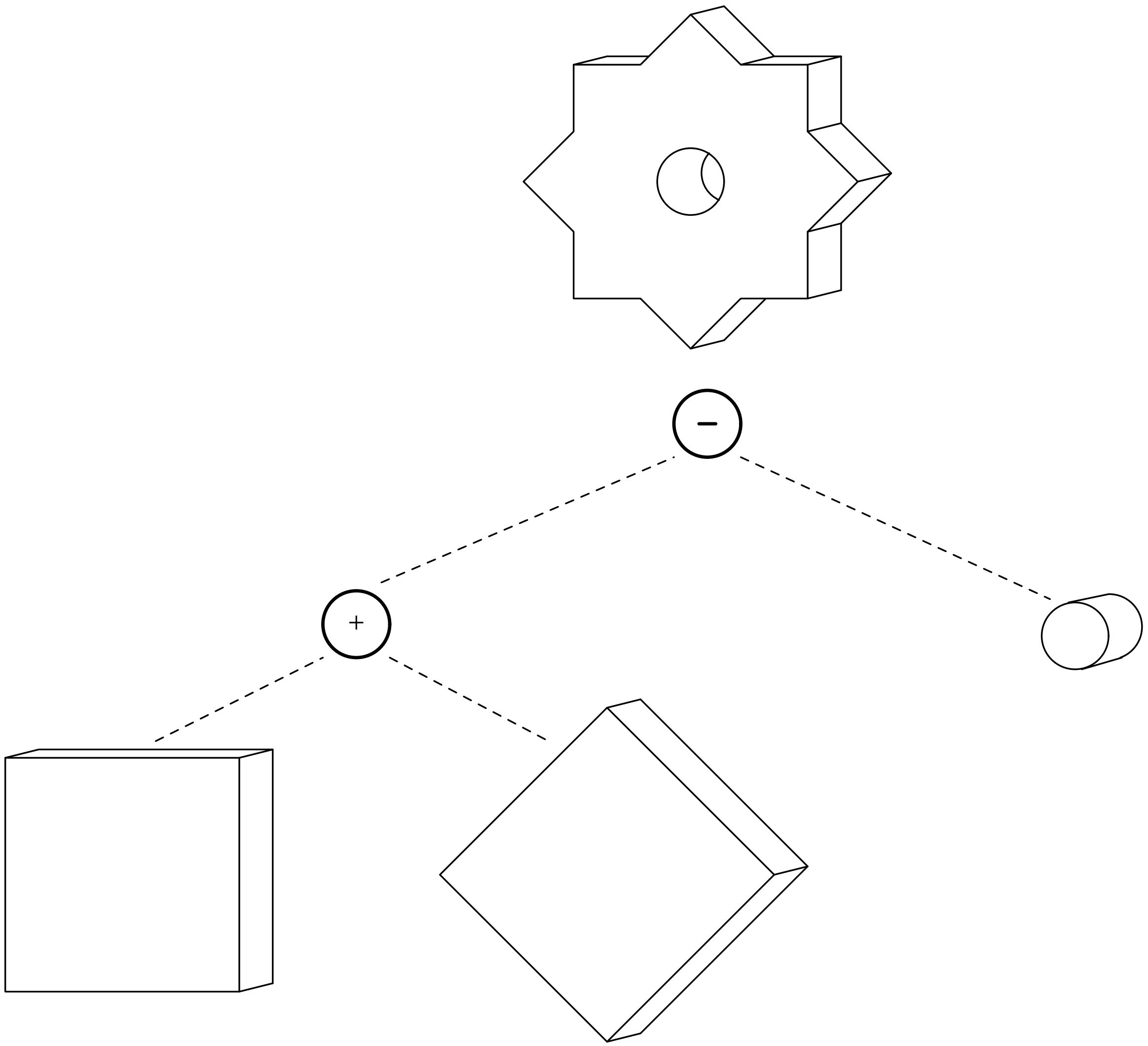
Constructive Solid Geometry

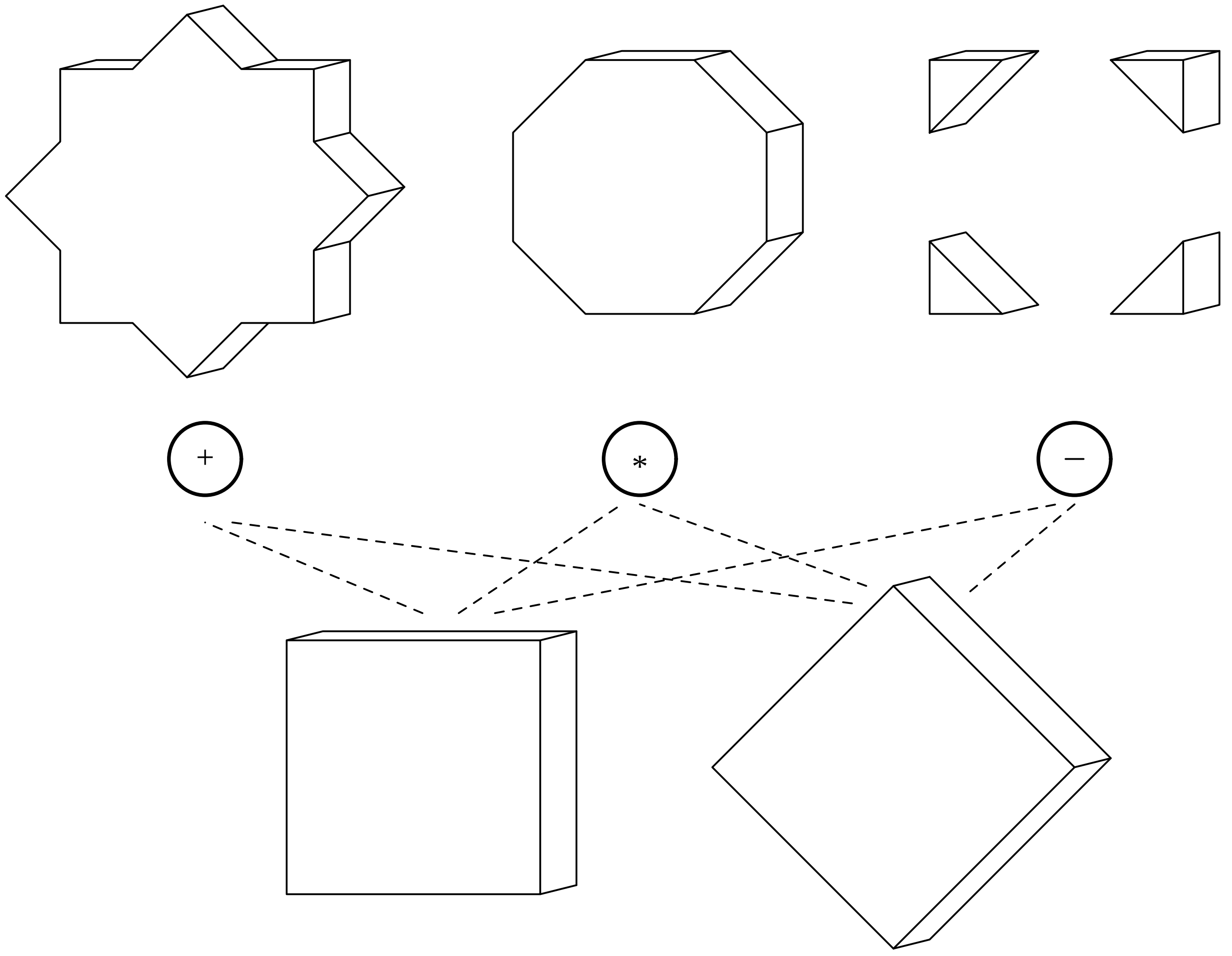
A technique for modeling objects, particularly manufactured objects.

