Prof. Reuven Aviv
Department of Computer Science
Tel Hai Academic College

Computer Graphics

Introduction to Ray Tracing

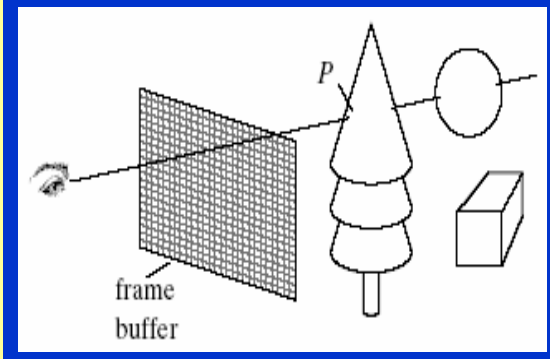
Slides adapted from F. Hill, S. Kelley Computer Graphics

Overview of ray tracing

What is written to pixels in the frame buffer?

- frame buffer: an array of pixels positioned in space
- the eye looking through it into the scene
- For each pixel in the frame buffer we ask:
  - What does the eye see through this pixel?

- a ray of light arriving at the eye through this pixel from some point *P* in the scene.
- The color of the pixel is the color of the light from P along this ray



Simple Ray Tracing

- follow a "ray" from the eye through a given pixel (r, c)
- Identify ray intersections with each object in scene
- the first surface hit by the ray is the closest object to the eye
  - more remote surfaces are ignored (for now)
  - Thus hidden surfaces automatically removed
- apply shading model to compute light from first hit point P_h
  - ambient, diffuse, and specular components of the light
- Resulting color is then written to frame buffer at the pixel

Advanced Ray Tracing

- Trace the ray through the scene, identify paths created by reflection and refraction. Create light tree
  - root at the pixel, leaves are faces and light sources
  - Internal nodes are intermediate surfaces emanating light
  - light at pixel combined of all lights along tree paths
  - Shadows and transparancies created
- Ray tracing can be applied not only to polygon meshes
  - Also to Constructive Solid Geometry (CSG) Models

compound objects Models

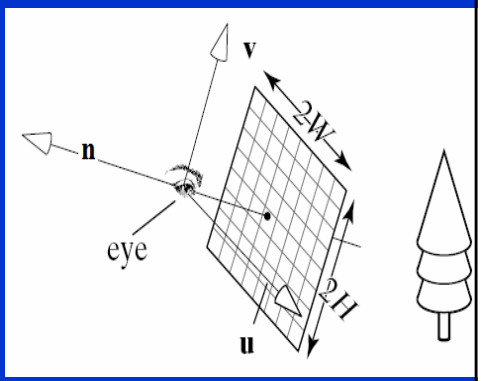
- Solid objects constructed of standard primitive objects
  - E.g. spheres, cubes, cylinders
  - Primitives transformed to alter size and orientation
  - Then combined by (e.g.) boolean operations
  - Union, intersection, subtraction (holes)…
- more compound objects are constructed by same operations
- Ray tracing applied to compound objects actually applied to the primitives
- Intersections and light tree construction is feasible

Mechanics of Ray tracing

eye and window

- **Eye** (camera) at point **eye**.
- Define 3 vectors **u**, **v**, and **n**.
- Near plane orthogonal to **n**, at distance *N* in front of the eye
- frame buffer in the near plane. Center at (0, 0, -N)

- two ways:
- Frame buffer
  - Nc columns, Nr rows
- window
  - width 2W, height 2H



a ray from eye through pixel P_{rc}

- Pixel at column *c*, row *r* is located at point $\mathbf{P}_{rc}$
- $\mathbf{P}_{rc} = (u_c, v_r, -N)$ (location in the **u**, **v**, **n** coordinate frame)
  - $u_c = W(2c/N_C - 1)$; $v_r = H(2r/N_R - 1)$ (easy to show)
- direction of ray from origin (eye) to P_{rc}
  - $\mathbf{dir}_{rc} = u_c\mathbf{u} + v_r\mathbf{v} - N\mathbf{n}$
- Parametric representation of ray from **eye** in direction $\mathbf{dir}_{rc}$
  - $\mathbf{r}_{rc}(t) = \mathbf{eye} + \mathbf{dir}_{rc}\, t$

