Interactive Computer Graphics

- The Graphics Pipeline: Clipping

Some slides adopted from
F. Durand and B. Cutler, MIT

The Graphics Pipeline

Modelling Transformations
Illumination (Shading)
Viewing Transformation (Perspective / Orthographic)
Clipping
Projection (to Screen Space)
Scan Conversion (Rasterization)
Visibility / Display

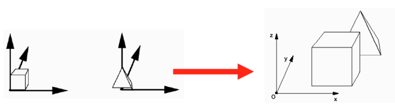
Input:
- geometric model
- illumination model
- camera model
- viewport

Output: 2D image
for framebuffer display

The Graphics Pipeline

Modelling Transformations
Illumination (Shading)
Viewing Transformation (Perspective / Orthographic)
Clipping
Projection (to Screen Space)
Scan Conversion (Rasterization)
Visibility / Display

- 3D models are defined in their own coordinate system
- Modeling transformations orient the models within a common coordinate frame (world coordinates)

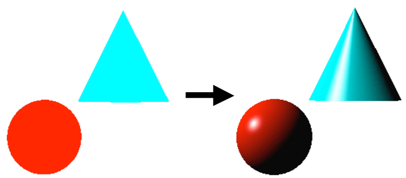


Object space World space

The Graphics Pipeline

Modelling Transformations
Illumination (Shading)
Viewing Transformation (Perspective / Orthographic)
Clipping
Projection (to Screen Space)
Scan Conversion (Rasterization)
Visibility / Display

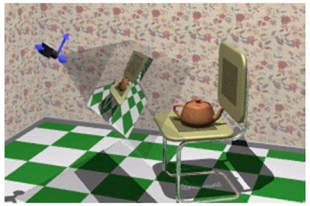
- Vertices are lit (shaded) according to material properties, surface properties and light sources
- Uses a local lighting model



The Graphics Pipeline

- Modelling Transformations
- Illumination (Shading)
- **Viewing Transformation (Perspective / Orthographic)**
- Clipping
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

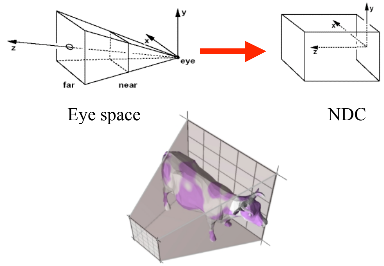
- Maps world space to eye (camera) space
- Viewing position is transformed to origin and viewing direction is oriented along some axis (typically z)



The Graphics Pipeline

- Modelling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- **Clipping**
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- Visibility / Display

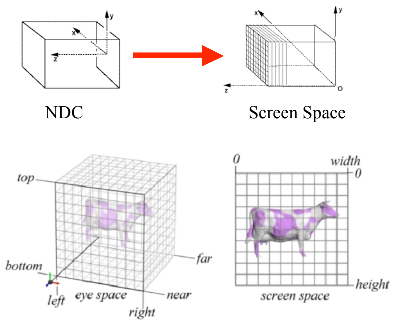
- Transforms to Normalized Device Coordinates
- Portions of the scene outside the viewing volume (view frustum) are removed (clipped)



The Graphics Pipeline

- Modelling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- **Projection (to Screen Space)**
- Scan Conversion (Rasterization)
- Visibility / Display

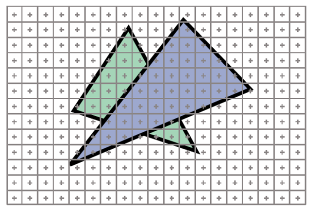
- The objects are projected to the 2D imaging plane (screen space)



The Graphics Pipeline

- Modelling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Projection (to Screen Space)
- **Scan Conversion (Rasterization)**
- Visibility / Display

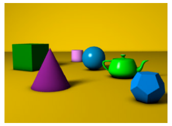
- Rasterizes objects into pixels
- Interpolate values inside objects (color, depth, etc.)

