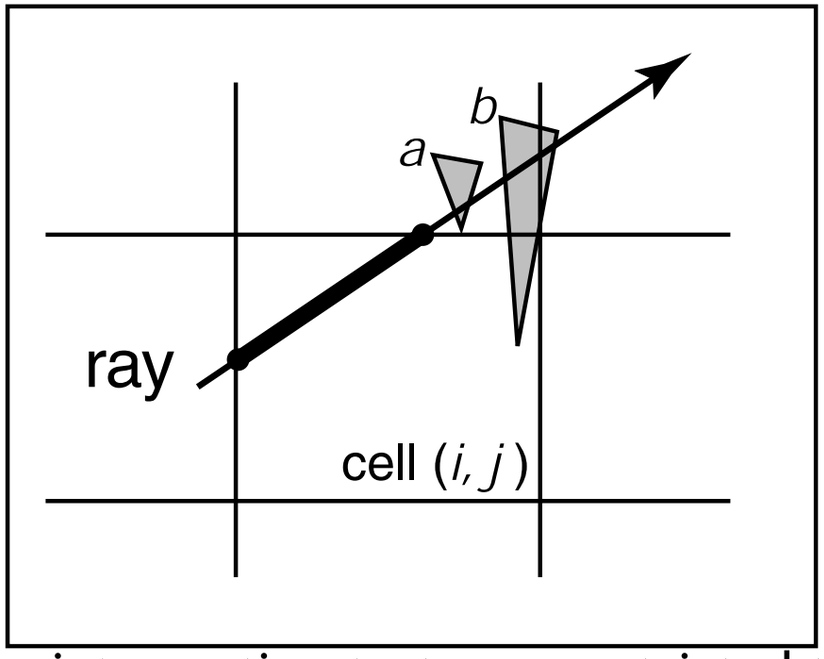
Space Subdivision

- This is an example for space partitioning (can be uniform or non-uniform)
- Uniform subdivision divides space into a uniform grid structure (3D cells, actually)
- Rays are incrementally tested against axis aligned 3D cells before they are tested against containing triangles
- Inside each cell, the line parameter t is restricted to two cell intersection points (this can in general also be done for bounding boxes)
- An example for a non-uniform space subdivision is realized with binary space partitioning (BSP) trees - we have used them for depth ordering (see depth handling and rasterization)
- In this case, the space is recursively divided by axis-aligned planes (in all three directions) and the scene is successively cut into two halves for each plane
- BSP trees are common for ray tracing (especially Kd-trees, K=3)



uniform space subdivision

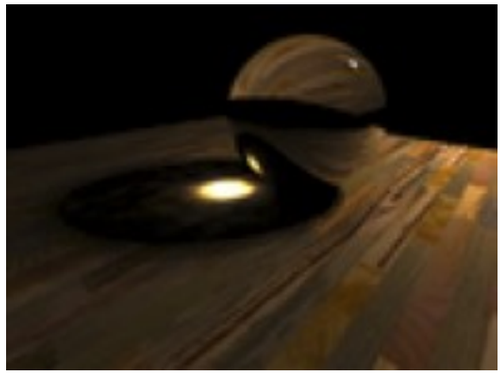
$$p(t) = e + t(s - e)$$



ray intersection tests are restricted to cell boundaries

Notes on GPU Ray Tracing

- Benefit from stream processors hardware (SIMD)
- Problems of a GPU implementation:
  - Efficient usage of parallel execution units complicated
  - Mapping of recursive ray tracing algorithm to constrained graphics hardware
  - Bottleneck: data traversal
  - Real-time data structure update for dynamic scenes required
- Many attempts to outsource individual parts of ray tracing algorithm to GPU (eg. intersection calculations, shadow calculation, shading)
- Better resource control possible with Compute Unified Device Architecture (CUDA) due to higher abstraction - implement GPUs in C (e.g. NVIDIA OptiX)



0.25 fps in 2002

6.1 fps in 2007

46 fps in 2009

NEXT ICG LAB TALK:
10. MAY 2016, 4:00PM

Institute of Computer Graphics

Dr. Bettina Heise

Johannes Kepler University Linz

(Low) Coherent Imaging: Current State, Trends and Perspectives

Science Park 2

Room S2 059

JKU

For more information about our talks visit
http://www.cg.jku.at/talks/invited

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

Thank You!