- **A set of primitive solid objects**. Sphere, cylinder, block, cone, ellipsoid, dodecahedron, pyramid, etc.
- **A set of operators for combining them**.
  Union: Create a new solid that is the union of two solids.
  Intersection: Create a new solid that is the intersection of two solids.
  Difference: Create a solid by subtracting one solid from another.

Example: Constructing a 8-pointed wheel

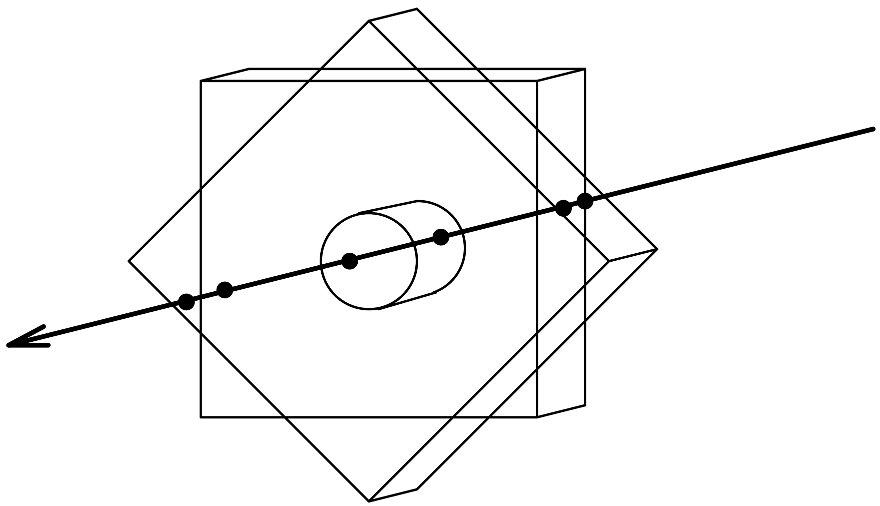


Boolean Set Operations on Solids

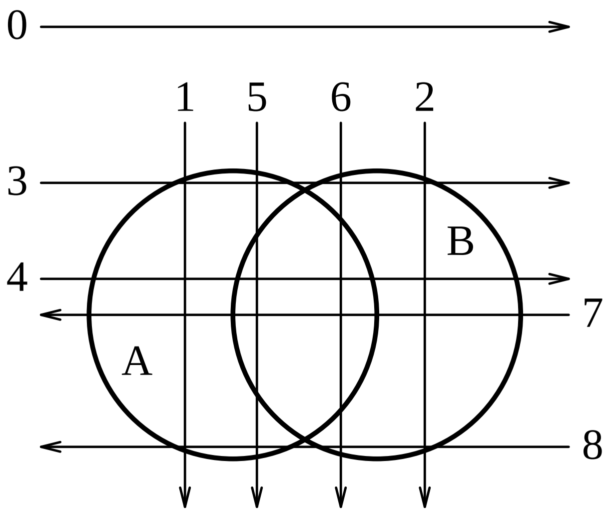


Rendering CSG Solids With Raytracing

Shoot a ray toward each of the primitives and compute *hit lists* of linked lists where the ray enters and exits each primitive. From these, we can compute where the ray enters and exits the combined solid and adjust the surface normals properly.

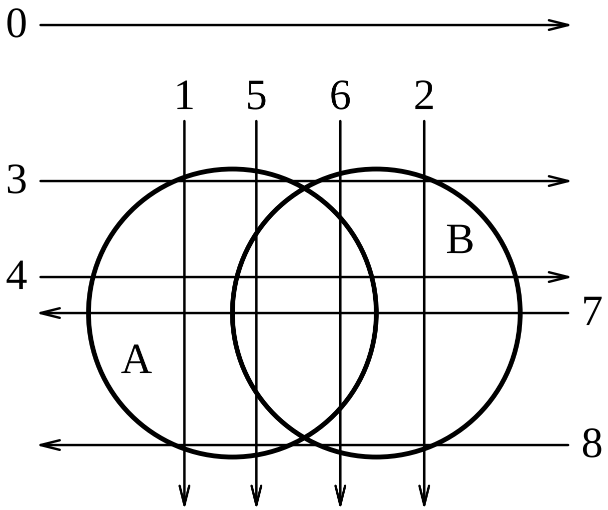


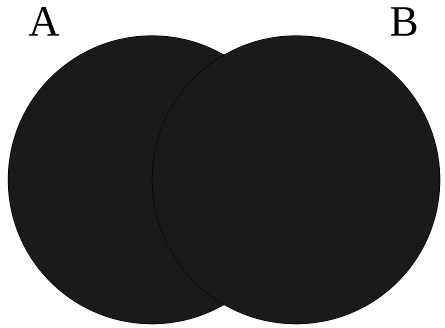
Boolean Operations and Hit Lists



0	1	2	3	4	5	6	7	8
	A_1	B_1	A_1	A_1	A_1	B_1	B_1	B_1
	A_2	B_2	A_2	B_1	B_1	A_1	A_1	B_2
			B_1	A_2	B_2	A_2	B_2	A_1
			B_2	B_2	A_2	B_2	A_2	A_2

