The Z-buffer algorithm

- Belongs to the class of image-precision algorithms

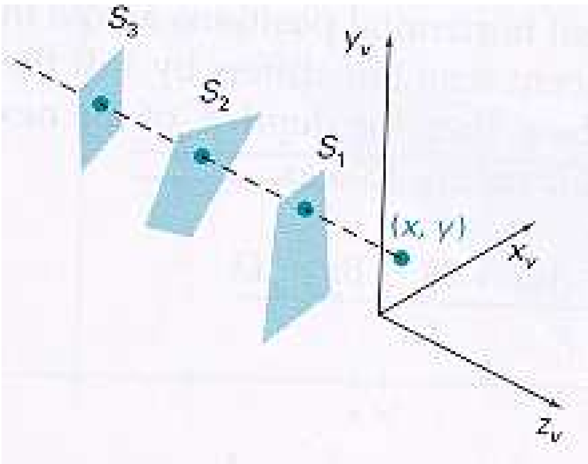
- Uses a buffer (z-buffer) to store a z-value (depth) for each pixel

- Very efficient when the objects consist of planar surfaces (depth values can be computed quickly)

- **Algorithm**

Initialize: Z(x,y)=0 (Z-buffer), F(x,y)=background-color (frame-buffer)
For each polygon in the model
For each pixel in the polygon
Calculate pixel color
Calculate pixel z-value
Compare pixel z-value to the previous value stoted in the Z-buffer
If pixel is closer, draw it in the frame-buffer and Z-buffer

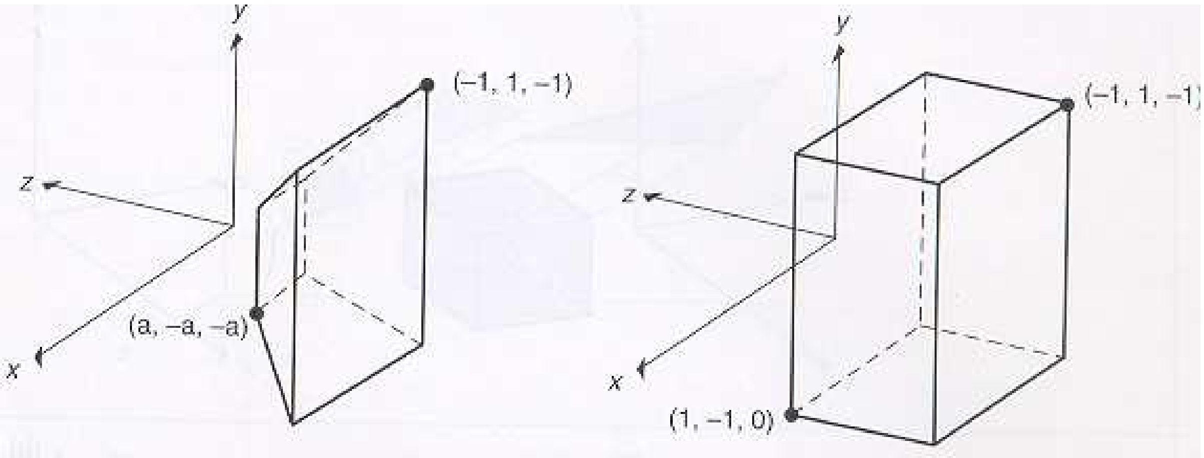
- Eventually, Z(x,y) contains the depth value which corresponds to the surface closest to the view plane