Pseudocode for Ray Tracing

```
<define objects, light sources and camera in the scene >
for (int r = 0; r < NR; r++)
  for (int c = 0; c < NC; c++)
  { < 1. Build the rc-th ray >
    < 2. Find all intersections of rc-th ray with objects >
    < 3. Sort intersections. Find intersection that lies closest to
    and in front of the eye >
    < 4. Compute the hit point where the ray hits this object,
    and the normal vector at that point >
    < 5. Find the color of the light returning to the eye along the
    ray from the point of intersection >
    < 6. write the color in the rc-th pixel into frame buffer > }
```

Intersections

Intersection with generic objects

- *generic sphere* (at the origin, radius1) : $x^2 + y^2 + z^2 = 1$
  - $F(x, y, z) \equiv x^2 + y^2 + z^2 - 1$ → F(x, y, z) = 0
- generic objects have "standard" location, size, orientation
  - With known implicit equation F(x, y, z) = 0
- to find intersection of *general ray* $\mathbf{r}(t) = \mathbf{S} + \underline{\mathbf{d}}t$ with objects:
  - $F(S_x+d_xt, S_y + d_yt, S_z + d_zt) = 0$ → solution t_{hit}

Intersection of a Ray with a Sphere

- $(S_x + d_x t)^2 + (S_y + d_y t)^2 + (S_z + d_z t)^2 - 1 = 0$
- $|(\underline{\mathbf{S}} + \underline{\mathbf{d}} t)|^2 - 1 = 0$
- $|\underline{\mathbf{d}}|^2 t^2 + 2(\underline{\mathbf{S}} \cdot \underline{\mathbf{d}}) t + |\underline{\mathbf{S}}|^2 - 1 = 0$
- $A = |\underline{\mathbf{d}}|^2 \quad B = (\underline{\mathbf{S}} \cdot \underline{\mathbf{d}}) \quad C = |\underline{\mathbf{S}}|^2 - 1$
  - $At^2 + 2Bt + C = 0$ (standard quadratic equation)
  - $t_{hit} = -(B/A) \pm \text{sqrt}(B^2 - AC)/A$
- If $\Delta < 0$ no solution, ray misses the sphere
- If $\Delta = 0$ ray tangent to the sphere

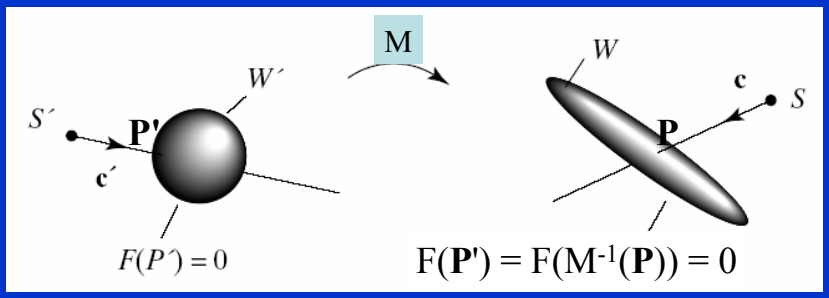
Numeric Example

- *Ray* $r(t) = (3, 2, 3) + (-3, -2, -3)t$
- $A = |\underline{\mathbf{d}}|^2 = 22$, $B = (\underline{\mathbf{S}} \cdot \underline{\mathbf{d}}) = -22$, $C = |\underline{\mathbf{S}}|^2 - 1 = 21$.
- $t_1 = 0.7868$ and $t_2 = 1.2132$.
- Hit points:
- $\mathbf{P}_1 = \boldsymbol{S} + \mathbf{d}t_1 = (0.64, 0.43, 0.64)$
- $\mathbf{P}_2 = \boldsymbol{S} + \mathbf{d}t_2 = (-0.64, -0.43, -0.64)$
- Opposite points relative to the origin