Union

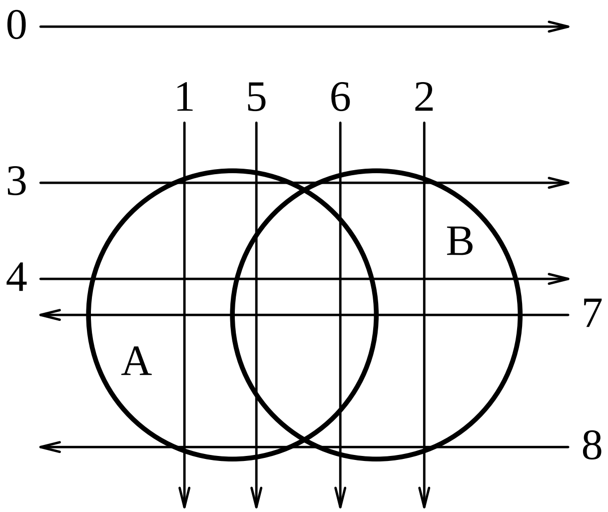


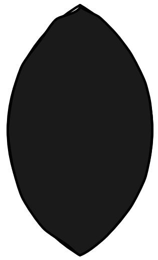


0	1	2	3	4	5	6	7	8
	A_1	B_1	A_1	A_1	A_1	B_1	B_1	B_1
	A_2	B_2	A_2	B_1	B_1	A_1	A_1	B_2
			B_1	A_2	B_2	A_2	B_2	A_1
			B_2	B_2	A_2	B_2	A_2	A_2

0	1	2	3	4	5	6	7	8
	A_1	B_1	A_1	A_1	A_1	B_1	B_1	B_1
	A_2	B_2	A_2					B_2
			B_1					A_1
			B_2	B_2	A_2	B_2	A_2	A_2

Intersection

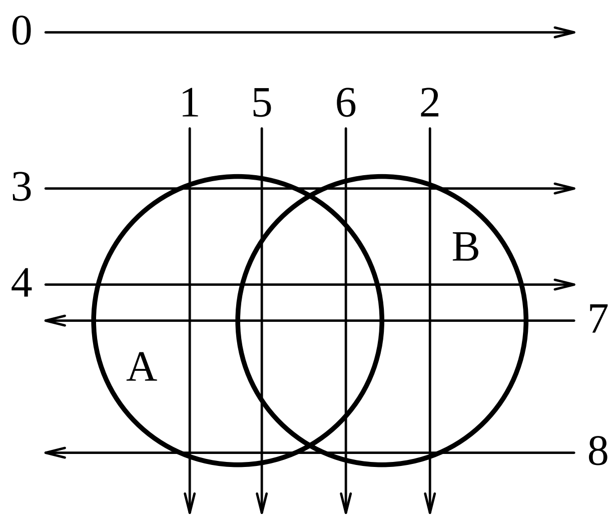


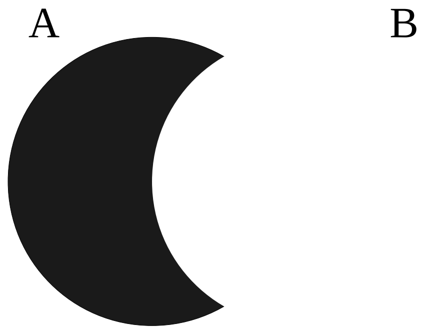


0	1	2	3	4	5	6	7	8
	A_1	B_1	A_1	A_1	A_1	B_1	B_1	B_1
	A_2	B_2	A_2	B_1	B_1	A_1	A_1	B_2
			B_1	A_2	B_2	A_2	B_2	A_1
			B_2	B_2	A_2	B_2	A_2	A_2

0	1	2	3	4	5	6	7	8
				B_1	B_1	A_1	A_1	
				A_2	B_2	A_2	B_2	

Difference $(A - B)$

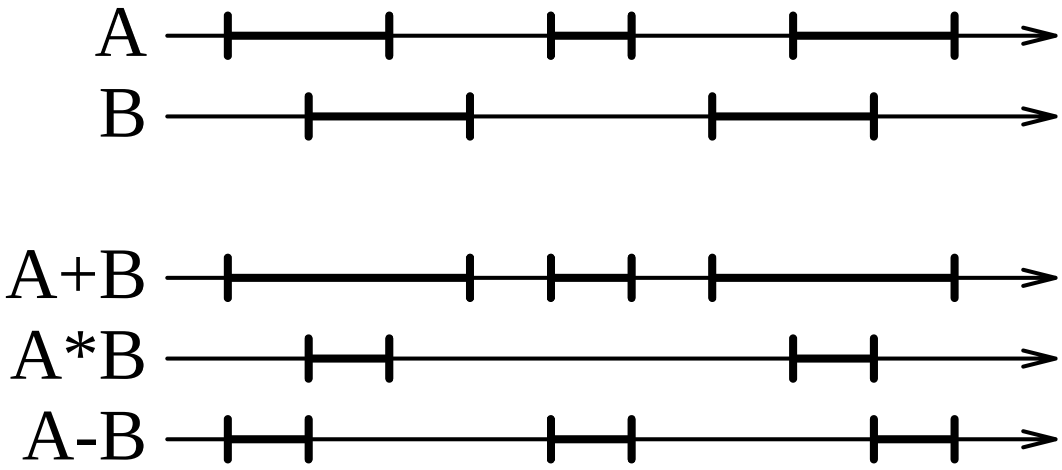




0	1	2	3	4	5	6	7	8
	A_1	B_1	A_1	A_1	A_1	B_1	B_1	B_1
	A_2	B_2	A_2	B_1	B_1	A_1	A_1	B_2
			B_1	A_2	B_2	A_2	B_2	A_1
			B_2	B_2	A_2	B_2	A_2	A_2

