# EECS 487: Interactive Computer Graphics

Lecture 28:
- Ray Tracing Implementation

## Ray Tracing

Basic tasks:
1. Specify the viewing geometry: eye coordinate frame, image plane, and view frustum
2. For each pixel in the image plane:
   a. build a ray in eye coordinates
   b. transform to world coordinates (why?)
   c. figure out what the ray hits
   d. compute shading, e.g., by using Phong illumination model

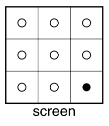
Replaces viewing and projection transforms

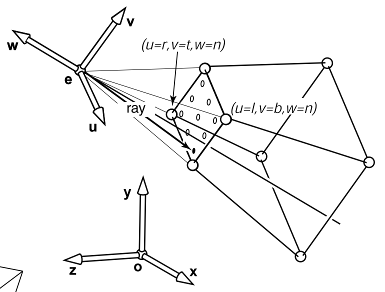
The core of any ray tracing systems, as well as its bottleneck (75% of time spent here)

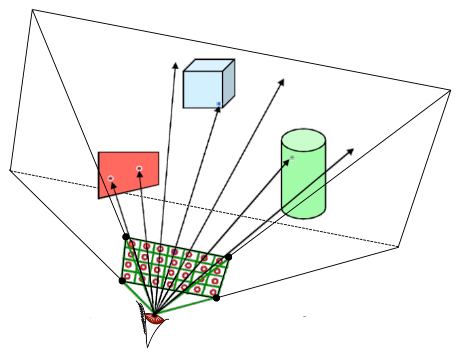
All ray intersection problems boil down to the mathematical process of finding roots

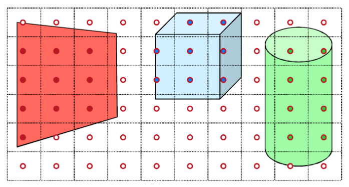
## Image Plane and View Frustum

Let the screen/image plane be at the near plane ($w = n$)





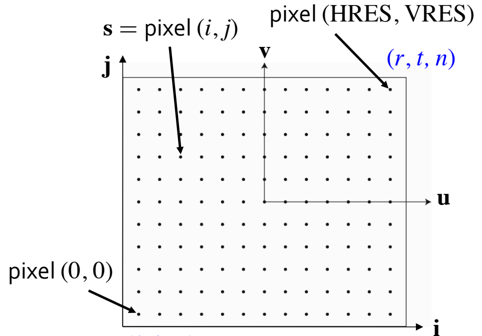




Shirley,Funkhouser

## Screen Pixels in Eye Coordinates

Given $\mathbf{s}$ in image space $(\mathbf{i}, \mathbf{j})$ and $r, l, t, b, n$ in eye space $(\mathbf{u}, \mathbf{v}, \mathbf{w})$, compute $\mathbf{s}$ in eye space: $\mathbf{s}_{eye}$



Let $\mathbf{ll} = (l, b, n)$

$$s_u = l + \frac{(r-l)}{\text{HRES}}(i+0.5)$$

$$s_v = b + \frac{(t-b)}{\text{VRES}}(j+0.5)$$

Assumming symmetric frustum:

$r = -l,\ t = -b$

$$\mathbf{s}_{eye} = \begin{bmatrix} s_u \\ s_v \\ s_w \\ 1 \end{bmatrix} = \begin{bmatrix} l + 2r\dfrac{i+0.5}{\text{HRES}} \\ b + 2t\dfrac{j+0.5}{\text{VRES}} \\ n \\ 1 \end{bmatrix}$$