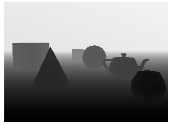


The Graphics Pipeline

- Modelling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- Projection (to Screen Space)
- Scan Conversion (Rasterization)
- **Visibility / Display**

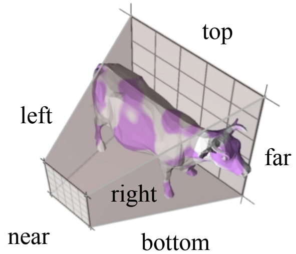
- Handles occlusions
- Determines which objects are closest and therefore visible





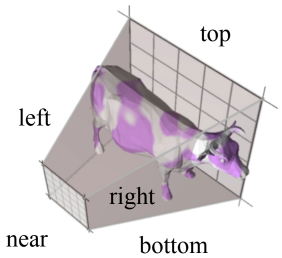
Clipping

- Eliminate portions of objects outside the viewing frustum
- View frustum
  - boundaries of the image plane projected in 3D
  - a near & far clipping plane
- User may define additional clipping planes



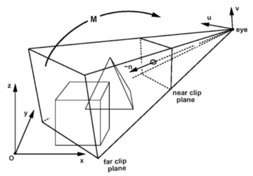
Why clipping ?

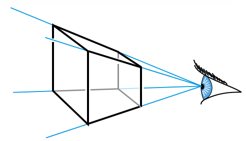
- Avoid degeneracy
  - e.g. don't draw objects behind the camera
- Improve efficiency
  - e.g. do not process objects which are not visisble

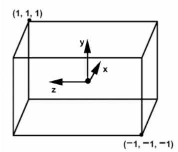


When to clip?

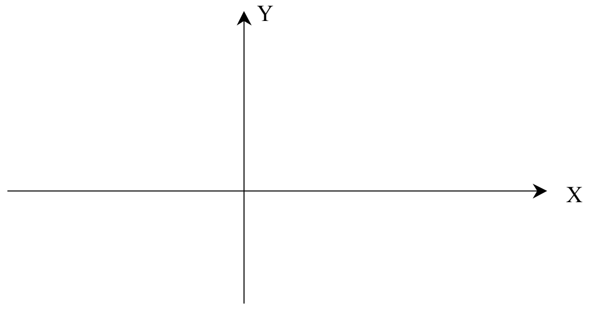
- Before perspective transform in 3D space
  - use the equation of 6 planes
  - natural, not too degenerate
- In homogeneous coordinates after perspective transform (clip space)
  - before perspective divide (4D space, weird *w* values)
  - canonical, independent of camera
  - simplest to implement
- In the transformed 3D screen space after perspective division
  - problem: objects in the plane of the camera