Intersection of a Ray with Transformed Objects

- object in the scene: generic object transformed by affine *M*
  - e.g: ellipsoid is a generic sphere transformed by some *M*
- *Every point* **P** of the ellipsoid object originated from a *generic point* $M^{-1}(\mathbf{P})$ on the generic object.
  - Hence **P** satisfies $F(M^{-1}(\mathbf{P})) = 0$
  - Where F() is the implicit function of generic object
- In particular, hit points of ray $\mathbf{S} + \underline{\mathbf{d}}t$ with object satisfy
  - $F(M^{-1}(\mathbf{S} + \underline{\mathbf{d}}t)) = 0$ $\quad \mathbf{r}'(t) = M^{-1}(\mathbf{S} + \underline{\mathbf{d}}t)$ *transformed ray*

Example



Intersection of a Ray with Transformed Objects

- transformed ray: $\mathbf{r}'(t) = M^{-1}(\mathbf{S} + \underline{\mathbf{d}}t) = \mathbf{S}' + \underline{\mathbf{d}}'t$
  - $\mathbf{S}' = M^{-1}\mathbf{S}$ $\quad \underline{\mathbf{d}'} = M^{-1}\underline{\mathbf{d}}$
- Each object in object list has its own M
- To intersect a ray $\mathbf{r}(t) = \boldsymbol{S} + \underline{\mathbf{d}}t$ with a transformed object:
  - Inverse transform the ray (obtaining $\mathbf{r}'(t) = \boldsymbol{S}' + \underline{\mathbf{d}'}t$)
  - Find intersection t_h of $\mathbf{r}'(t)$ with *generic object*
  - The *same* t_h in $\mathbf{r}(t) = \boldsymbol{S} + \underline{\mathbf{d}}t$ is the actual hit point

Example: Intersecting an ellipsoid

- An ellipsoid *W* is formed from the generic sphere by
- First, scaling
  - S(1, 4 ,4)
- Then, translation
  - T(2, 4, 9)
- Combined transformation M and its inverse M^{-1} are:

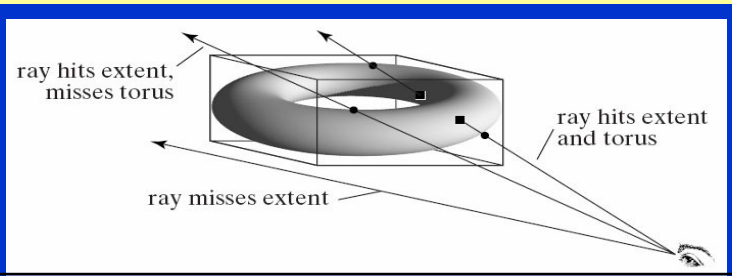
$$M = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 4 & 0 & 4 \\ 0 & 0 & 4 & 9 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M^{-1} = \begin{pmatrix} 1 & 0 & 0 & -2 \\ 0 & 1/4 & 0 & -1 \\ 0 & 0 & 1/4 & -9/4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example: Intersecting an ellipsoid (2)

- Ray: $\mathbf{r}(t) = (10, 20, 5) + (-8, -12, 4)t$
- Finding intersection with ellipsoid W:
- inverse transformed ray:
  - $\mathbf{r'}(t) = M^{-1}\,\mathbf{r} = (8,4,-1) + (-8,-3,1)t$
- Intersection: $(S'_x + d'_x t)^2 + (S'_y + d'_y t)^2 + (S'_z + d'_z t)^2 - 1 = 0$
- $At^2 + 2Bt + C = 0$
  - $(A, B, C) = (74,-77,80) \quad \Delta = 9$
  - $t_h = 1.1621 ; 0.9189$
  - Intersection: $\mathbf{r}(0.9189) = (2.649, 8.97, 8.67)$
  - $x = 2.649; \quad y = 8.97; \quad z = 8.67$

extents

- a torus is expensive to intersect
- Hence it is virtually enclosed in a box-like *extent*
- 1. test intersections with the box
- 2. if no intersection then no intersection with torus
- else, test for intersection with torus
- sphere extents are also used (for other objects)