Zwicker

## Screen Pixels in World Coordinates

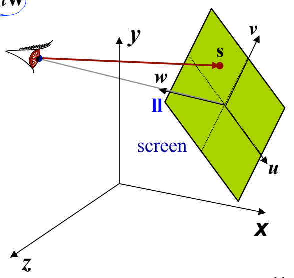
$$\mathbf{s}_{world} = \mathbf{M}_{eye\rightarrow world}\mathbf{s}_{eye}$$

$$= \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{s}_{eye} = \begin{bmatrix} u_x & v_x & w_x & e_x \\ u_y & v_y & w_y & e_y \\ u_z & v_z & w_z & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l + 2r\dfrac{i+0.5}{\text{HRES}} \\ b + 2t\dfrac{j+0.5}{\text{VRES}} \\ n \\ 1 \end{bmatrix}$$

$$\mathbf{s}_{world} = \mathbf{e} + \left(l + 2r\frac{i+0.5}{\text{HRES}}\right)\mathbf{u} + \left(b + 2t\frac{j+0.5}{\text{VRES}}\right)\mathbf{v} + n\mathbf{w}$$

Since $\mathbf{ll} = \mathbf{e} + l\mathbf{u} + b\mathbf{v} + n\mathbf{w}$

$$\mathbf{s}_{world} = \mathbf{ll} + \left(2r\frac{i+0.5}{\text{HRES}}\right)\mathbf{u} + \left(2t\frac{j+0.5}{\text{VRES}}\right)\mathbf{v}$$



Note: no perspective projection matrix, ray generation took care of that!

Merrell

## Building a Ray in Eye Coordinates

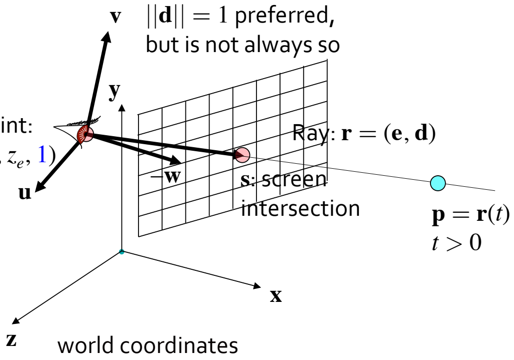
Direction:
$\mathbf{d} = \mathbf{s} - \mathbf{e} = (x_d, y_d, z_d, 0)$
$||\mathbf{d}|| = 1$ preferred, but is not always so

Anchor point:
$\mathbf{e} = (x_e, y_e, z_e, 1)$

Ray: $\mathbf{r} = (\mathbf{e}, \mathbf{d})$

$\mathbf{s}$: screen intersection

$\mathbf{p} = \mathbf{r}(t) = \mathbf{e} + t\,\mathbf{d}$, $t > 0$



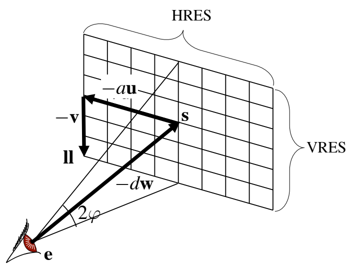
Hart

## Expressed in Aspect Ratio and FoVy

Assuming symmetric frustum:
- focal length: $d = t\,/\tan(\varphi)$, $\varphi = \text{fovy}/2$
- aspect ratio:
  $a = \text{HRES}/\text{VRES} = (r - l)/(t - b)$
  $= 2r/2t \Rightarrow r = at$
  ($r$ and $t$ are unknown)

Then:
- $\mathbf{s}_{world} = \mathbf{ll} + (2\,at\,(i + 0.5)/\text{HRES})\,\mathbf{u} + (2\,t\,(j + 0.5)/\text{VRES})\,\mathbf{v}$
- $\mathbf{ll} = \mathbf{e} - d\mathbf{w} - r\mathbf{u} - t\mathbf{v} = \mathbf{e} - d\mathbf{w} - at\mathbf{u} - t\mathbf{v}$
- dividing by $t$:

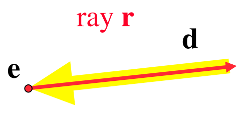
```
llDIVt = eDIVt - 1/tan(phi)w - au - v
for (j = 0; j < VRES; j++) {
  for (i = 0; i < HRES; i++) {
    sDIVt = llDIVt + 2au (double)(i+0.5)/HRES
             + 2v (double)(j+0.5)/VRES;
    color = raytrace(ray(e, sDIVt - eDIVt));
    plot(i,j,color);
  }
}
```



