Math Basics

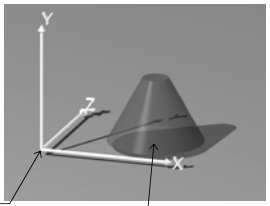
Coordinate Systems, Vectors

Coordinate Systems

- Points in 3D are defined by (x,y,z)
- Where are the (x,y,z) values coming from? With respect to what?
  - Origin
  - Where is it located?

Coordinate Systems

- In CG, objects are usually placed with respect to the World coordinate system

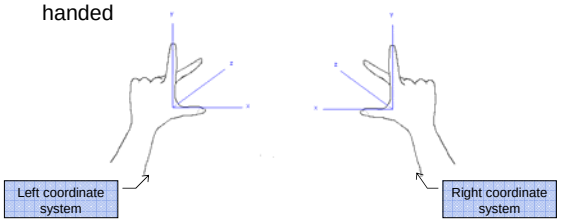


World coordinate system (0,0,0)

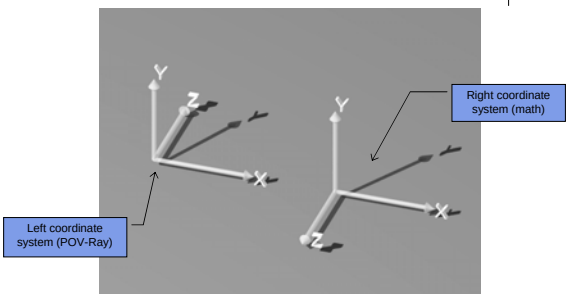
Center of cone located at (3,0.2,2)

Coordinate Systems

- In POV Ray, the coordinate system is Left handed
- In the standard mathematical system it is Right handed

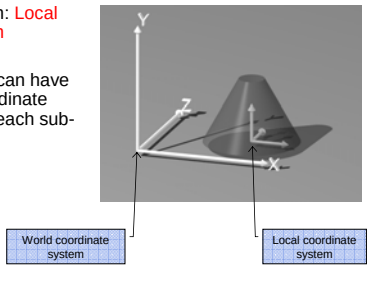


Coordinate Systems



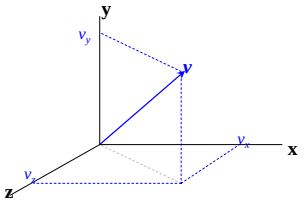
Coordinate Systems

- Objects can also be represented in their own coordinate system: Local coordinate system
- Complex objects can have several local coordinate systems: one for each sub-component

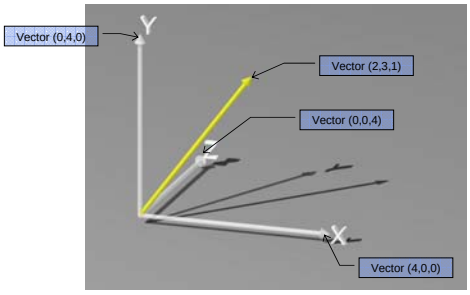


Vectors

- In 3D, a vector is defined by 3 scalar numbers: (v_x, v_y, v_z)
- Vector length: $|v| = \sqrt{v_x^2+v_y^2+v_z^2}$

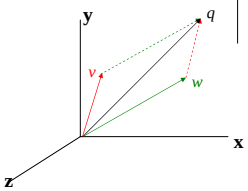


Vectors



Vector Algebra

- Addition
  - $q = v + w$

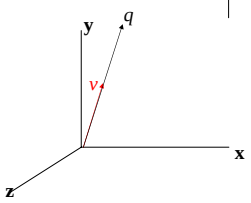


q is obtained by component-wise addition:

$q_x = v_x + w_x$, $q_y = v_y + w_y$, $q_z = v_z + w_z$

Vector Algebra

- Scaling
  - Scalar number s.
  - $q = sv$



q is defined by component-wise scaling:

$q_x = sv_x$, $q_y = sv_y$, $q_z = sv_z$

Dot Product

Given 2 vectors: u and v, where:

$u = (u_x, u_y, u_z)$

$v = (v_x, v_y, v_z)$

Their dot product (or "inner product") is a scalar number obtained as the sum of the component-wise products:

$u \cdot v = u_x v_x + u_y v_y + u_z v_z$

Dot Product

Given 2 vectors: u and v, where:

$u = (u_x, u_y, u_z)$

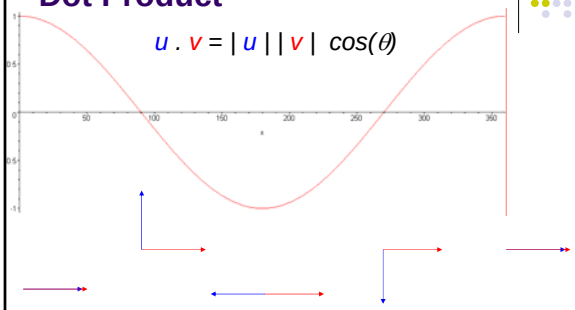
$v = (v_x, v_y, v_z)$

Their dot product (or "inner product") is a scalar number obtained as the sum of the component-wise products:

$u \cdot v = u_x v_x + u_y v_y + u_z v_z$

Dot Product

$u \cdot v = |u||v| \cos(\theta)$



Dot Product

- Properties
  - Symmetric: $u.v = v.u$
  - Bilinear: $v.(u + aw) = v.u + a(v.w)$
  - Non-degenerate: $v.v = 0$ only if $v = 0$.
  - If v and u are orthogonal: $v.u = 0$

Dot Product

- Positive
  - If angle is less than 90
- Negative
  - If angle is more than 90

Cross Product

- The cross product of two vectors produces a third vector which is perpendicular (orthogonal) to the plane in which the first two lie

Cross Product

Given 2 vectors: u and v, where:

$u = (u_x, u_y, u_z)$

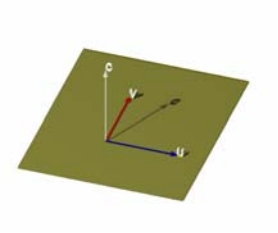
$v = (v_x, v_y, v_z)$

Their cross product (or "outer product") is a new vector $c=(c_x, c_y, c_z)$ computed as follows:

$c_x = u_y v_z - u_z v_y$

$c_y = u_z v_x - u_x v_z$

$c_z = u_x v_y - u_y v_x$



More Concepts in Graphics

The Frame Buffer

The Frame Buffer

- The portion of memory reserved for holding the complete bit-mapped image that is sent to the monitor
- Typically the frame buffer is stored in the memory chips on the video adapter. In some instances, however, the video chipset is integrated into the motherboard design, and the frame buffer is stored in general main memory

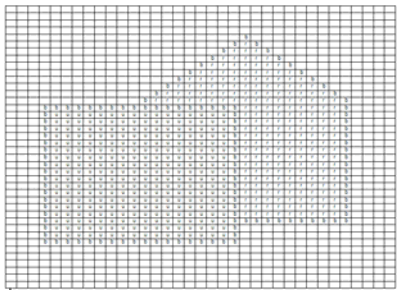
The Frame Buffer

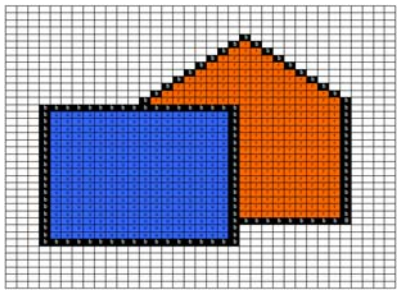
- For each pixel there is an entry in the frame buffer which holds the color information for that pixel
- Changing the contents of the frame buffer, changes the image on the screen
- The Graphics Controller sends the frame buffer info to the computer monitor
  - Typically, 60 Hz (60 times per second)

The Frame Buffer

- Example
  - A 35x40 pixel image

b = black
u = blue
r = orange





The Frame Buffer

- Example
  - Screen resolution: 1280 x 1024 pixels
  - 256 colors per pixel
- Size of the frame buffer?
  - How much space do you need to represent 256 colors?
    - 1 byte (8 bits, 2^8=256)
    - So you need 1 byte per pixel to store the color info
  - 1280x1024x1 = 1310720 bytes (1.3 Mb approx)

Surface Algorithms

What is visible?