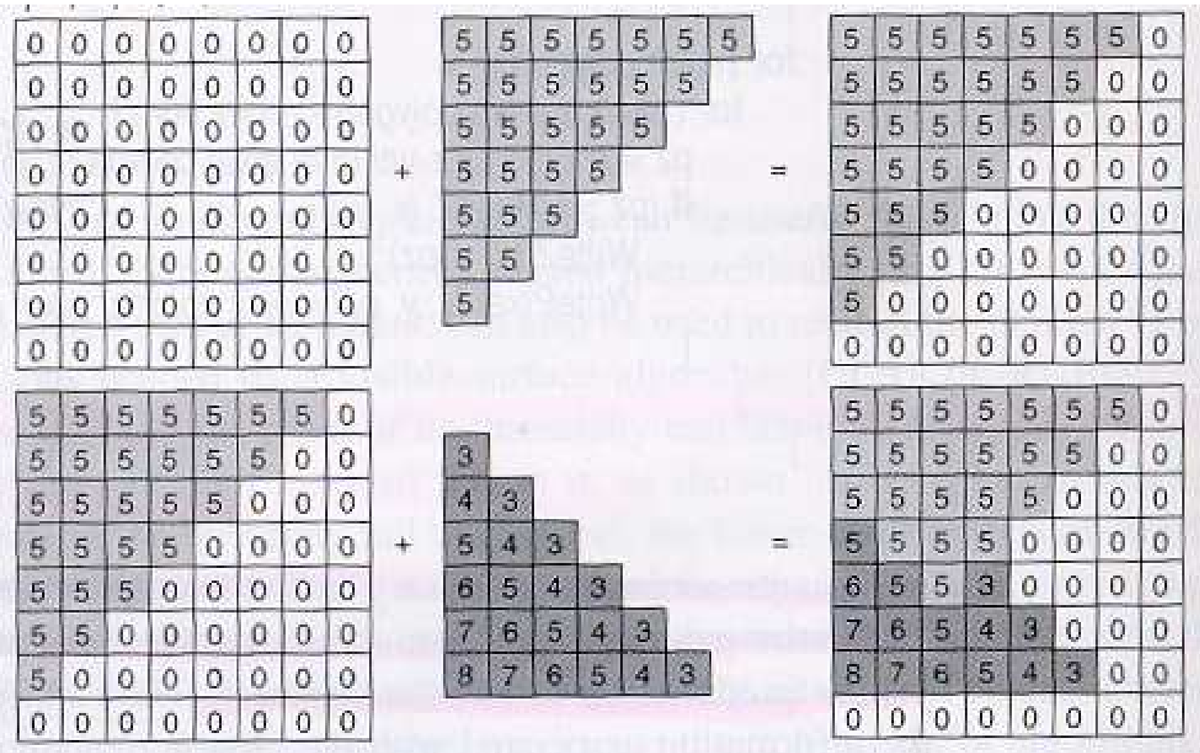
- **Min-max values for the Z-buffer**

- min z-value: determined by the back clipping plane (z=0)

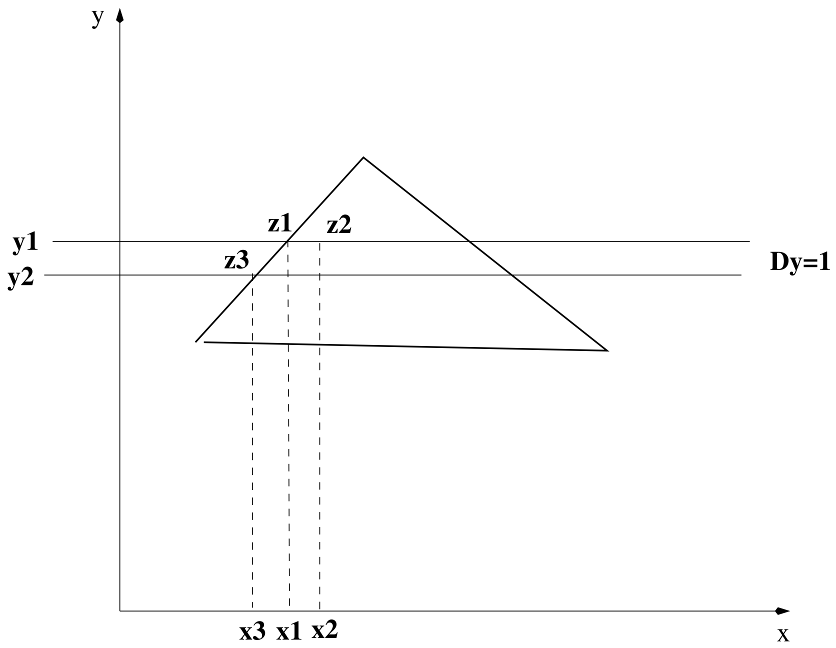
- max z-value: determined by the front clipping plane (z= -1)