Hart

## Ray-Object Intersection

raytrace($\mathbf{r} = (\mathbf{e}, \mathbf{d})$) returns the intensity of light (e.g., an RGB triple) arriving at the ray anchor at $\mathbf{e}$ in the opposite direction ($-\mathbf{d}$)



raytrace($\mathbf{r}$)

```
color raytrace(ray r, int depth) {
  color c = background;
  if ((hit = intersect(r)) != NULL) {
    hit->depth = depth - 1;
     // shading details simplified,
     // see earlier version
    if (hit->depth > 0) {
      c = raytrace(r, hit->depth);
    }
  }
  return c;
}
```

Hart08

## Ray-Object Intersection

Ray: $\mathbf{r}(t) = \mathbf{e} + t\,\mathbf{d}$

For each ray we must find the nearest intersection point with all objects in the scene

We can define the scene using:
- surface models: plane, triangle, polygon
- solid models: sphere, cylinder

Recall: implicit surfaces: for point $\mathbf{p} = (x, y, z)$ in surface, $f(\mathbf{p}) = 0$

Then ray-surface intersection is when $f(\mathbf{r}(t)) = 0$

Solve for $t$, $\mathbf{r}(t)$ is the intersection point

## Ray-Plane Intersection

Plane implicit equation: $f(\mathbf{p}) = (\mathbf{p} - \mathbf{q})\boldsymbol{\cdot}\mathbf{n} = 0$,
where $\mathbf{n}$ is the plane normal and $\mathbf{q}$ a point in the plane

For $\mathbf{p} = \mathbf{r}(t)$, the ray-plane intersection is at:

$$((\mathbf{e} + t\mathbf{d}) - \mathbf{q})\boldsymbol{\cdot}\mathbf{n} = 0,\ t = (\mathbf{q} - \mathbf{e})\boldsymbol{\cdot}\mathbf{n}/\mathbf{d}\boldsymbol{\cdot}\mathbf{n}$$

Equivalently, for $\mathbf{p} = (x_p, y_p, z_p)$, the implicit plane equation is:

$$Ax_p + By_p + Cz_p + D = 0$$

- the unit normal of the plane: $\mathbf{n} = [A\ \ B\ \ C]^T, A^2 + B^2 + C^2 = 1$
- $D$ is the distance from the coordinate system origin (sign of $D$ determines which side of the plane the origin is at)

Ray-plane intersection is at $t$ such that:

$$A(x_e + t\,x_d) + B(y_e + t\,y_d) + C(z_e + t\,z_d) + D = 0$$

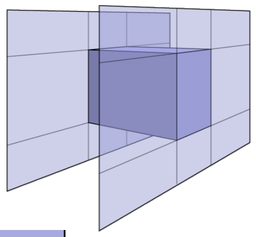
## Ray-Triangle Intersection

Find ray-plane intersection for plane defined by the triangle

If intersection exists:
- compute barycentric coordinates of the intersection point
- if barycentric coordinates are all positive and sum to 1, point is a convex combination of the vertices of the triangle and is inside triangle
- otherwise outside
- (Möller&Trumbore algorithm does it in 1 div, 27 muls, 17 adds)

## Ray-Box Intersections

Could intersect with 6 faces individually: $O(n^2)$
Better: box is the intersection of 3 slabs $O(n)$
$n$: number of comparisons
2D example (similarly for 3D):