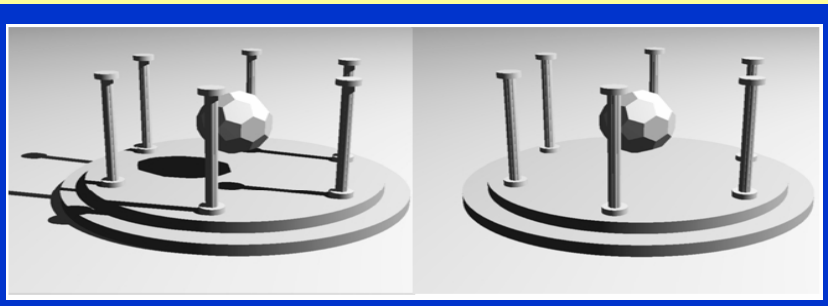
time saving by extent

- Assumes it takes T time units to test a ray against the extent
  - and mT time units to test intersection against the torus.
- Assume that N rays are cast to create the image
  - and only fraction f of them hit the box extent
- N tests are made against the extent, time = NT,
  - fN tests are then made on the torus, time = $fNmT$
- If using extent, the total time is $t_E = NT(1 + fm)$
- if extents are not used, total time of $t_N = NmT$.
- Speedup ratio $t_E/t_N = m/(1 + fm)$
  - Extent useful if $m >> 1$ and $f << 1$ (small, tight extent)

Adding Shadows

Affect of Shadows

- Shadows are important
- Without shadows (right) it is difficult to see how far above the platform the ball lies,
- With shadows (left) we can see this immediately.
- Ray tracing produces shadows easily. Unfortunately, it slows down the ray tracing process.

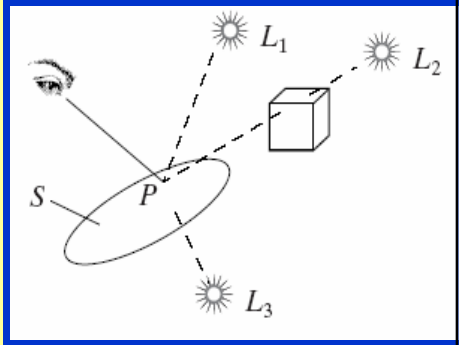


How a shadow is created

- until now we fired a ray through a certain pixel P_{rc}
  - we found closest hit point P_h on some face *F*
- We assumed that *light originated from all light sources* arrived to face *F* at P_h, reflected by F through the pixel
- But if another object lies between P_h and a light source L
  - P_h does not get light from light source L
  - P_h does not reflect light from L
  - Intensity of light from P_h is lower than near by points
  - P_h is in the shadow of that object relative to L

Shadowing situations

- Point P can see source L_1
  - P is not in shadow relative to L_1.
- P *is* in the shadow of the cube relative to source L_2.
- the hit object itself hides the source L_3 from P
  - self-shadowing



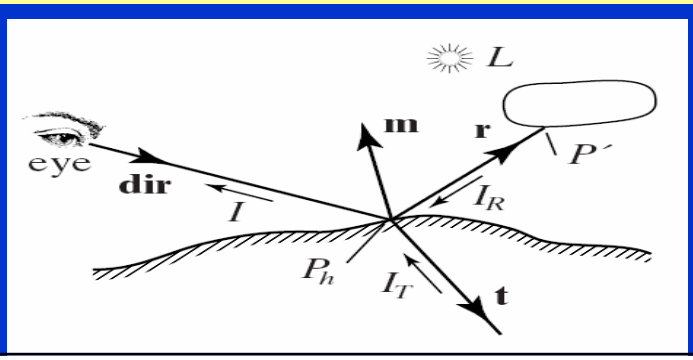
Finding shadows

- Is P_h in some shadow relative to light source L_i?
- spawn a *shadow feeler ray* from P_h at $t = 0$, to L_i at $t = 1$
  - parametric representation $P_h + (L_i - P_h)t$
- Test for intersection of shadow feeler with *all objects*.
- If any intersection between $t = 0$ and $t = 1$, answer is yes
  - ignore *diffuse* & *specular* light from P_h for light source L_i
- Repeat for all light sources, L_k
- P_h emits ambient light, and
  - diffuse & specular lights from *visible light sources* only

