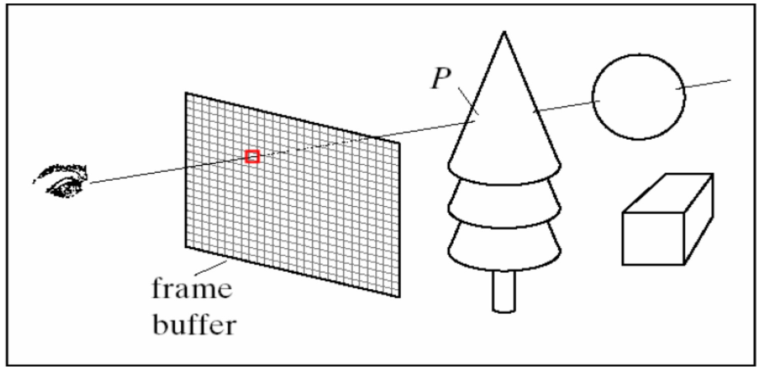
Ray Tracing

Vignesh A C

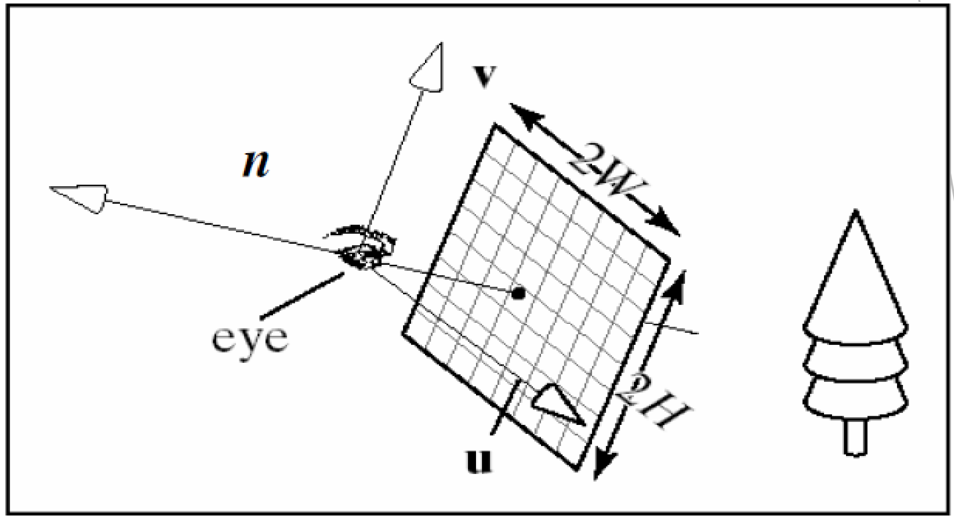
Introduction

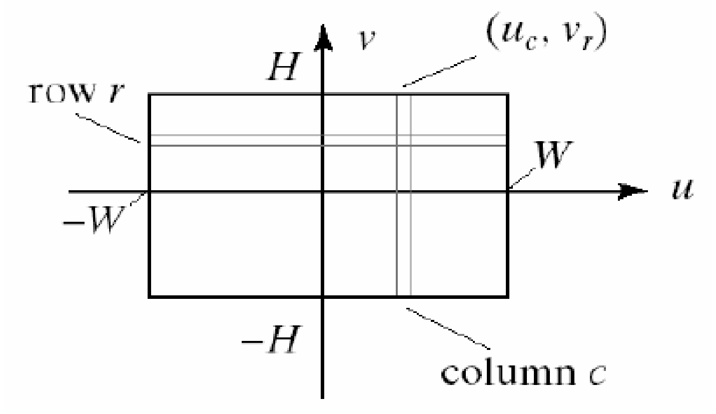
- Goruaud and Phong interpolation schemes
- Ray tracing is similar, but even more powerful.
- The conventional way – a ray of light hits the human eye.
- In actuality, the process is a slight variation
  - A ray is thought to be cast from eye through the pixel center and out into the scene.
  - Path tested against the scene to see which object is hit first, and at what point
  - Parameter t, can be considered equivalent to time
  - Automatically removes the “hidden surface removal problem”
- Visual effects – shadowing, reflection, refraction