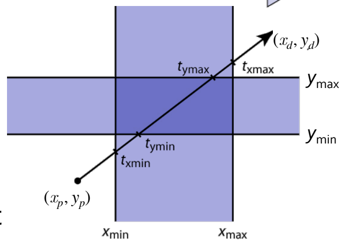


$$x_{min} = x_p + t_{x_{min}}\,x_d$$
$$t_{x_{min}} = (x_{min} - x_p)/x_d$$

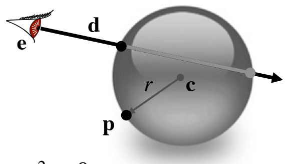
$$y_{min} = y_p + t_{y_{min}}\,y_d$$
$$t_{y_{min}} = (y_{min} - y_p)/y_d$$

$$t_{min} = \max(t_{x_{min}}, t_{y_{min}})$$
$$t_{max} = \min(t_{x_{max}}, t_{y_{max}})$$

if $t_{min} > t_{max}$, box is missed
if $t_{max} < 0$, box is behind eye
else $\mathbf{r}(t_{min})$ is intersection point



## Ray-Sphere Intersections



A sphere is defined by the sphere center $\mathbf{c} = (x_c, y_c, z_c)$, and radius $r$

Sphere implicit function: $||\mathbf{p} - \mathbf{c}|| = r$

$$(x_p - x_c)^2 + (y_p - y_c)^2 + (z_p - z_c)^2 - r^2 = 0$$

Ray-sphere intersection:

$$((x_e + tx_d) - x_c)^2 + ((y_e + ty_d) - y_c)^2 + ((z_e + tz_d) - z_c)^2 - r^2 = 0$$

Multiplying out and simplifying:

$$0 = (x_d^2 + y_d^2 + z_d^2)\, t^2 + 2(x_d(x_e - x_c) + y_d(y_e - y_c) + z_d(z_e - z_c))\, t$$
$$+ (x_e - x_c)^2 + (y_e - y_c)^2 + (z_e - z_c)^2 - r^2$$
$$0 = At^2 + Bt + C$$

If **d** is normalized, $A = x_d^2 + y_d^2 + z_d^2 = 1$

The solutions for $t$ can be found using the quadratic equation: $t = \dfrac{-B \pm \sqrt{B^2 - 4C}}{2}$

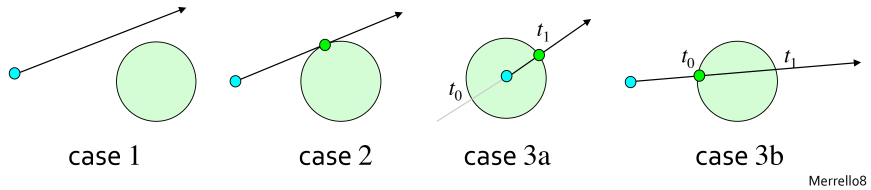
## Ray-Sphere Intersections

$$\boxed{t = \frac{-B \pm \sqrt{B^2 - 4C}}{2}}$$

$B^2 - 4C$ is the discriminant

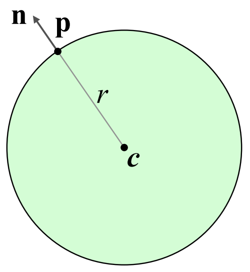
Three possibilities:

1. $B^2 - 4C < 0$
   - no real roots, sphere was missed, no intersection ⇒ always check the discriminant first
2. $B^2 - 4C = 0$
   - one real root, ray “grazes” the sphere, $t_0 = t_1 = -B/2$
3. $B^2 - 4C > 0$
   - two real roots
     a. $t_0 < 0, t_1 > 0$
        negative values of $t$ indicate that the ray started in the sphere ⇒ only positive roots are valid
     b. $0 < t_0 < t_1$
        the smaller root is closer to the ray’s starting point, **e** ⇒ save time by computing the small root first



## Ray-Sphere Intersections

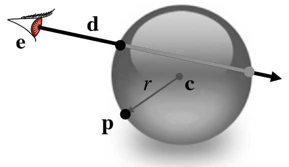
Computation time per ray-sphere test:
- 17 additions / subtractions
- 17 multiplies
- 1 square root