Far, Behind, Away

- In order to properly display 3D objects on 2D screens, we need to deal with additional issues
  - How “far” is the object from the viewer?
  - Which objects are “behind” others?
  - How are the objects oriented relative to the viewer? (“away”)

Far, Behind

- Which one is closer?
- Which one is behind the other?
- We need to know that so that the frame buffer is filled out with the correct information

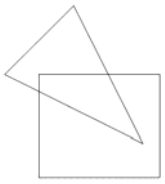


Figure 3-7: Overlapping polygons

Far, Behind

- Knowing the z coordinate of the objects is essential

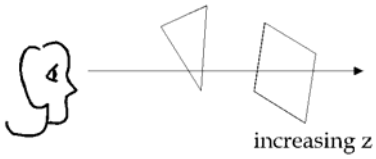
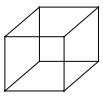


Figure 3-8: Distance from viewer

Away

- Why is it important to know the relative orientation of the objects?
  - Lighting
  - Visibility

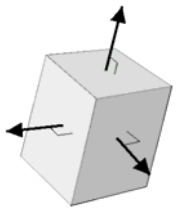


How can we determine the orientation?

- Normal vector

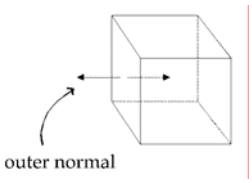
Normal Vector

- Vector perpendicular to a plane
- In CG, every polygon has a normal vector (a polygon is a subsection of a plane, right?)
- Tells you which way a polygon is facing

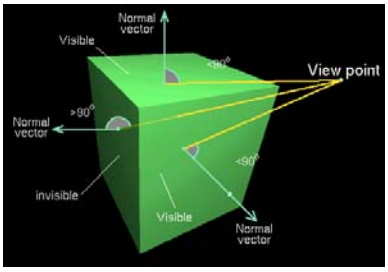


Polygon Normal

- The normal to a polygon can have two directions
- We are interested in the outer normal



Polygon Normal



The Algorithms

Four Basic Algorithms

- Wireframe
- Hidden line
- Z-buffer
- Ray tracing

Visual Literacy

- Things to look for
  - Polygon appearance
  - Visibility of far sides, backgrounds, and horizons
  - Reflection, refraction, and shadows

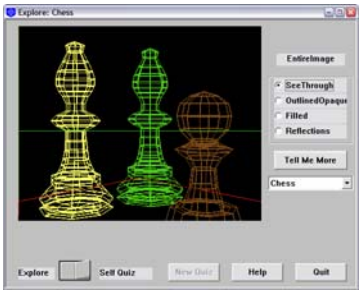
Wireframe

- Creates a line drawing of a model by connecting its vertex points
- Very fast and is usually used to preview work in the modeling window
- Ignores lighting and doesn't distinguish between hollow or solid areas

Wireframe

- Visual cues
  - Outlined polygons
  - Horizons, background objects completely visible

Wireframe



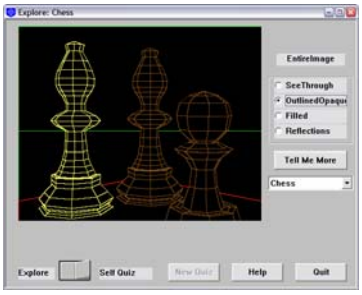
Hidden Line

- Draws the polygons same as Wireframe
- Removes some of the ambiguities by drawing only the parts visible to the viewer

Hidden Line

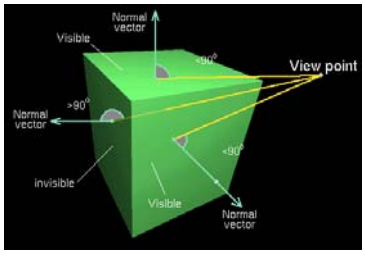
- Visual cues
  - Outlined polygons
  - Occluded objects are invisible