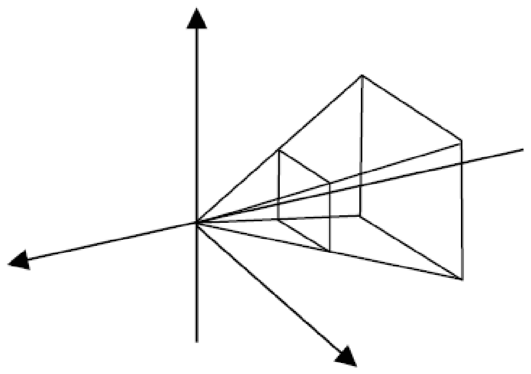


Setting up the geometry

- The three vectors
  - u and v – perpendicular to n and to each other
  - n – parallel to the direction of viewing
- Frame buffer lies in the near plane
- Other terminologies
  - N – distance of near plane from eye
  - aspect – Aspect ratio
  - θ – view angle in the near plane
- The near plane
  - -H to H in v-direction and –W to W in u-dir
- H = N x tan(θ/2)
- W = H x aspect







- Separate viewport transformation is not necessary. Here, the near plane is itself the viewport.
- Pixels –
  - 0 to (nRows – 1) rows : r
  - 0 to (nCols – 1) columns : c
  - Each pixel is denoted as rcth pixel
- Each pixel has lower left corner at (u_c, v_r)
  - $u_c = -W + W \times 2c / nCols$, for $c = 0,1,\ldots,nCols - 1$
  - $v_r = -H + H \times 2r / nRows$, for $r = 0,1,\ldots,nRows - 1$
- Equation of the 3D point in space
  - $eye - N\mathbf{n} + u_c\,\mathbf{u} + v_r\,\mathbf{v}$ <- location of pixel corner in space
- Parametric expression for the ray
  - $r(t) = eye(1-t) + pixelcorner\ t$
  - $r(t) = eye(1-t) + (eye - N\mathbf{n} + u_c\,\mathbf{u} + v_r\,\mathbf{v})\,t$ (substituting for pixel corner)
- $r(t) = eye + dir_{rc}t$
  - $dir_{rc} = -N\mathbf{n} + W\left(\frac{2c}{nCols} - 1\right)\mathbf{u} + H\left(\frac{2r}{nRows} - 1\right)\mathbf{v}$

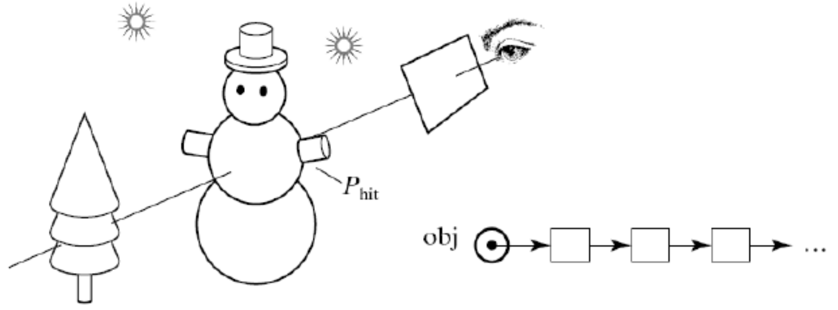
Overview of Ray Tracing

- The scene to be ray-traced is inhabited by various geometric objects(spheres, cones, boxes, cylinders) and light sources
- For each pixel in the window, we construct a ray that starts at the eye and passes through the lower left corner of the pixel. This involves simply evaluating direction dir_{rc} for the rc^th array.
- Pseudocode

```
<define the objects and light sources in the scene>
<set up the camera(eg., as in gluLookAt)>
for (int r = 0; r < nRows; r++)
    for (int c = 0; c < nCols; c++) {
      < 1. Build the rcth array >
      < 2. Find all intersections for the rcth ray with objects in the scene >
      < 3. Identify the intersection that lies closest to, and in front of, the eye >
      < 4. Compute the hit point where the ray hits the 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. Place the color in the rcth pixel >
    }
```