Computing normal: the normal **n** at an intersection point **p** on a sphere is the same as the coordinates of **p** in the sphere’s frame of reference:

$$\mathbf{n} = \frac{\mathbf{p} - \mathbf{c}}{\|\mathbf{p} - \mathbf{c}\|} = \frac{\mathbf{p} - \mathbf{c}}{r} = \left( \frac{x_p - x_c}{r} \quad \frac{y_p - y_c}{r} \quad \frac{z_p - z_c}{r} \right)^T$$

## Ray-Sphere Intersections



A sphere is defined by the sphere center $\mathbf{c} = (x_c, y_c, z_c)$, and radius: $r$

Sphere implicit function: $||\mathbf{p} - \mathbf{c}|| = r$

$$(\mathbf{p} - \mathbf{c})\cdot(\mathbf{p} - \mathbf{c}) - r^2 = 0$$

Ray-sphere intersection:

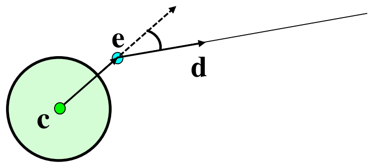
$$((\mathbf{e} + t\mathbf{d}) - \mathbf{c})\cdot((\mathbf{e} + t\mathbf{d}) - \mathbf{c}) - r^2 = 0$$

Multiplying out and simplifying:

$$0 = (\mathbf{d}\cdot\mathbf{d})\, t^2 + 2((\mathbf{e} - \mathbf{c})\cdot\mathbf{d})\, t + (\mathbf{e} - \mathbf{c})\cdot(\mathbf{e} - \mathbf{c}) - r^2$$
$$0 = t^2 + Bt + C$$

if **d** is normalized, $||\mathbf{d}|| = 1$

Acceptance/rejection tests:
- $(\mathbf{e} - \mathbf{c})\cdot\mathbf{d} > 0?$
- $(\mathbf{e} - \mathbf{c})\cdot(\mathbf{e} - \mathbf{c}) - r^2 < 0?$



## Ray-Sphere Intersections–Geometric

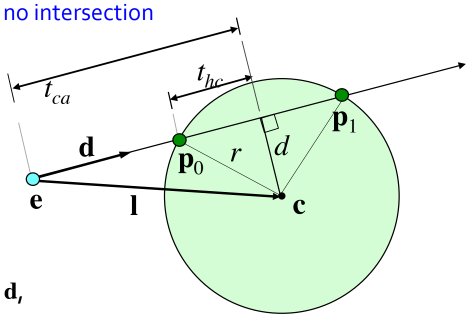
Let $\mathbf{l} = \mathbf{c} - \mathbf{e}$, $l^2 = \mathbf{l}\bullet\mathbf{l}$
if $(l^2 < r^2)$ **e** is inside of sphere

$t_{ca} = \mathbf{l} \bullet \mathbf{d}$ // **d** normalized, projection of **l** on **d**
if $(t_{ca} < 0$ and **e** is outside of sphere)
ray pointing away from sphere, no intersection

$d^2 = l^2 - t_{ca}^2$
if $(d^2 > r^2)$ ray misses sphere

$t_{hc} = \sqrt{(r^2 - d^2)}$
if (**e** is outside sphere)
intersection is at $t_0 = t_{ca} - t_{hc}$
else
intersection is at $t_1 = t_{ca} + t_{hc}$
// to compute **p** with unnormalized **d**,
// normalize **d** first: use $t/||\mathbf{d}||$

Worst case computation reduced by 4 mults and 1 add



Funkhouser09

## Ellipsoid Intersection

We have an optimized ray-sphere test
- but we want to ray trace an ellipsoid…

Let **M** be a 4×4 transformation matrix that distorts a sphere ($f()$) into an ellipsoid