Hidden Line

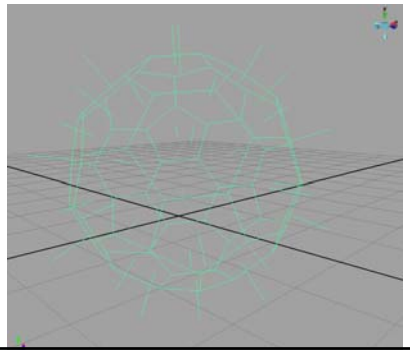


Backface Cull

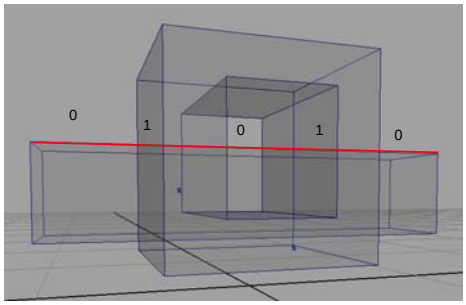
- Throws away all polygons that face away from the viewer.



Backface Cull



Appel's Algorithm



Z-Buffer (Depth buffer)

- This is the method most commonly used in graphics workstations. Each polygon is rendered in turn and its colors written into a frame buffer
- The distance to the surface is also calculated for each pixel rendered: these distance values are stored in a separate frame buffer store. As distance is usually the z-coordinate, this is called the z-buffer

Z-Buffer Algorithm

```
for each x,y
   color[x,y] := background_color
   z-buffer[x,y] := 0

for each polygon
   for each pixel (x,y) that polygon projects to
        z := z-value of polygon at pixel (x,y)
        if z >= z-buffer[x,y]
                color[x,y] := polygon's color at pixel (x,y)
                z-buffer[x,y] := z
```

Z-Buffer

- maintains color and z-value for each pixel
- polygons are drawn one at a time

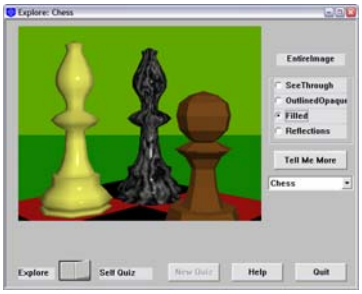
- can be done incrementally
- does not require object to object comparisons
- renders visible surfaces very fast
- widely implemented in both software and hardware

- Z buffering does not address how light interacts between objects

Z-Buffer

- Visual cues
  - Filled-in polygons
  - No refraction, reflection or shadow

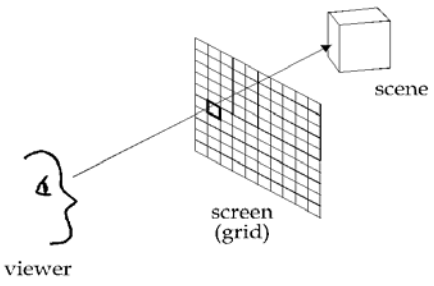
Z-Buffer



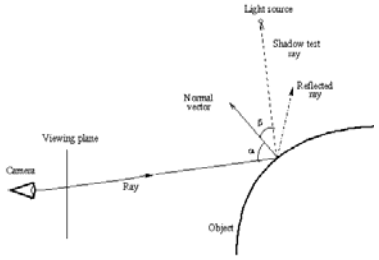
Ray Tracing

- Rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world
- A ray of light is traced in a backwards direction. That is, we start from the eye or camera and trace the ray through a pixel in the image plane into the scene and determine what it hits. The pixel is then set to the color values returned by the ray

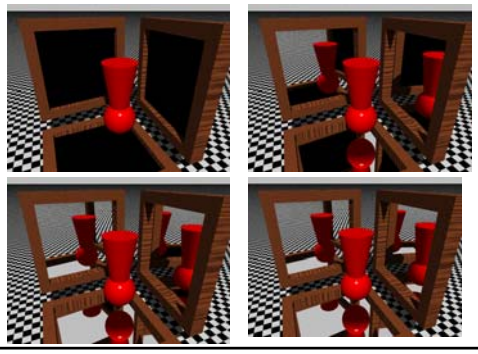
Ray Tracing



Ray tracing



Ray Tracing – Trace Level



Ray Tracing

- Visual cues
  - Solid polygons
  - Refraction, Reflection, Shadows

Ray Tracing

References

- http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm
- Foley, van Dam, Feiner, Hughes. Computer Graphics, Principles and Practice, Addison-Wesley, 1997