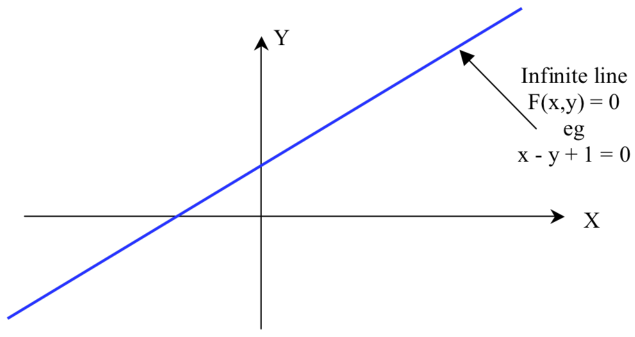




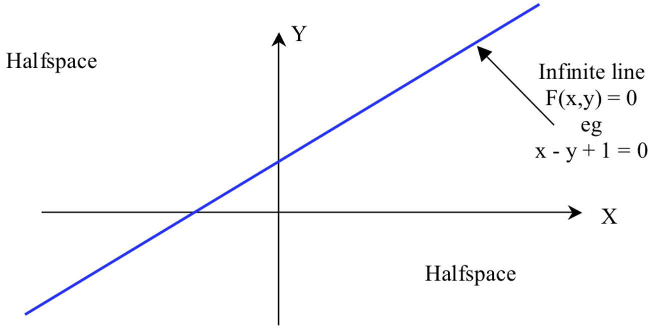
The concept of a halfspace



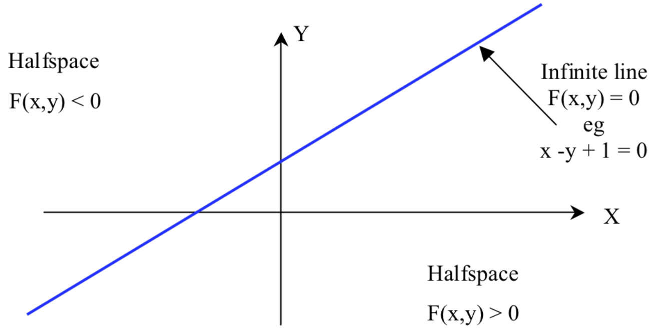
The concept of a halfspace



The concept of a halfspace



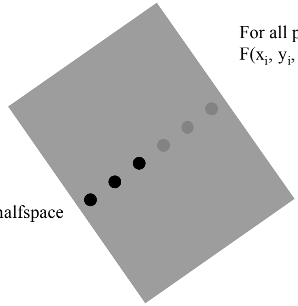
The concept of a halfspace



The concept of a halfspace in 3D

Plane equation F(x, y, z) = 0
or Ax + By + Cz + D = 0

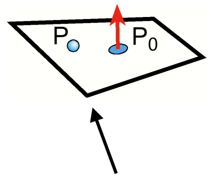
For all points in this halfspace
$F(x_i, y_i, z_i) > 0$



For all points in this halfspace
$F(x_i, y_i, z_i) < 0$

Reminder: Homogeneous Coordinates

- Recall:
  - For each point (x,y,z,w) there are an infinite number of equivalent homogenous coordinates: (sx, sy, sz, sw)

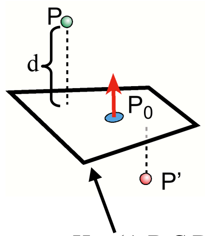


H = (A,B,C,D)

- Infinite number of equivalent plane expressions:
  sAx+sBy+sCz+sD = 0 → H = (sA,sB,sC,sD)

Point-to-Plane Distance

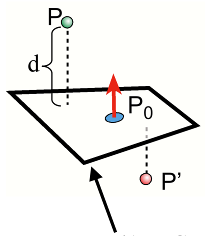
- If (A,B,C) is normalized:
  $d = H \bullet p = H^T p$
  (the dot product in homogeneous coordinates)
- d is a *signed distance:*
  positive = "inside"
  negative = "outside"



H = (A,B,C,D)

Clipping a Point with respect to a Plane

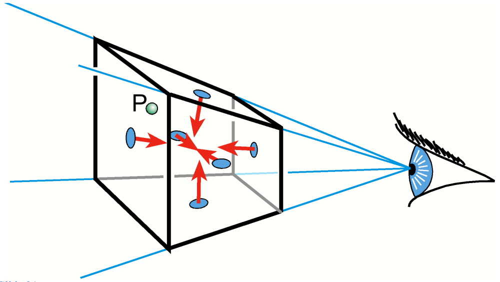
- If $d = H \bullet p \geq 0$
  Pass through
- If $d = H \bullet p < 0$:
  Clip (or cull or reject)



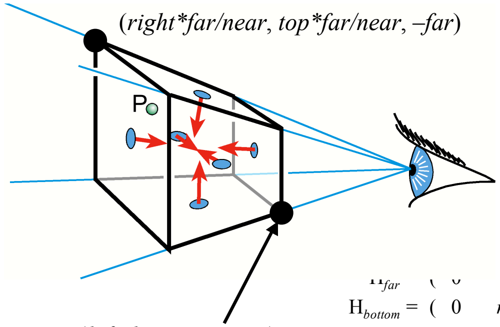
H = (A,B,C,D)

Clipping with respect to View Frustum

- Test against each of the 6 planes
  - Normals oriented towards the interior
- Clip (or cull or reject) point p if any $H \bullet p < 0$



What are the View Frustum Planes?