Reflections and Transparancy

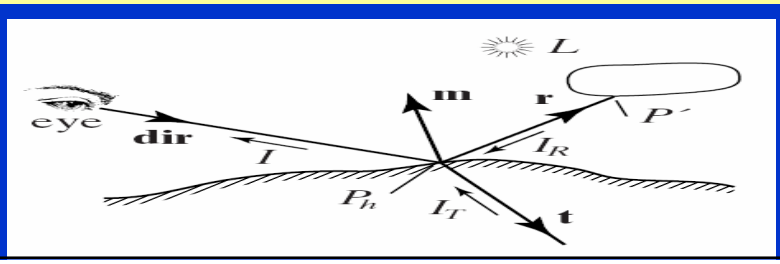
Reflections and Transparency

- before reaching the eye
  - light might bounce off several shiny surfaces
  - Light might be refracted through transparent objects
  - $I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + I_{tran}$ (5 components of light)
- ray tracing is capable of adding these effects



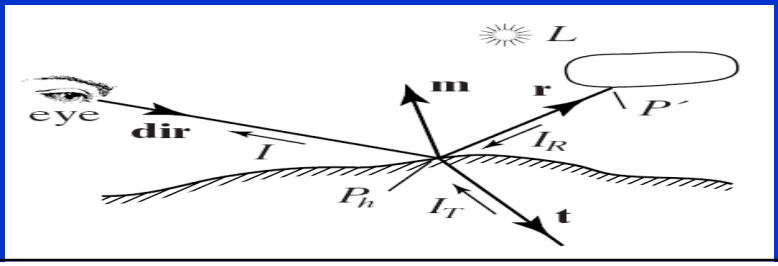
Reflections and Transparency (2)

- Light that reaches the eye: $I = I_{amb} + I_{diff} + I_{spec} + \boldsymbol{I_{refl}} + I_{tran}$
  - diffuse and specular parts arise from visible light sources
  - $\boldsymbol{I_{refl}}$ is the reflected light component
  - arising from light, $\boldsymbol{I_R}$, incident at P_h along direction **-r**.
  - such that the angles of incidence and reflection are equal
  - **r** = **dir** – 2 (**dir**·**m**) **m,** where **m** is normalized



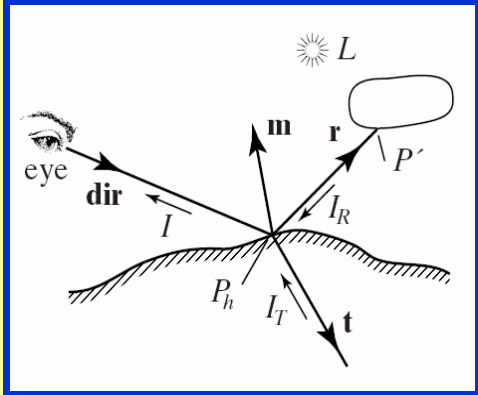
Reflections and Transparency (3)

- $I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + \boldsymbol{I_{tran}}$ (5 components of light)
- $\boldsymbol{I_{tran}}$ transmitted light component,
  - arising from the light, $\boldsymbol{I_T}$, that is transmitted through the transparent material to P_h along direction **-t**.
- A portion of this light passes through the surface and in so doing is bent. It then continues its travel along **-dir**.