For **p** on ellipsoid, $f(\mathbf{M}^{-1}\mathbf{p}) = 0$

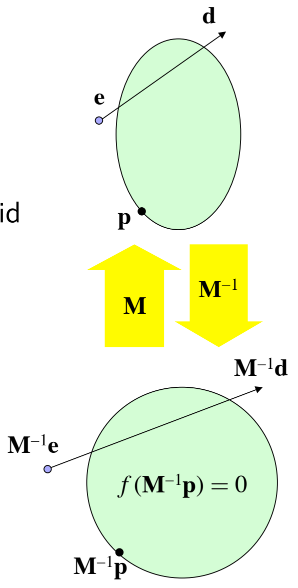
$$f(\mathbf{M}^{-1}\mathbf{r}(t)) = f(\mathbf{M}^{-1}(\mathbf{e} + t\,\mathbf{d}))$$
$$= f(\mathbf{M}^{-1}\mathbf{e} + t\,\mathbf{M}^{-1}\mathbf{d})$$

Intersection point must be in world coordinates
- $t$ is the same in both cases

$$\mathbf{p} = \mathbf{e} + t\,\mathbf{d}$$

Don't forget to transform the normal:

$$\mathbf{n}_{ellipsoid} = (\mathbf{M}^{-1})^T \mathbf{n}_{sphere}$$



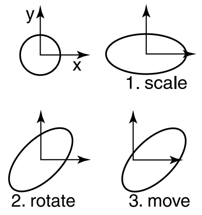
$\mathbf{d}$
$\mathbf{e}$
$\mathbf{p}$
$\mathbf{M}$
$\mathbf{M}^{-1}$
$\mathbf{M}^{-1}\mathbf{d}$
$\mathbf{M}^{-1}\mathbf{e}$
$f(\mathbf{M}^{-1}\mathbf{p}) = 0$
$\mathbf{M}^{-1}\mathbf{p}$

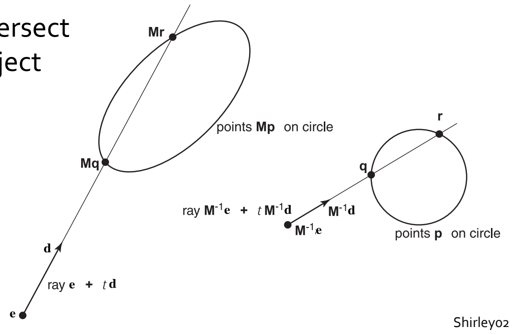
Hart

## Intersection

In general, for an object that is to be transformed by matrix **M**, ray intersection may be easier done on original object, before the transformation

- apply $\mathbf{M}^{-1}$ to ray and intersect objects in their local (object coordinates





Mr
points Mp on circle
Mq
r
q
ray $\mathbf{M}^{-1}\mathbf{e}$ + $t\,\mathbf{M}^{-1}\mathbf{d}$
$\mathbf{M}^{-1}\mathbf{d}$
$\mathbf{M}^{-1}\mathbf{e}$
points p on circle
d
ray e + $t$ d
e

Shirley02

## Instancing

Object stored in untransformed state along with the transformation matrix

The transformation of the object is delayed until instantiation/rendering time

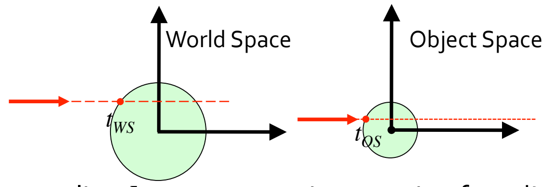
Bonus: re-use objects without replicating them in memory!