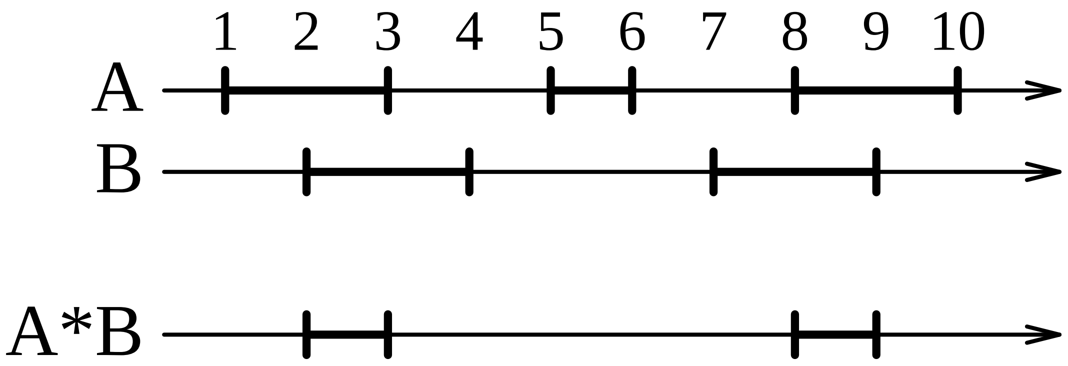
0	1	2	3	4	5	6	7	8
	A_1		A_1	A_1	A_1			
	A_2		A_2	B_1	B_1			
					B_2		B_2	A_1
					A_2		A_2	A_2

Roth Diagrams



State of		State of *flagc* for		
flaga	*flagb*	$A+B$	$A*B$	$A-B$
0	0	0	0	0
0	1	1	0	0
1	0	1	0	1
1	1	1	1	0

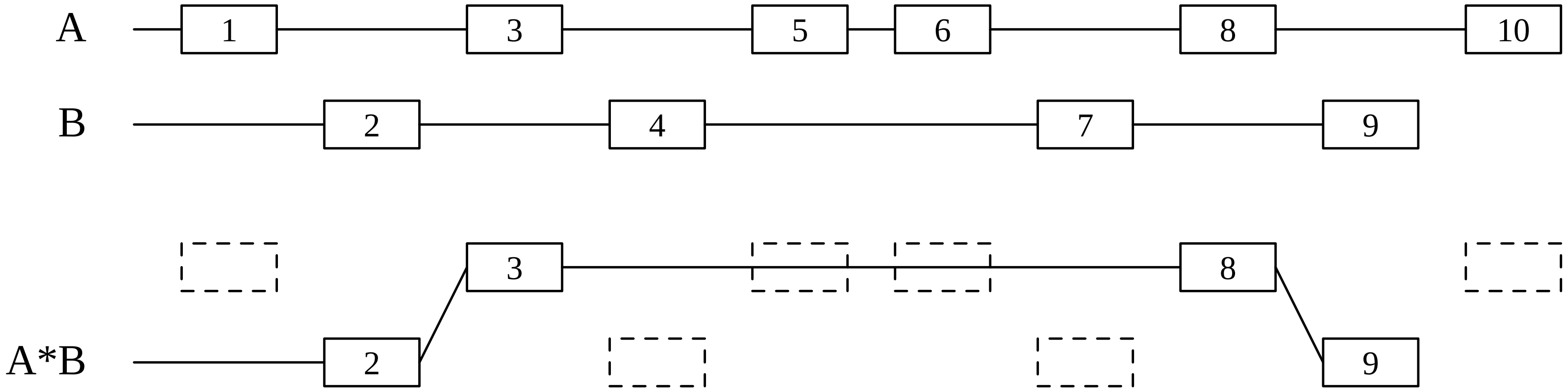
Computing the Intersection



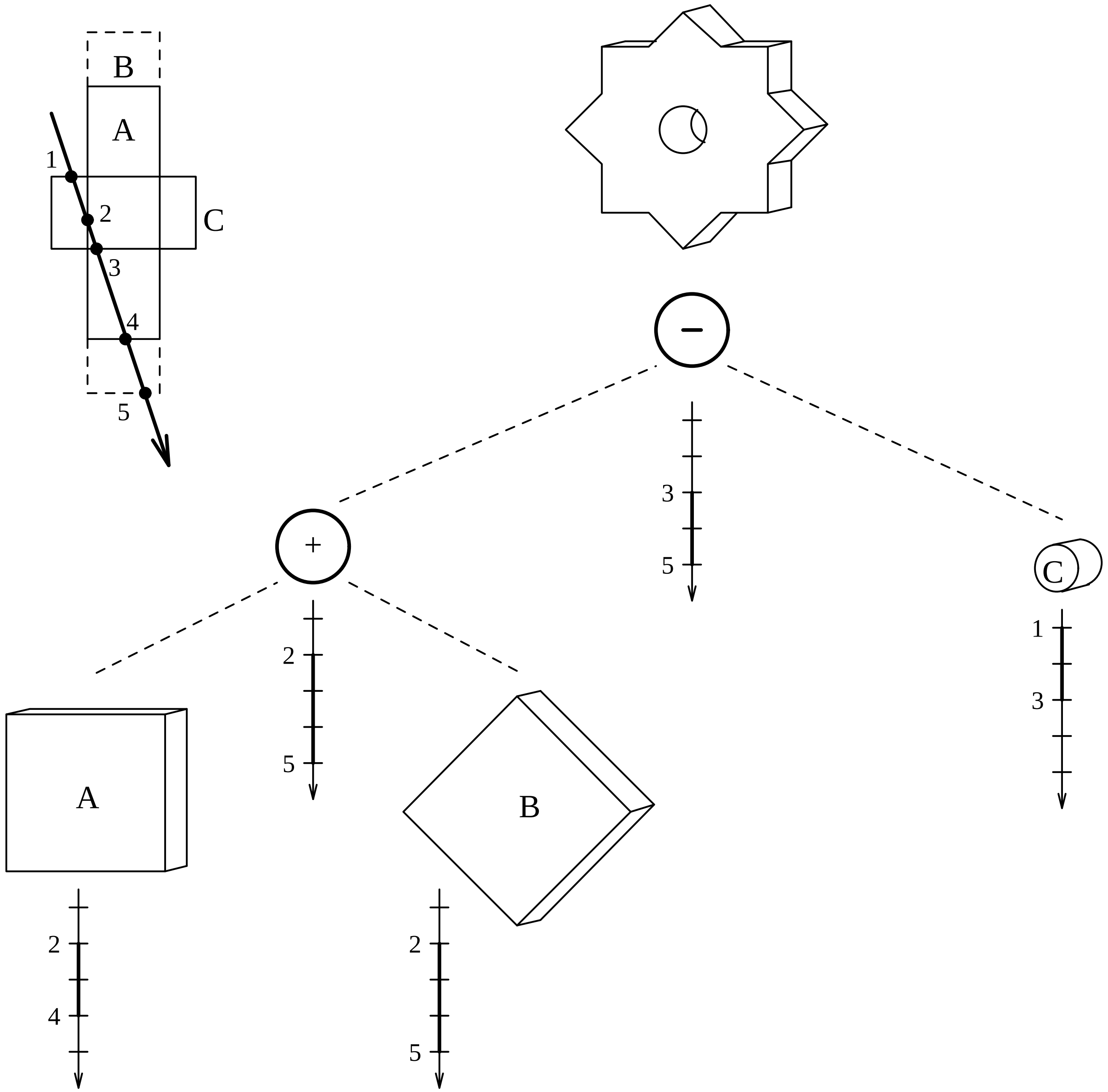
Point	*flaga*	*flagb*	*flagc*	Notes
start	0	0	0	
1	1	0	0	
2	1	1	1	enter
3	0	1	0	exit
4	0	0	0	
5	1	0	0	
6	0	0	0	
7	0	1	0	
8	1	1	1	enter
9	1	0	0	exit
10	0	0	0	