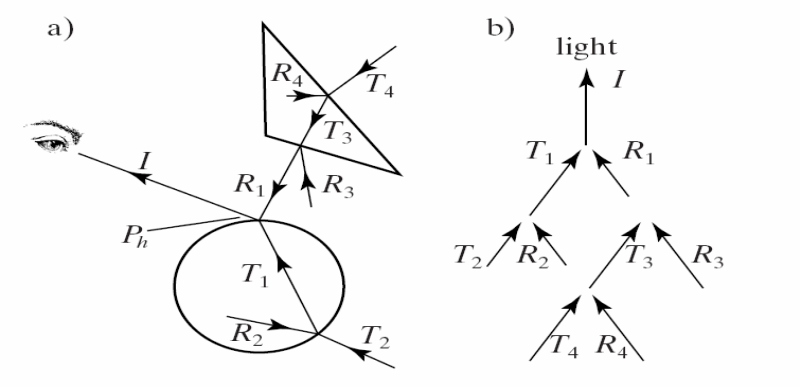
Reflections and Transparency (4)

- I_R and I_T are composed of their own five components:
  - ambient, diffuse, etc..
- I_R emitted from **P'** to $\boldsymbol{P}_h$.
- find **P'**: spawn a ray from $\boldsymbol{P}_h$ in the direction **r**
- find the first object it hits
  - I_R is a sum of its 5 light components
- similarly for I_T



The Tree of Reflected & Refracted light rays

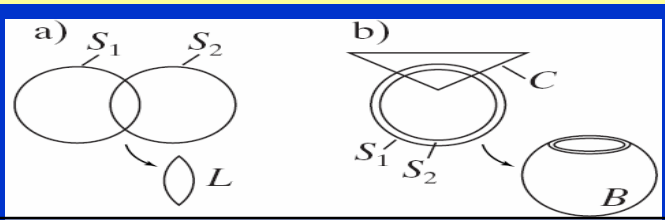
- Note: local components of light at P_h
  - Ambient at P_h
  - diffuse & specular from light sources visible at P_h
- These are not shown here



Constructive Solid Geometry

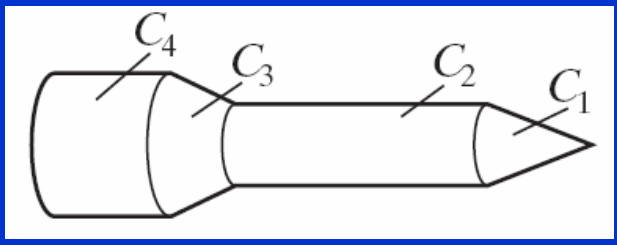
Constructive Solid Geometry

- complex objects defined by set operations on simple objects
  - compound, Boolean, or CSG objects
- E.g Primitives: *solid* spheres S_1, S_2, cone C
- Left: Lens constructed from *intersection* of the 2 spheres
  - $L = S_1 \cap S_2$ (every point in L is in *both* S_1 or S_2)
- Bowl constructed from *difference* of 2 spheres and cone
- $B = (S_1 - S_2) - C$ (every point in B is in S_1 but *not* in S_2 , C)



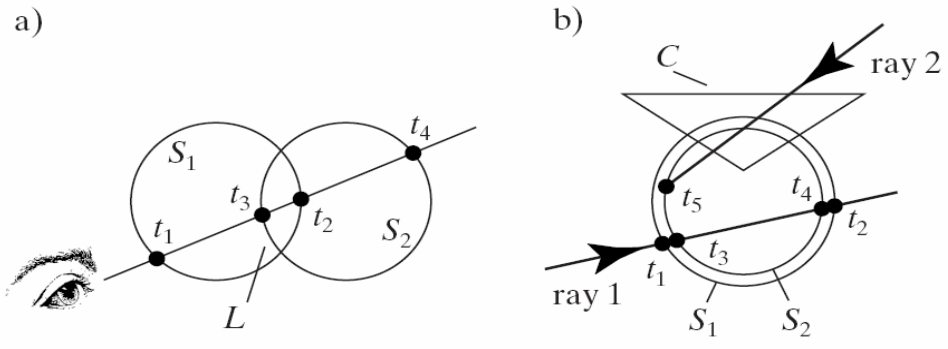
More Compound Objects

- A point is in the *union* of sets *A, B*, if it is in *A* or in *B* or in both (A and B are glued together)
- The rocket is a *union* of two cones and two cylinders
- $R = C_1 \cup C_2 \cup C_3 \cup C_4$
- Note: Cone C_3 is partially embedded in C_2