- **Example**



- **Depth calculations for planar surfaces**



- Use depth coherence to simplify calculations

- Plane Equation: $Ax + By + Cz + D = 0 \Rightarrow z = \dfrac{-Ax - By - D}{C}$

- Same scan-line:

$$z_1 = \frac{-Ax_1 - By_1 - D}{C}, \quad z_2 = \frac{-Ax_2 - By_1 - D}{C}$$

$$z_2 - z_1 = -\frac{A}{C}(x_2 - x_1) = -\frac{A}{C}\Delta x$$

If Δx=1, then $z_2 = z_1 - A/C$

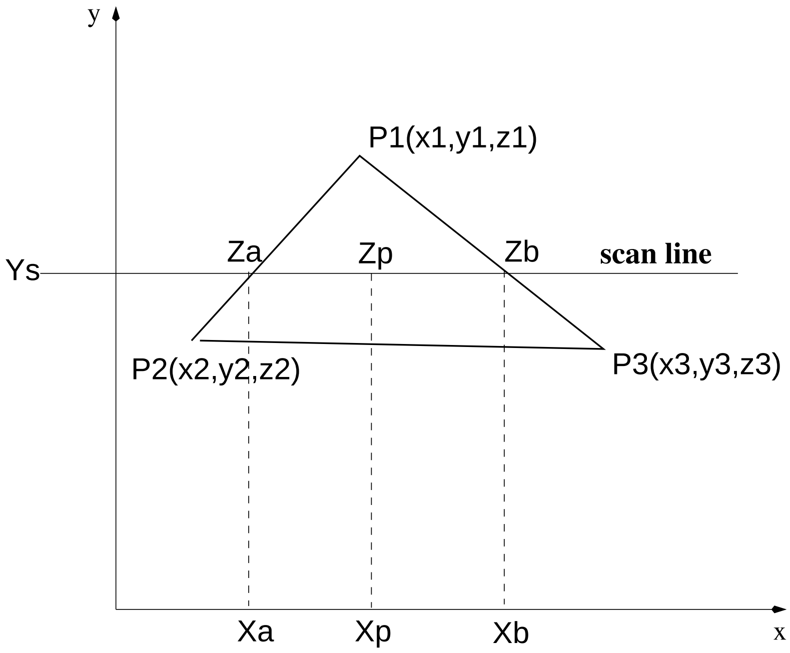
- Different scan-line:

$$z_1 = \frac{-Ax_1 - By_1 - D}{C}, \quad z_3 = \frac{-Ax_3 - By_2 - D}{C}$$

$$z_3 - z_1 = \frac{-A(x_3 - x_1) + B(y_2 - y_1)}{C}$$

If Δy=1, then $z_3 = z_1 - \dfrac{A/m + B}{C}$

- **Depth calculations using interpolation**



- Compute z_a:

$$z(t) = z_1 + t(z_2 - z_1)$$

$$y(t) = y_1 + t(y_2 - y_1)$$

If $y(t) = y_s$, then $t = \frac{y_s - y_1}{y_2 - y_1}$

Thus, $z_a = z_1 + \frac{y_s - y_1}{y_2 - y_1}(z_2 - z_1)$

- Compute z_b: similarly

- Compute z_p:

$$z(t) = z_a + t(z_b - z_a)$$

$$x(t) = x_a + t(x_b - x_a)$$

If $x(t) = x_p$, then $t = \frac{x_p - x_a}{x_b - x_a}$

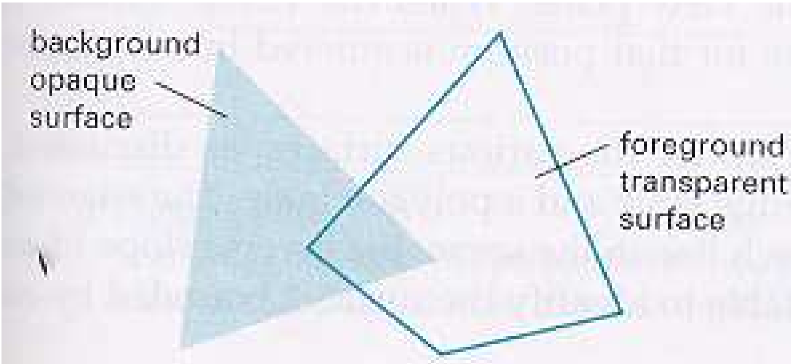
Thus, $z_p = z_a + \frac{x_p - x_a}{x_b - x_a}(z_b - z_a)$