Shirley02

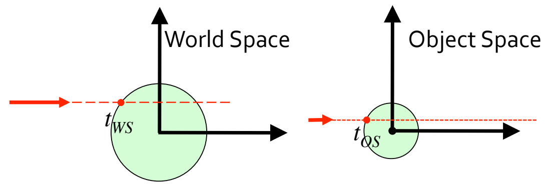
## Caveat for Instantiation with Scaling

If $\mathbf{M}$ includes scaling, don't re-normalize $\mathbf{d}$: you'll get the right $t$ when inverse transforming intersection ($\mathbf{M}^{-1}t\mathbf{d}$)

- if you re-normalize $\mathbf{d}$, $t_{OS} \neq t_{WS}$ and must be rescaled after inverse transform



- if you don't re-normalize $\mathbf{d}$, $t_{OS} = t_{WS}$ ⇒ intersection found!
  - but don't rely on $t_{OS}$ being true distance during intersection routines



Durand08

## Rules of Thumb for Intersection Testing

Perform acceptance and rejection test
- try them early on to make a fast exit

Postpone expensive calculations if possible

Use dimension reduction
- e.g., 3 1D tests instead of one complex 3D test, or 2D instead of 3D

Share computations between objects if possible

Use instancing, delay transformation

Akenine-Möller03

## Accelerating Intersection Tests

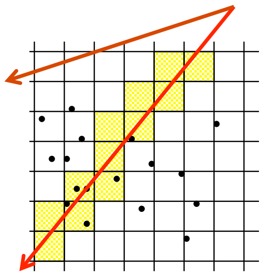
Find the first-hit object

Simplest linear approach: $\mathrm{O}(N_{\text{pixels}} * M_{\text{objects}})$

Acceleration techniques (sublinear in $M_{\text{objects}}$): use spatial data structure to reduce the number of tests needed
- spatial subdivision: space partitioning
- object subdivision: hierarchical bounding volumes

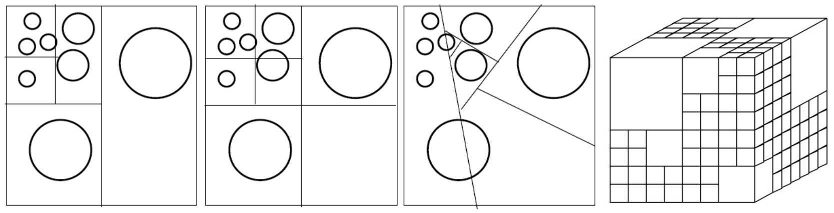
Probably the single most important efficiency improvement
- others include shadow caching: start shadow intersection search with the last object intersected

## Spatial Subdivision



Divide up space and record what objects are in each cell
- store objects in a 3D array
- trace ray through voxel array

For example: uniform grid, quadtree/octree, BSP tree, $k$d-tree (most popular, $k$-dimensional, axis-aligned BSP)

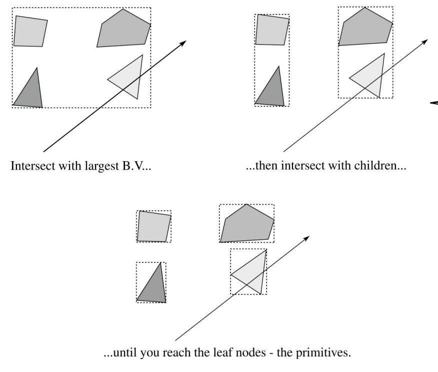


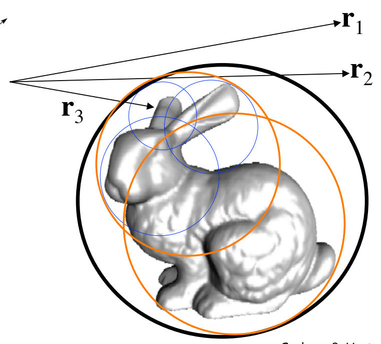
Hanrahan09, Curless08

## Hierarchical Bounding Volumes

Arrange scene into a tree

- internal nodes consist of primitives with very simple intersection tests (boxes or spheres)
- each internal node's volume contain all objects in subtree
- leaf nodes contain the original geometry


Intersect with largest B.V...
...then intersect with children...
...until you reach the leaf nodes - the primitives.