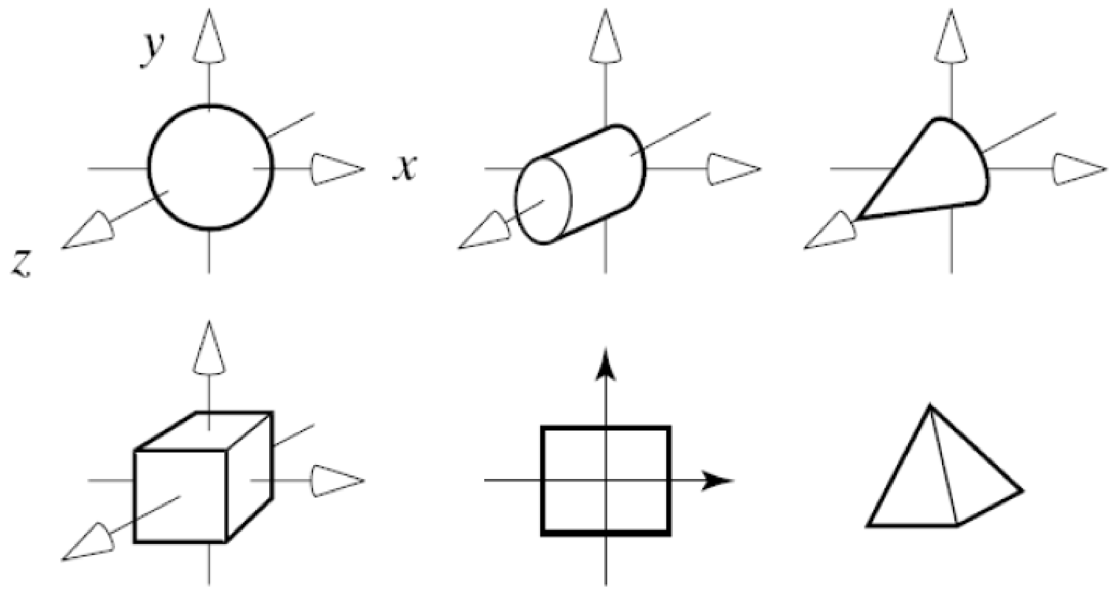
(contd.)

- Step 3, elaborated..
  - For each intersection of rc^th ray with an object, note the "hit time" value (t_{hit})
  - Object with smallest hit time is closest to the eye
- The location of the hit point is then found, along with normal vector. The color of the light that reflects off this object, in the direction of the eye is then computed and stored in the pixel.
- Objects are maintained as a list, and the intersection point is computed for each object
- The hit spot, P_{hit} is computed as P_{hit} = eye + $dir_{r,c}t_{hit}$
- In some cases, a ray will not hit any object in the object list
  - which means, it contributes only background light to the relevant pixel.



Intersection of a ray with an object

- Remember the Scene Description Language??
- The various objects in the scene are retrieved and placed in a list.
  - Each object is an instance of a generic shape along with transformations applied to it.



(contd.)

- Implicit form for each shape
  - Generic sphere - a sphere of radius 1 centered at origin
    $F(x, y, z) = x^2 + y^2 + z^2 - 1$
    $F(P) = |P|^2 - 1$
  - Generic cylinder: Aligned along z-axis, with longitudinal axis of length 1
    $F(x, y, z) = x^2 + y^2 - 1 = 0$ for $0 < z < 1$
  - In a real scene, an object may be subject to transformations.
  - Finding the intersection point with implicit form of shape will still suffice!
- Let the ray of light have starting point S and direction c
  $r(t) = S + \mathbf{c}t$
- All the points on the shape surface satisfy the equation $F(P) = 0$.
  Hence, $F(r(t)) = 0 \rightarrow F(S + \mathbf{c}t_{hit}) = 0$
- The task is to solve this equation efficiently.