- **Extensions of the z-buffer algorithm**

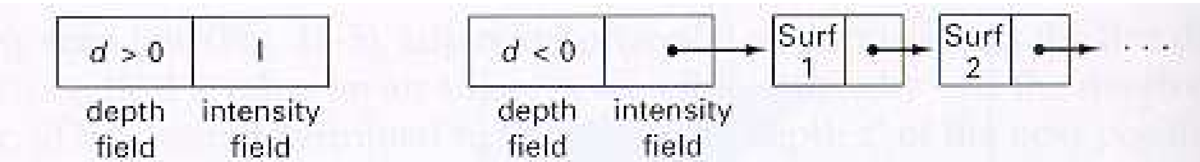
- The contents of the z-buffer can be saved and used later to merge in new objects

- If memory is an issue, one scan-line can be processed each time

- Must be modified when the surfaces are not opaque



- The "A-buffer" algorithm is used for transparent surfaces



- Stores a linked list to surface data

- The intensity of each pixel is the average of the contributions from the overlapping surfaces

Scan-line algorithms

- Belong to the class of image precision algorithms

- Idea: create one scan-line of the image at a time

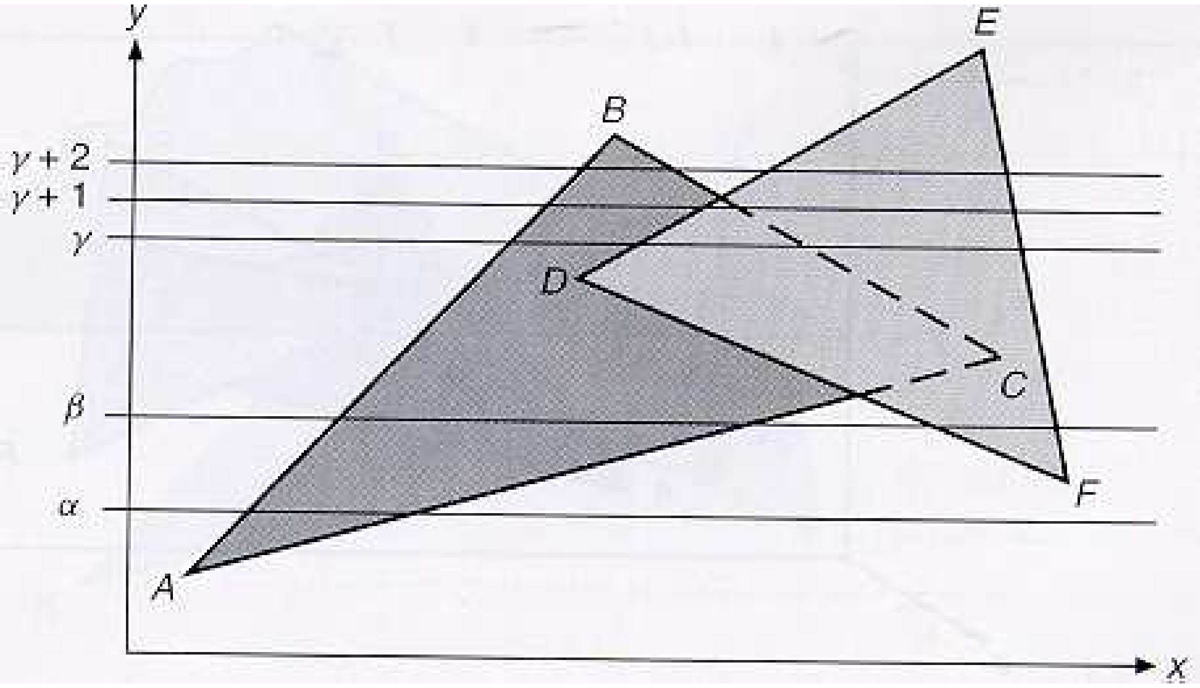
- It is based on an extension of the polygon scan-conversion algorithm (it handles multiple polygons)