0 –1 –near)

H_{far} = (0 0 1 *far*)

H_{bottom} = (0 *near* *bottom* 0)

H_{top} = (0 –*near* –*top* 0)

H_{left} = (*left* *near* 0 0)

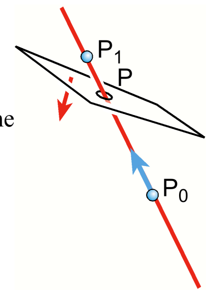
H_{right} = (–*right* –*near* 0 0)

Line – Plane Intersection

- Explicit (Parametric) Line Equation

  $L(t) = \mathbf{P_0} + \mu\,(\mathbf{P_1} - \mathbf{P_0})$
- How do we intersect?

  Insert explicit equation of line into implicit equation of plane or use the normal vector

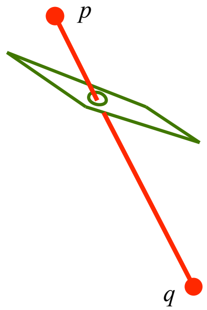


Line – Plane Intersection

- Compute the intersection between the line and plane for any vector **p** lying on the plane $\mathbf{n}\bullet\mathbf{p} = 0$
- Let the intersection point be $\mu\mathbf{p_1} + (1\text{-}\mu)\mathbf{p_0}$ and assume that **v** is a vertex of the object, a vector on the plane is given by $\mu\mathbf{p_1} + (1\text{-}\mu)\mathbf{p_0} - \mathbf{v}$
- Thus $\mathbf{n}\bullet(\mu\mathbf{p_1} + (1\text{-}\mu)\mathbf{p_0} - \mathbf{v}) = 0$ and we can solve this for μ_i and hence find the point of intersection
- We then replace $\mathbf{p_0}$ with the intersection point

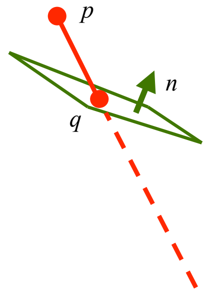
Segment Clipping

- If H•p > 0 and H•q < 0
- If H•p < 0 and H•q > 0
- If H•p > 0 and H•q > 0
- If H•p < 0 and H•q < 0



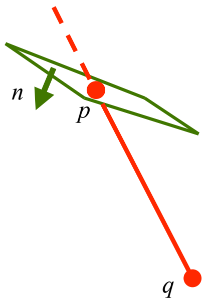
Segment Clipping

- If H•p > 0 and H•q < 0
  - clip q to plane
- If H•p < 0 and H•q > 0
- If H•p > 0 and H•q > 0
- If H•p < 0 and H•q < 0



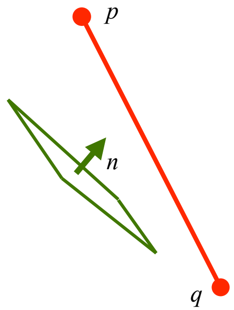
Segment Clipping

- If H•p > 0 and H•q < 0
  - clip q to plane
- If H•p < 0 and H•q > 0
  - clip p to plane
- If H•p > 0 and H•q > 0
- If H•p < 0 and H•q < 0



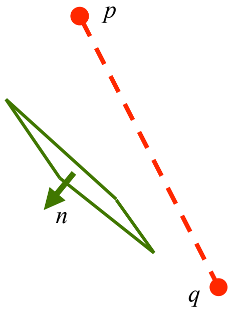
Segment Clipping

- If H•p > 0 and H•q < 0
  - clip q to plane
- If H•p < 0 and H•q > 0
  - clip p to plane
- If H•p > 0 and H•q > 0
  - pass through
- If H•p < 0 and H•q < 0