Intersection of a Ray with the Generic Plane

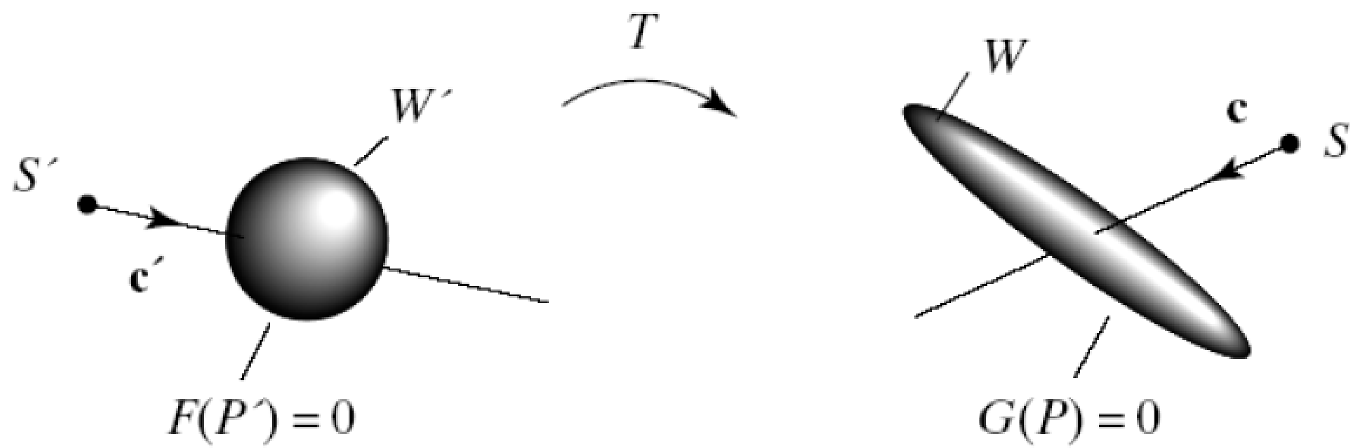
- In any scene, walls, floor and ceiling are represented as planes.
- They can have a plain color or a textured pattern.
- Generic plane is the xy plane, i.e., $z = 0$
  Thus the implicit form becomes $F(x, y, z) = z$
- The ray $S + \mathbf{c}t$ intersects the plane when $S_z + c_z t_h = 0$
- Solution of the equation becomes
  $t_h = - S_z / c_z$
- What happens when $c_{z\,=}\ 0$?
- In general, the ray hits the plane at the point
  $P_{hit} = S - \mathbf{c}\,(S_z / c_z)$
- Where does the ray below hit the generic plane?
  $r(t) = (4,1,3) + (-3,-5,-3)t$

Intersection of a Ray with the Generic Sphere

- $|S + ct|^2 - 1 = 0$
- This makes it $|c|^2t^2 + 2(S.c)t + |S|^2 - 1 = 0$, which is a quadratic equation
  $A = |c|^2$, $B = 2(S.c)$, $C = |S|^2 - 1$
- Solving the equation above, we get
  $t_h = (-B \pm \sqrt{(B^2 - 4AC)}) / 2A$
- If the discriminant is negative, the ray misses the sphere, if zero, the ray grazes the sphere at one point. If positive, then there are two hit times.
- Proceed with the next step as in previous case.