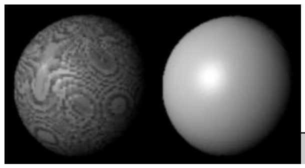
Curless08, Hart08

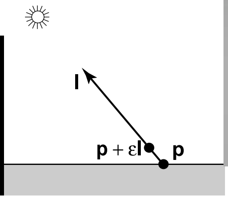
## Precision Problems

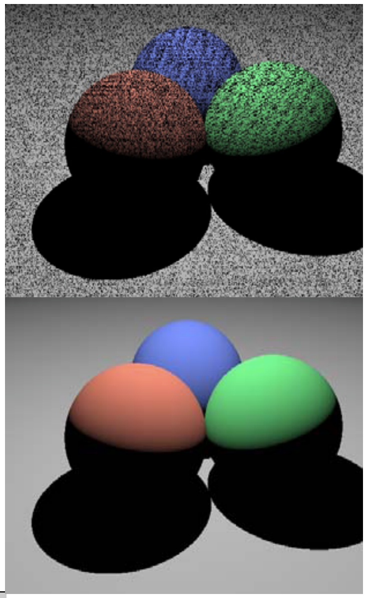
Numerical inaccuracy may cause intersection to be below surface

⇒ causing surface to incorrectly shadow itself

Move a little along surface normal before shooting shadow ray, or move a little along shadow ray to compute intersection







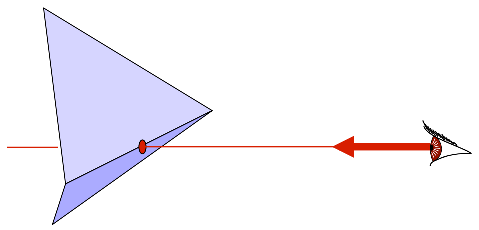
O'Brien08, Shirley02, James07

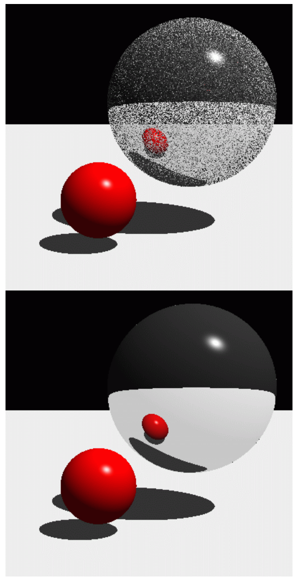
## Precision Problems

Also when computing reflection ray

And when computing intersection with edges in triangle meshes

⇒ must report intersection





Durand08

## Transmission Ray Exit Caveats

To compute Fresnel reflectance coefficient when exiting an object, remember to invert the normal and transmission ray

Similarly for the computation of the refracted ray exiting the object

In computing ray-object intersection at the exit point, be sure to translate by $\varepsilon$ in the right direction

## Ray Tracing vs. Pipelined Rasterization

Ray Tracing
- ray-centric
- needs to store scene in memory
- (mostly) random access to scene

Pipelined Rasterization
- triangle centric
- needs to store image (and depth) in memory
- (mostly) random access to frame buffer

Which requires less memory? Scene or frame buffer?
frame buffer
Which image is easiest to access randomly?
frame buffer due to regular sampling

Durand

## Interactive RayTracing

Advantages of ray tracing:
- can handle very complex scenes relatively easily
  - sublinear complexity with acceleration (hierarchical bbox), need not process all triangles in scene
- provide complex materials and shading for free
- easy (but expensive) to add global illumination, specularities, etc.

But ray tracing is historically slow because
- hard to access data in memory-coherent way, cannot take advantage of incremental computation
- requires many samples for complex lighting and materials

Ramamoorthi

## Interactive Raytracing

Leverage power of modern CPUs: develop cache-aware, parallel implementations

Modern GPUs have general streaming architecture: can map various elements of ray tracing kernels like eye rays, intersection tests, etc. into vertex or fragment programs

Search youtube for “nvidia OptiX” and “realtime ray tracing”

hothardware.com/News/NVIDIA-Shows-Interactive-Ray-Tracing-on-GPUs/

Ramamoorthio8