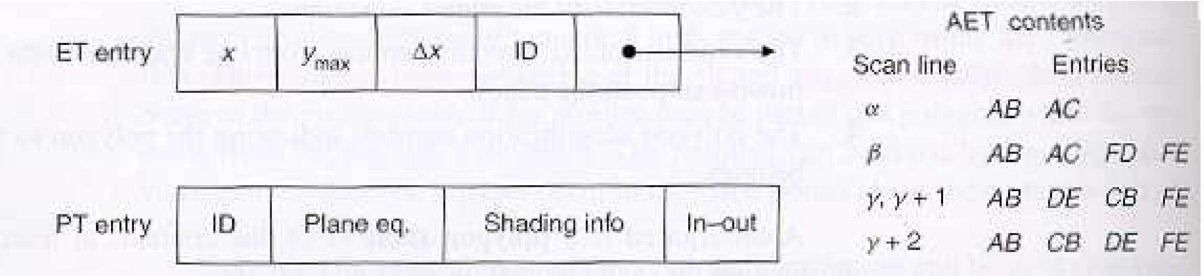
• Algorithm

For each scan-line in image
For each pixel in scan-line
Determine the closest object
Calculate pixel color, draw pixel

• Data structures

- Polygon Table (PT), Edge Table (ET), Active Edge List (AEL)





- **Steps**

  - Process the edges from left to right (using the AEL)

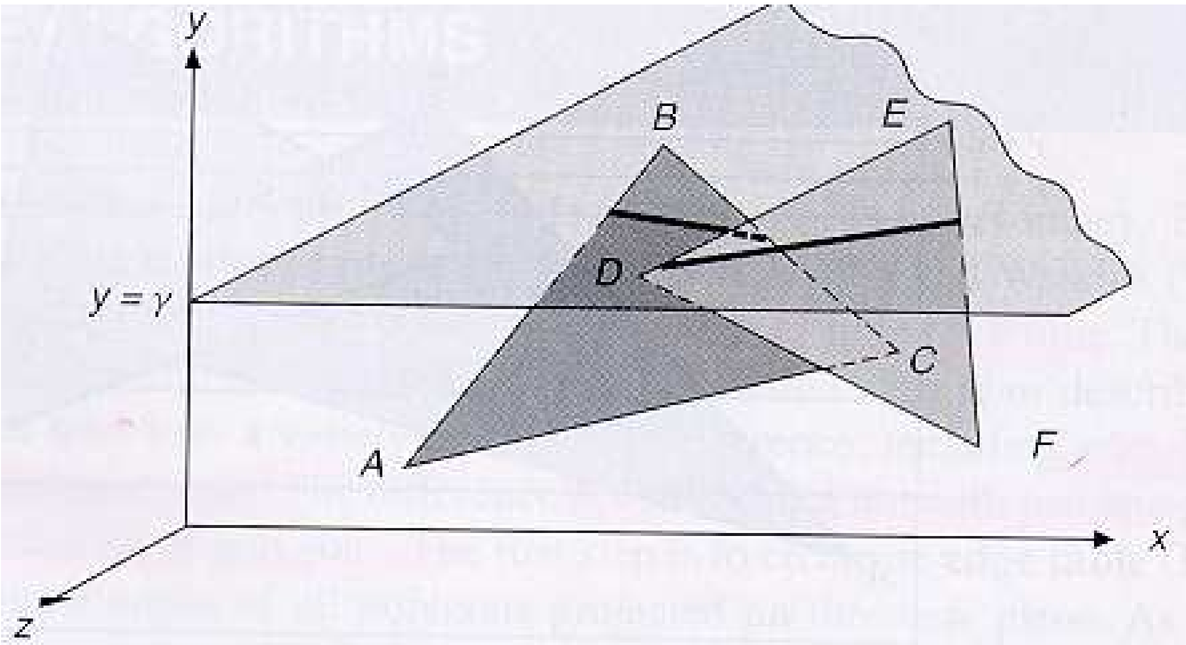
  - Invert the *in-out* flag from *false* to *true* when the scan-line enters the corresponding polygon

  - Invert the *in-out* flag from *true* to *false* when the scan-line exits the polygon

  - If there is only one *in-out* flag whose value is *true*, then the scan-line is in one polygon