Intersection of the Ray with Transformed Objects

- Say the transform T maps a generic sphere W' into an ellipsoid W
- If the original generic object has the implicit function F(P) then the transformed version has implicit function $F(T^{-1}(P))$
  - T^{-1} is the inverse transformation to T
  - If T has the matrix M then T^{-1} has the matrix M^{-1}
- Thus we need to solve $F(T^{-1}(S + ct)) = 0$ for the hit time t. Because the transformation T is linear, the inverse transformed array is $T^{-1}(S + ct) = (T^{-1}S) + (T^{-1}c)\, t = S' + c't$



(contd.)

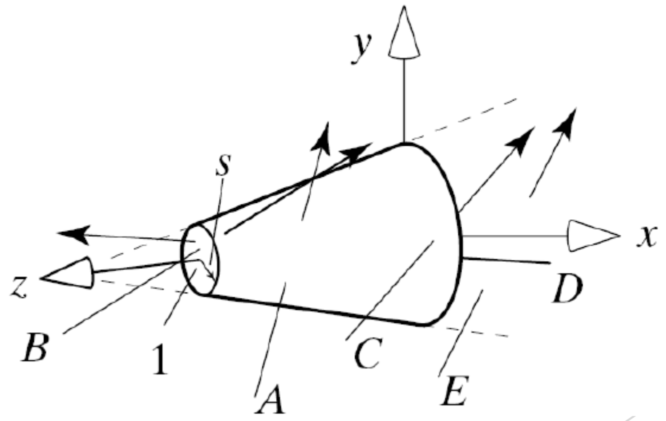
- Using homogenous coordinates, we end up in the following equation for the inverse transformed ray.

$$\tilde{r}(t) = M^{-1}\begin{pmatrix} S_x \\ S_y \\ S_z \\ 1 \end{pmatrix} + M^{-1}\begin{pmatrix} c_x \\ c_y \\ c_z \\ 0 \end{pmatrix} t = \tilde{S}' + \tilde{c}'t$$

- Which means, instead of transforming the object first and then finding the intersection points, we find the intersection points with reference to a generic object and use an inverse transformed ray to intersect the generic object.
- General procedure:
  - Inverse transform the ray (obtaining S' + c't)
  - Find its intersection time t_h with the generic object
  - Use the same t_h in S + ct to identify the actual hit point
- Why do we do this?

Intersecting Ray with Other Primitives

- Intersecting with generic square
  - The generic square lies in the z = 0 plane and extends from –1 to 1 in both x and y.
  - Its implicit form is $F(P) = P_z$ for $|P_x| \leq 1$ and $|P_y| \leq 1$
  - Can be transformed into any parallelogram positioned in space, commonly used for doors and windows.
- Intersecting with cylinder
  - One side of a cylinder is said to be a part of an infinitely long wall
  - Radius of 1 at z=0 and a small radius at z=1
  - Implicit form equation is $F(x, y, z) = x^2 + y^2 + (1 + (s - 1)z)^2$
  - Now, s = 0 → it becomes a generic cone and s = 1 → it becomes generic cylinder
  - Figure beside shows different possible intersections



(contd.)

- In case of multiple intersections, store them in an array, and swap the contents if they are out of order
- Individual tests are straight forward. Substitute S + ct in the implicit form
- We get an equation of the form $At^2 +\ 2Bt + C$ where