Weaving Hit Lists

When we trace the ray through each solid, we obtain a **hit list**: a linked list of entry and exit points. We "weave" these hit lists together to obtain the hit list of the composite object by computing *flaga, flagb,* and *flagc*.



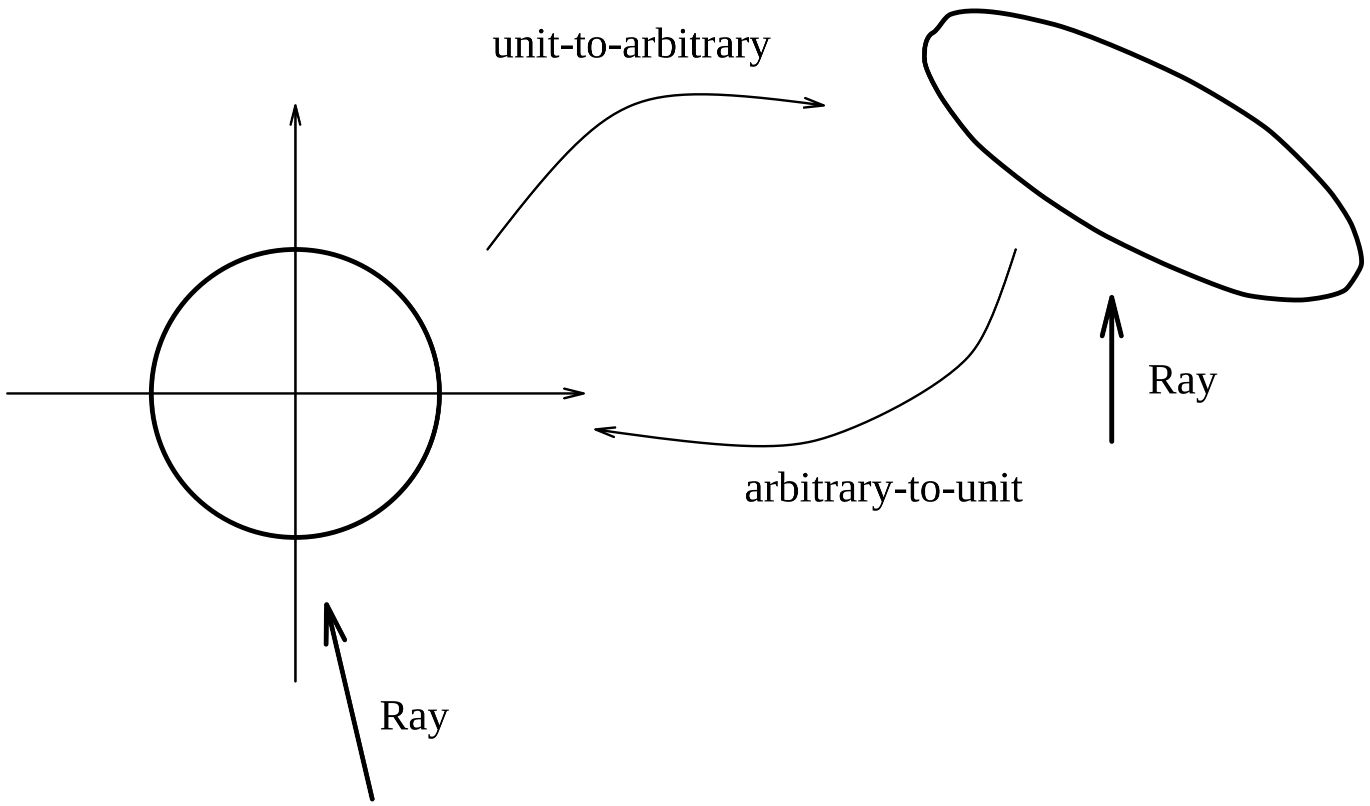
Raytracing CSG Solids



Manipulating Geometric Primitives

We represent each primitive in a standard orientation, location, and scale. To trace a ray against it, we apply the **arbitrary-to-unit** transformation to the ray to place it in the primitive's standard frame of reference. This gives us the value of u, which we can apply to the original ray without further transformations.

However, we must transform the surface normal vector from the standard frame of reference to the world coordinates.



Example: Ellipsoids

An ellipsoid is a sphere that has been scaled by different amounts along each axis. After scaling, it is rotated and translated. Let

$c.x$	Amount to translate in x dimension
$c.y$	Amount to translate in y dimension
$c.z$	Amount to translate in z dimension
$a.x$	Amount to scale in x dimension
$a.y$	Amount to scale in y dimension
$a.z$	Amount to scale in z dimension
$r.x$	Amount to rotate about x axis
$r.y$	Amount to rotate about y axis
$r.z$	Amount to rotate about z axis

$sx = \sin(r.x) \qquad sy = \sin(r.y) \qquad sz = \sin(r.z)$

$cx = \cos(r.x) \qquad cy = \cos(r.y) \qquad cz = \cos(r.z)$

Let P be a point on a sphere centered at the origin with radius 1. This will be our "unit" solid for spheres and ellipsoids.

Unit-to-Arbitrary Transformation

It is easiest to compute the arbitrary-to-unit transformation by first computing its inverse—the unit-to-arbitrary transforamtion.