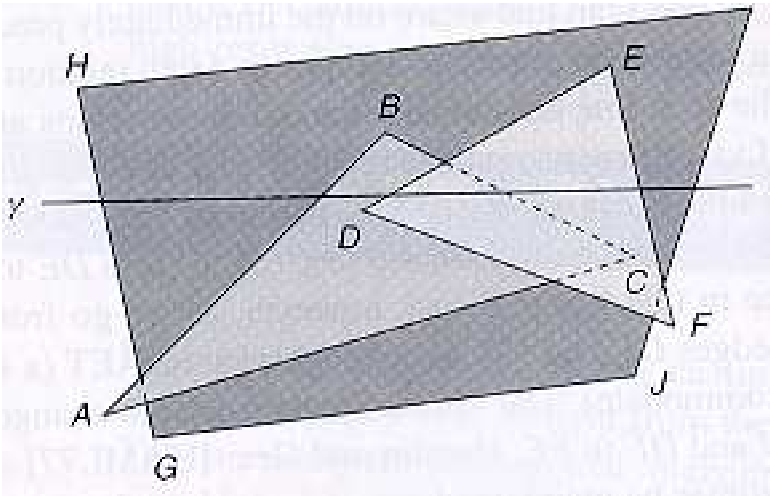
  - If more *in-out* flags are *true*, then the scan-line is in two or more polygons

  - Find $z_{ABC}(x, \gamma)$ and $z_{DEF}(x, \gamma)$ using the plane equations from the PT



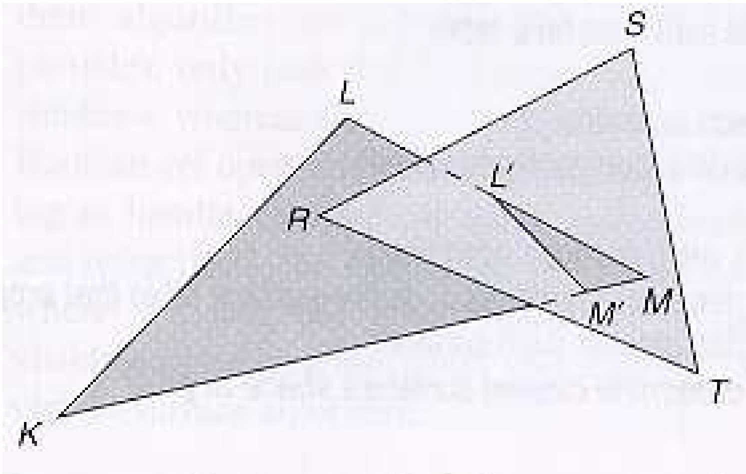
- **Improving the scan-line approach**

  - Depth computations are unecessary when the scan-line leaves an <u>obscured</u> polygon, and are required only when it leaves an <u>obscuring</u> polygon



  - Penetrating plygons

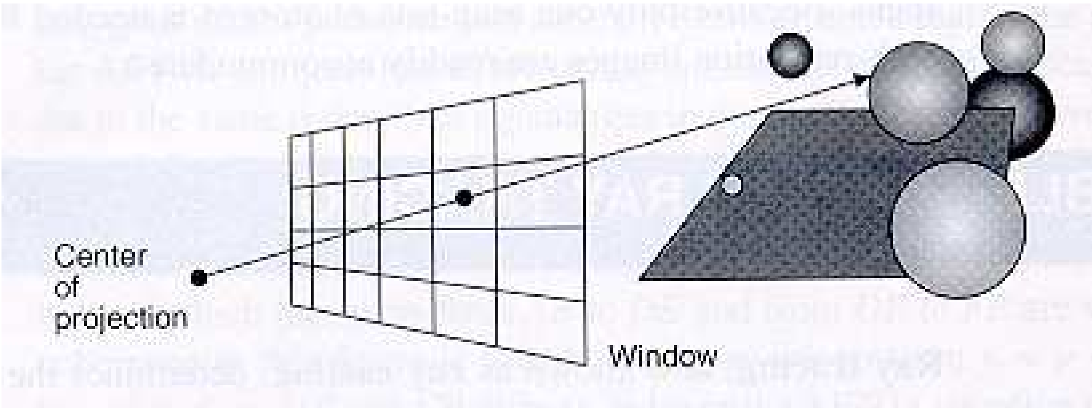
    Break KLM : $KLL'M'$ and $L'MM'$



Ray tracing (or ray casting)