$$A = c_x^2 + c_y^2 - d^2$$

$$B = S_x c_x + S_y c_y - Fd$$

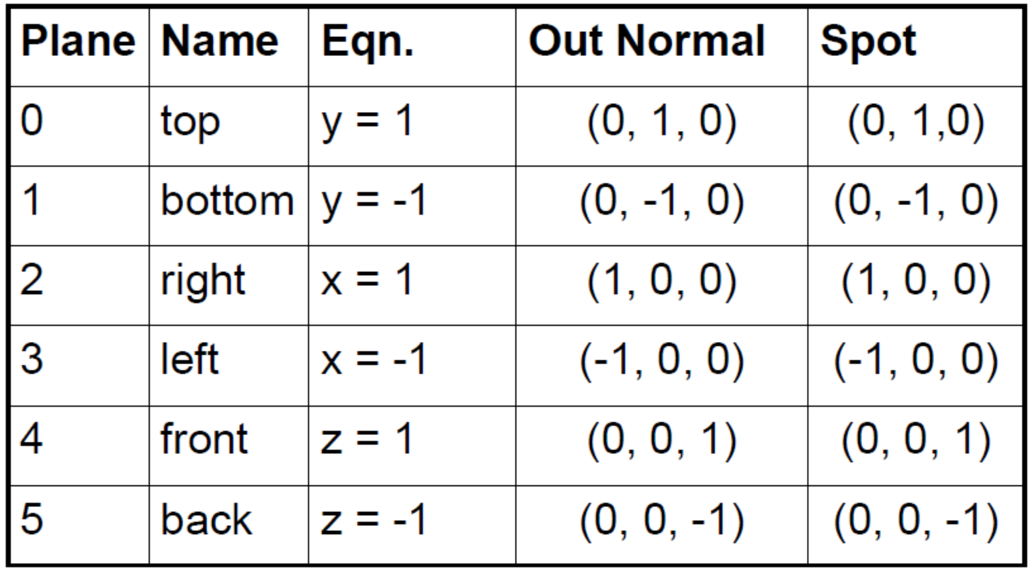
$$C = S_x^2 + S_y^2 - F^2$$

where $d = (s - 1)\, c_z$
and $F = 1 + (s - 1)\, S_z$

- Based on the values of discriminant (<0, =0 or >0), the intersection of ray with various walls/surfaces can be found.

- Intersecting Rays with a cube (or any convex polyhedron)
  - Generic cube – centered at origin and passing through (±1, ±1, ±1)
  - Used for modeling generic boxes
  - Also acts as a bounding surface for many other primitives

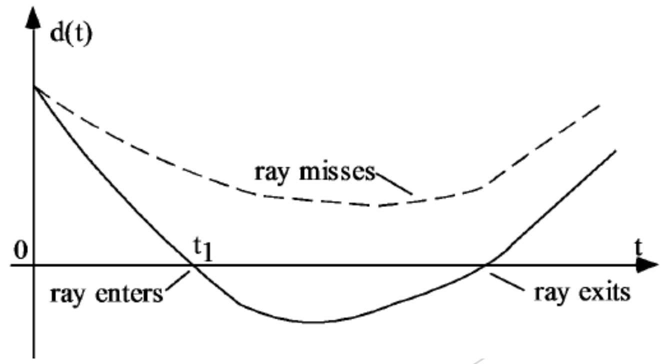
- The six planes



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)

- The process of intersecting a ray with a cube is essentially the Cyrus-Beck clipping algorithm, in which a line is clipped against a convex window in 2D space.
- Candidate Interval (CI) – the interval in which the ray could be inside the object CI = $[t_{in}, t_{out}]$
- An entering ray : for all $t < t_{in}$, the ray is outside the cube
  An exiting ray : for all $t > t_{out}$, the ray is outside the cube
- CI empty → ray does not enter the cube

- Intersection with Other Primitives
  - We only need to know the implicit form *F(P) of the shape*.
  - Then, as before, to find where the ray *S* + ***ct intersects the surface, we substitute S + ct for P in F(P) forming a function of time t:***
    d(t) = F(S + ct), which is
    - positive at those values of ***t for which the point on the ray is outside the object***,
    - zero when the ray coincides with the surface of the object,
    - negative when the ray is inside the surface.
  - Intersecting a ray is equivalent to solving this equation
    d(t) = 0
  - For a torus, it leads to a fourth order implicit function
  - Any generic shape has a graph which looks like the one beside
  - Intersecting curve : d(t) decreases from +ve, crosses 0, becomes -ve and then vice versa
  - Non intersecting curve : d(t) decreases to a +ve value then increases back



Thank you! ☺