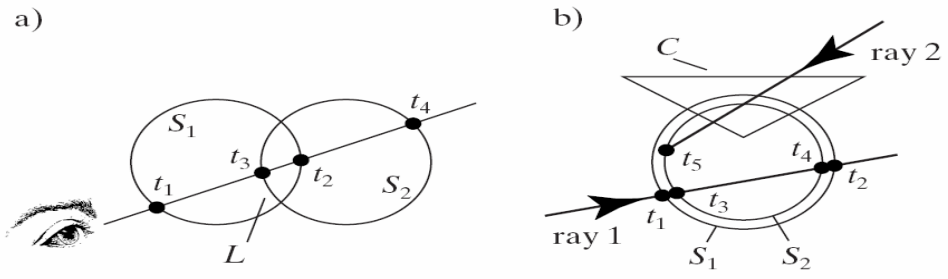
Ray Tracing CSG Objects: Example 1

- Left: ray intersect spheres at t_1, t_2, t_3, t_4
- Ray inside lens *L* from t_3 to t_2,
- hit point is t_3
- First point which is in *both* S_1 and in S_2



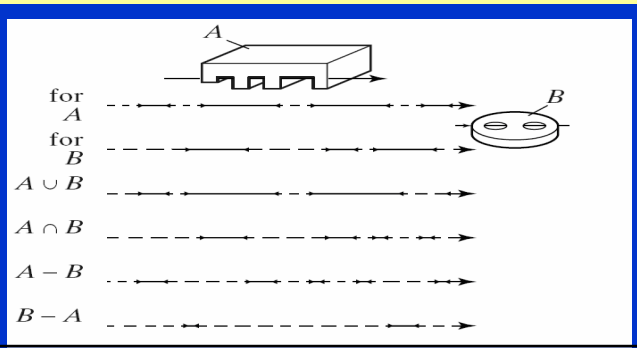
Ray Tracing CSG Objects: Example 2

- Ray 1 first strikes the bowl at t_1,
  - smallest of times for which it is in S_1 but not in S_2 or C.
- Ray 2, first hits the bowl at t_5.
  - smallest time for which it is in S_1, but in neither S_2, C
- hits at earlier times are with component parts of the bowl, but not the bowl itself.



Ray Tracing CSG Objects: general idea

- Consider two *solid* objects, A and B, and a ray.
- Build 2 sorted lists of *intersection times* with A, B
  - objects are solid, so entry and exit times alternate.
- graphically:
  - intervals inside object are solid. Outside they are dashed



Ray Tracing CSG Objects: general idea (2)

- *T(A) Inside set for a ray with a compound object*:
  - set of *t*-values for which the ray is inside that object
  - T(A) = (t_1, t_2 , . . .) t_1 enter time, t_2 is an exit time, etc..
  - also called ray's ***t*-list**.
- Given the inside sets T(A), T(B) with objects *A*, *B*
  - what is the inside set (for the ray) with a compound object built from *A* and *B*?
- inside set with a Union is the union of inside sets
- inside set with intersection is the intersection of inside sets
- in general: T(A op B) = T(A) op T(B)

Ray Tracing CSG Objects: general idea (3)

- A_list: (1.2, 1.5) (2.1, 2.5) (3.1, 3.8)
- B_list: (0.6, 1.1) (1.8, 2.6) (3.4, 4.0)
- Union: (0.6, 1.1) (1.2, 1.5) (1.8, 2.6) (3.1, 4.0)
- Intersection: (2.1, 2.5) (3.4, 3.8)
- A – B: (1.2, 1.5) (3.1, 3.4)
- B – A: (0.6, 1.1) (1.8, 2.1) (2.5, 2.6) (3.8, 4.0)