To transform P_{unit} to the world coordinate frame of reference, we perform:

$$P_{arb} = P_{unit} \cdot A \cdot R_x \cdot R_y \cdot R_z + C$$

where

$$A = \begin{pmatrix} a.x & 0 & 0 \\ 0 & a.y & 0 \\ 0 & 0 & a.z \end{pmatrix} \qquad R_y = \begin{pmatrix} cy & 0 & sy \\ 0 & 1 & 0 \\ -sy & 0 & cy \end{pmatrix}$$

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cx & sx \\ 0 & -sx & cx \end{pmatrix} \qquad R_z = \begin{pmatrix} cz & sz & 0 \\ -sz & cz & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The Arbitrary-to-Unit Transformation

Given a ray with starting point S_{arb} and unit direction vector D_{arb} we must do

$$S_{unit} = (S_{arb} - C) \cdot T \qquad D_{unit} = D_{arb} \cdot T$$

where T is the arbitrary-to-unit transformation matrix:

$$T = R_z^{-1} \cdot R_y^{-1} \cdot R_x^{-1} \cdot A^{-1}$$

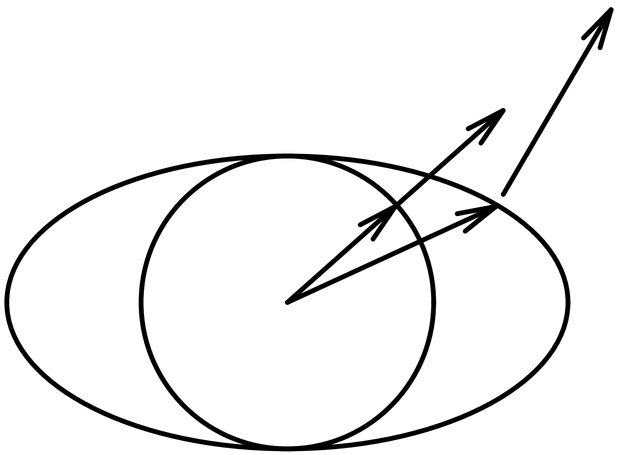
Computing these inverses is easy, because the inverse of a rotation matrix is just its transpose, and the inverse of a diagonal matrix is a diagonal matrix of reciprocals.

$$R_z^{-1} \cdot R_y^{-1} \cdot R_x^{-1} = \begin{pmatrix} cx\,cz & -sx\,sy\,cz - cx\,sz & -cx\,sy\,cz + sx\,sz \\ cx\,sz & -sx\,sy\,sz + cx\,cz & -cx\,sy\,sz - sx\,cz \\ sy & sx\,cy & cx\,cy \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1/a.x & 0 & 0 \\ 0 & 1/a.y & 0 \\ 0 & 0 & 1/a.z \end{pmatrix}$$

Transforming Normals

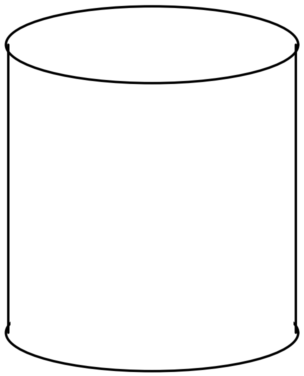
In addition to computing the intersection parameter u, we also compute the surface normal N_{unit}. We must now convert this into the surface normal in world coordinates, N_{arb}. It does not work to apply the unit-to-arbitrary transformation to N_{unit}:

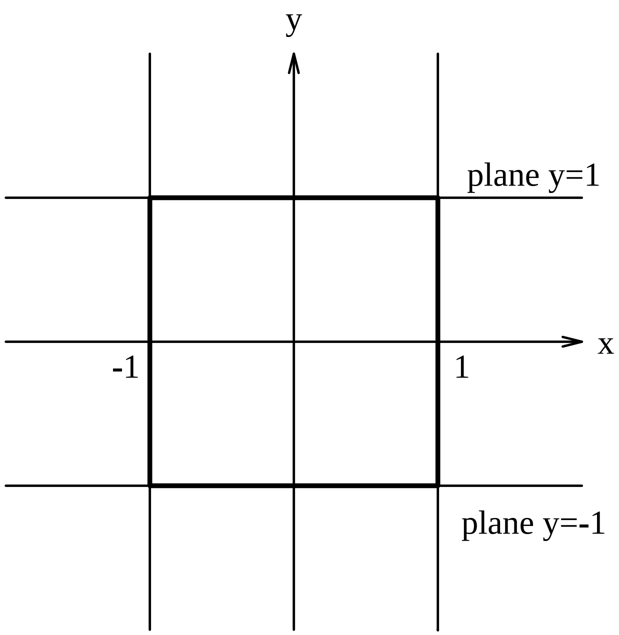


The normal must remain orthogonal to the transformed surface. This can be accomplished by using A^{-1} instead of A:

$$N_{arb} = N_{unit} \cdot A^{-1} \cdot R_x \cdot R_y \cdot R_z$$

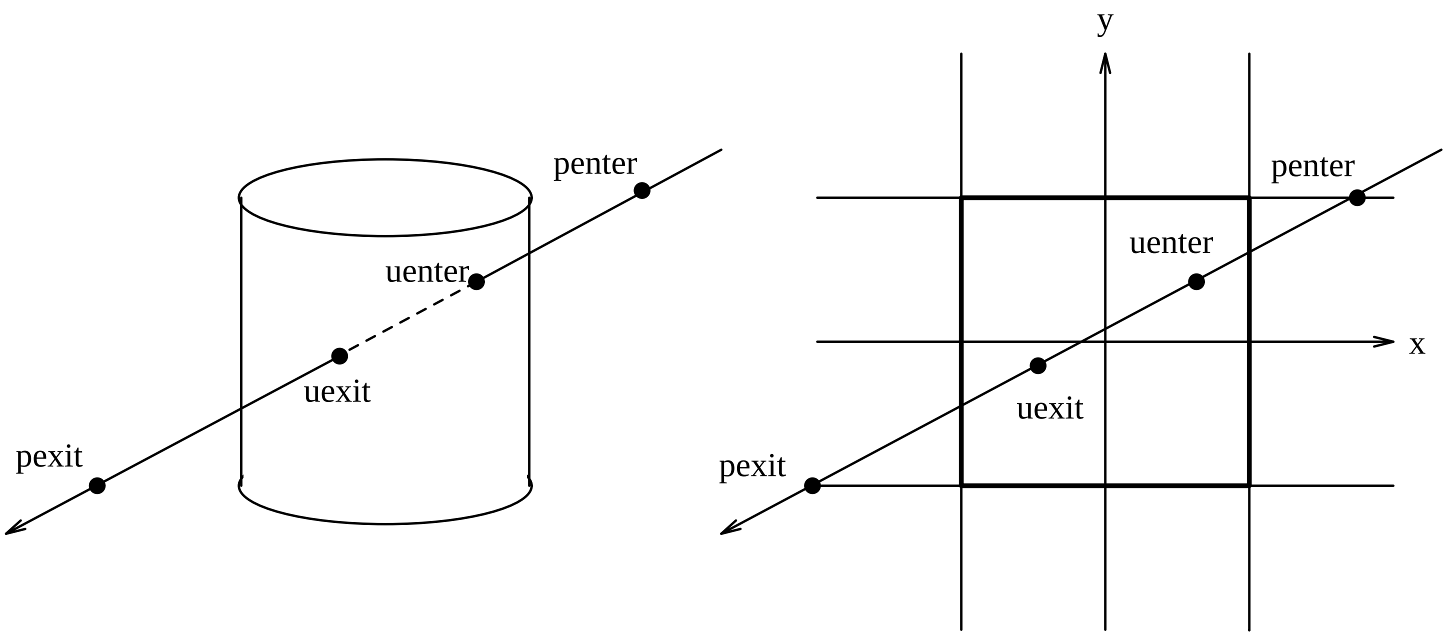
A Cylinder Primitive





The unit cylinder has height 2, radius 1, and center at the origin.

Ray Intersects Cylinder



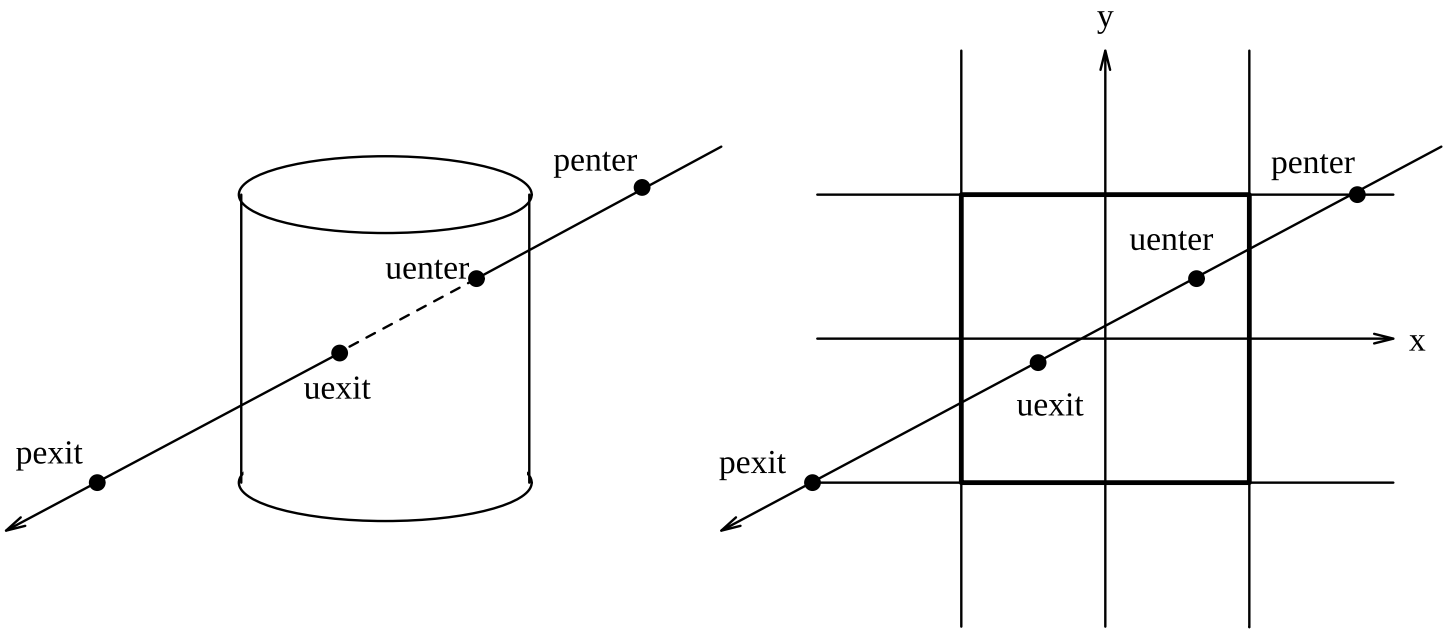
Intersection of ray (start point S, direction D) with infinite cylinder $x^2 + z^2 = 1$:

$$a_2\, u^2 + a_1\, u + a_0 = 0$$

$$a_2 = D.x^2 + D.z^2 \qquad a_1 = 2(S.x\; D.x + S.z\; D.z) \qquad a_0 = S.x^2 + S.z^2 - 1$$

Plug into quadratic formula. If two real solutions exist, the smaller is u_{enter} and the larger is u_{exit}.

Ray Intersects Cylinder (2)