- It belongs to the class of image precision algorithms

- Determines the visibility of surfaces by tracing imaginery rays of light from the viewer's eye to the objects in the scene



- **Algorithm**

Select center of projection and window on viewplane
For each scan line in image
  For each pixel in scan-line {
    Determine ray from center of projection through pixel
    For each object in scene {
      If object is intersected and is closest considered thus far
        Record intersection and object name
    }
    Set pixel's color to that at closest object intersection
  }

- **Ray tracing versus z-buffer**

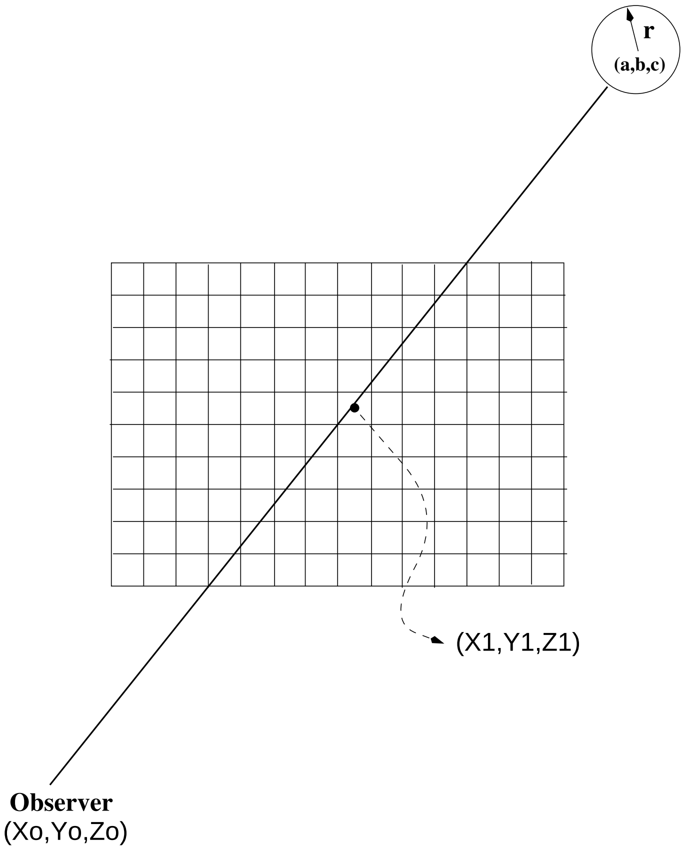
- The z-buffer approach processes surfaces one at a time and calculates depth values for all projection points over the surface

- The ray tracing approach process pixels at one at a time and calculates depths for all surfaces along the projection path

- **Computing intersections with a sphere**

- The most important element of a visible surface ray tracing algorithm is the intersection routine

- Ray tracing is very effective for scenes with curved surfaces, particularly spheres



$$x = x_0 + t(x_1 - x_0) = x_0 + t\Delta x$$

$$y = y_0 + t(y_1 - y_0) = y_0 + t\Delta y$$

$$z = z_0 + t(z_1 - z_0) = z_0 + t\Delta z$$

<u>Sphere's equation:</u> $(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$

$x^2 - 2ax + a^2 + y^2 - 2by + b^2 + z^2 - 2cz + c^2 = r^2$ =>

$$(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2 + (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2 + (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2$$

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] + (x_0 - a)^2 + (y_0 - a)^2 + (z_0 - a)^2 - r^2 = 0$$

* No real roots: ray and sphere do not intersect
* One real root: the ray is tangent to the sphere
* Two real roots: choose the point with the smallest t

- **Computing intersections with a polygon**

- First, determine whether the ray intersects the polygon's plane

- Find whether the point of intersection lies within the polygon

Plane's equation: $Ax + By + Cz + D = 0$

or $A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$

or $t = -\dfrac{Ax_0 + By_0 + z_0 + D}{A\Delta x + B\Delta y + C\Delta z}$

(If $A\Delta x + B\Delta y + C\Delta z$=0, the ray does not intersect the plane !!)

- **Optimizing intersection calculations**

- Compute intersections with bounding volumes first

- Spatial partitioning