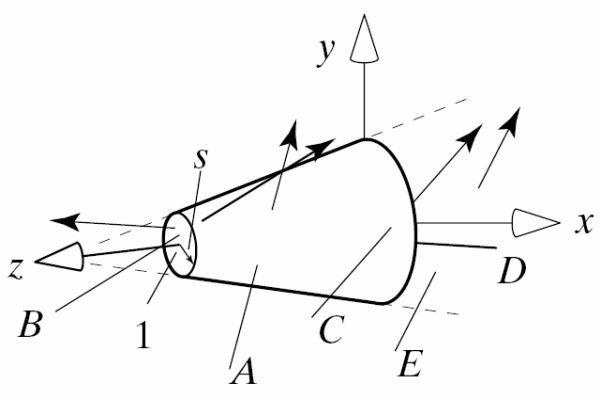
Ray Tracing CSG Objects: general idea (4)

- Ray trace component objects,
- building inside sets for each,
- combining inside sets according to the operation required
- The first positive hit time on the combined list yields the point on the compound object that is hit first by the ray.
- The usual shading is then done, including the casting of secondary rays if the surface is shiny or transparent.

Appendix: Intersections with Tapered Cylinders and Cubes

Intersecting Rays with Tapered Cylinders

- (transformed) ray: **r**(t) = **S** + **d**t
- The ray can intersect the cylinder in many ways



Intersection with side of tapered cylinder

- part of an infinitely long wall; radius (1, s) at $z = (0, 1)$
- Implicit form: for $0 < z < 1$
  - $F(x, y, z) = x^2 + y^2 + (1 + (s - 1)z)^2 = 0$
  - $s = 1$ generic cylinder
  - $s = 0$ generic cone.
- Intersection:
  - Substitute $(x, y, z) = (S_x, S_y, S_z) + (d_x, d_y, d_z)t$ in $F = 0$
  - Result: $At^2 + 2Bt + C = 0$
  - $A = d_x^2 + d_y^2 - e^2$ $\quad B = S_x d_x + S_y d_y - Fe$
  - $C = S_x^2 + S_y^2 - F^2$
  - Where $e = (s - 1)d_z$ $\quad F = 1 + (s - 1)S_z$

Intersection with base/cap of tapered cylinder

- If $B^2 - AC$ is negative no intersection with infinite wall
- Else ray intersect the infinite wall
- find the z-component of the hit spot. The ray hits the actual wall of cylinder only if the $z(t_{hit})$ lies between 0 and 1
- intersection with base:
- intersect ray with the plane $z = 0$. If intersection point is $(x, y, 0)$, then it is on the base if $x^2+y^2 < 1$.
- intersection with the cap:
- intersect ray with the plane $z = 1$. If intersection point is $(x, y, 1)$ then it is in the cap if $x^2+y^2 < s^2$

Intersecting a Ray with a Cube

- generic cube: centered at **O**, corners at $(\pm1, \pm1, \pm1)$
- Six planes that define the generic cube & normals

Plane	Name	Eqn.	Out Normal	Spot
0	top	y = 1	(0, 1, 0)	(0, 1,0)
1	bottom	y = -1	(0, -1, 0)	(0, -1, 0)
2	right	x = 1	(1, 0, 0)	(1, 0, 0)
3	left	x = -1	(-1, 0, 0)	(-1, 0, 0)
4	front	z = 1	(0, 0, 1)	(0, 0, 1)
5	back	z = -1	(0, 0, -1)	(0, 0, -1)

Intersecting a Ray with a Cube (2)

- Scene contains many boxes
  - Generated by transforming a generic cube
- cubes is used as a bounding box *(extent)* for other primitives
  - E.g. bounding box surrounding a tapered cylinder
  - If a ray skips the extent, it doesn't intersect the primitive
- Intersection of a ray with generic cube is important
- basic idea
  - each plane defines an inside & outside half spaces
  - A point is inside a cube iff it is inside of all half-spaces
  - intersecting a ray with a cube: find the interval of t in which the ray lies inside all of the planes of the cube.