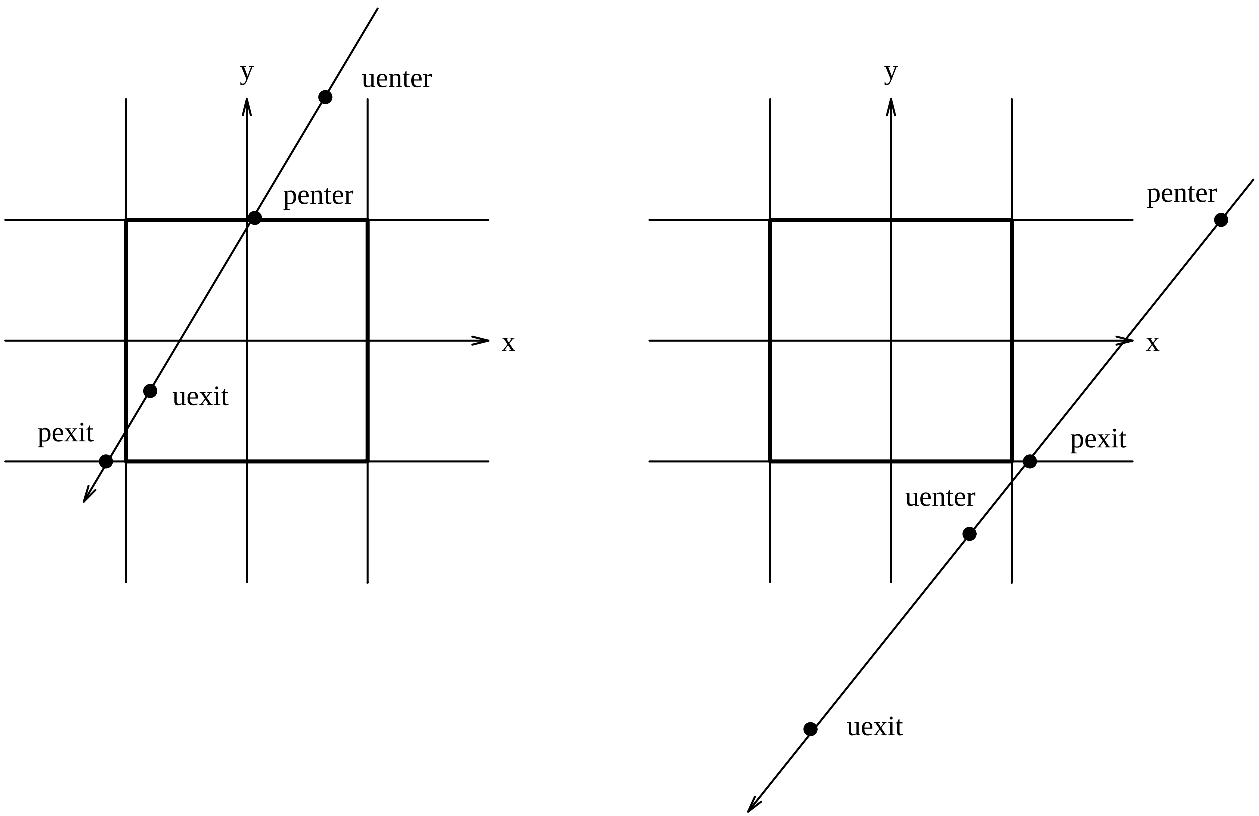


Intersection of ray with the two planes $y = 1$ and $y = -1$. (Skip if $D.y = 0$.)

$$
\begin{aligned}
p_1 &= \tfrac{1-S.y}{D.y} & p_2 &= \tfrac{-1-S.y}{D.y} \\
p_{enter} &= \min(p_1, p_2) & p_{exit} &= \max(p_1, p_2) \\
u_{enter} &= \max(p_{enter}, u_{enter}) & u_{exit} &= \min(p_{exit}, u_{exit})
\end{aligned}
$$

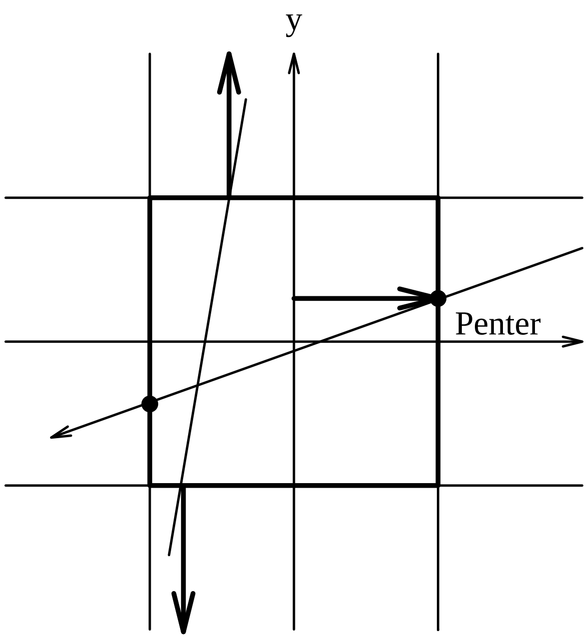
If $u_{enter} > u_{exit}$ then ray does not intersect cylinder.

Ray Intersects Cylinder (3)



General principle: Entry point is **max** of several entry points. Exit point is **min** of several exit points. (This is basically constructing a cylinder from the intersection of an infinite tube and a block.)

Cylinder Normals

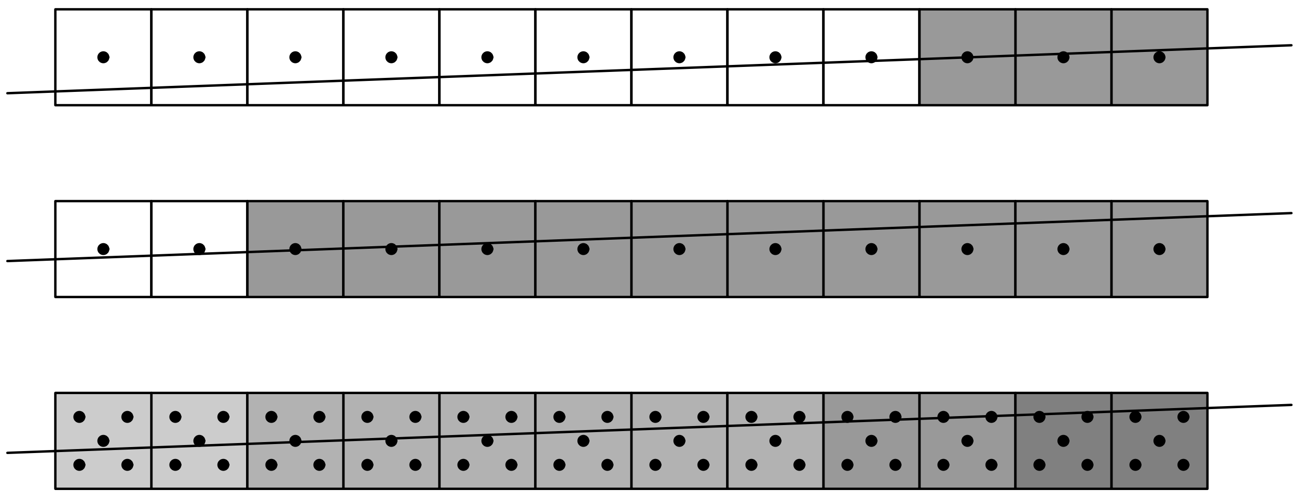


Compute P_{enter} in unit coordinates.

If $P_{enter}.y = 1$, normal is $(0, 1, 0)$.
If $P_{enter}.y = -1$, normal is $(0, -1, 0)$.
Else normal is $(P_{enter}.x, 0, P_{enter}.z)$.

Anti-Aliasing

Aliasing is any strange artifact that arises from digitizing continuous quantities. A slight change in the position of a line in world coordinates can cause a huge "jump" in the position of the digitized line in screen coordinates.



Super-sampling is one way to avoid this problem. We shoot 5 rays through each pixel and average the results. This can make a huge improvement in the quality of rendered images. Best results are usually obtained if we randomize the positions of the rays within each pixel.