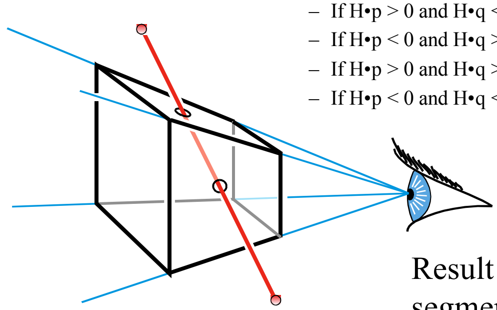
Segment Clipping

- If H•p > 0 and H•q < 0
  - clip q to plane
- If H•p < 0 and H•q > 0
  - clip p to plane
- If H•p > 0 and H•q > 0
  - pass through
- If H•p < 0 and H•q < 0
  - clipped out



Clipping against the frustum

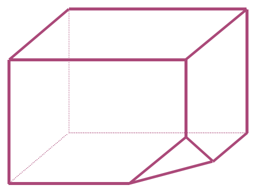
- For each frustum plane H
  - If H•p > 0 and H•q < 0, clip q to H
  - If H•p < 0 and H•q > 0, clip p to H
  - If H•p > 0 and H•q > 0, pass through
  - If H•p < 0 and H•q < 0, clipped out



Result is a single segment. Why?

Two Definitions of Convex

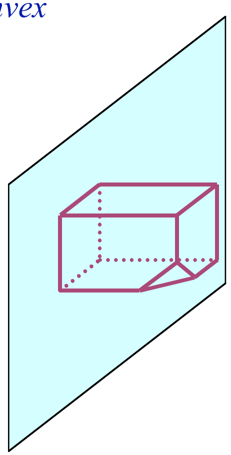
1. A line joining any two points on the boundary lies inside the object.
2. The object is the intersection of planar halfspaces.



Algorithm for determining if an object is convex

convex = *true*
for each face of the object
{ find the plane equation of the face F(x,y,z) = 0
  choose one object point (x_i,y_i,z_i) not on the face
    and find *sign*$(F(x_i,y_i,z_i))$
  for all other points of the object
  { *if* (*sign*$(F(x_j,y_j,z_j))$! = *sign*$(F(x_i,y_i,z_i))$)
    then convex = *false*
  }
}

Testing for Convex



Testing for Containment

- A frequently encountered problem is to determine whether a point is inside an object or not.
- We need this for clipping against polyhedra

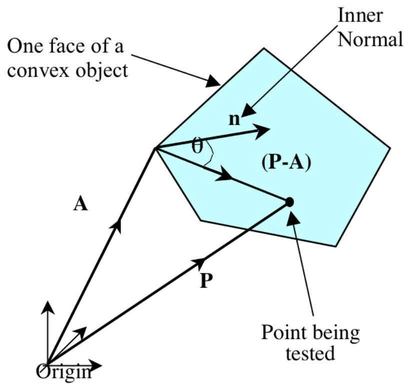
Algorithm for Containment

let the test point be (x_t,y_t,z_t)
contained = *true*
for each face of the object
{ find the plane equation of the face $F(x,y,z) = 0$
choose one object point (x_i,y_i,z_i) not on the face
and find *sign*$(F(x_i,y_i,z_i))$
if (*sign*$(F(x_t,y_t,z_t))$ not = *sign*$(F(x_i,y_i,z_i))$)
then contained = *false*
}

Vector formulation

- The same test can be expressed in vector form.
- This avoids the need to calculate the Cartesian equation of the plane, if, in our model we store the normal **n** vector to each face of our object.

Vector test for containment



Contained if θ is acute
ie Cos(θ) is positive
or **n**• (**P**-**A**) is positive
$\mathbf{n} \bullet (\mathbf{P}-\mathbf{A}) = |\mathbf{n}||\mathbf{P}-\mathbf{A}|\text{Cos}(\theta)$

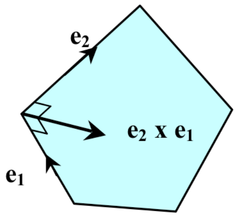
Normal vector to a face

- The vector formulation does not require us to find the plane equation of a face, but it does require us to find a normal vector to the plane; same thing really since for plane Ax + By +Cz + D = 0 a normal vector is

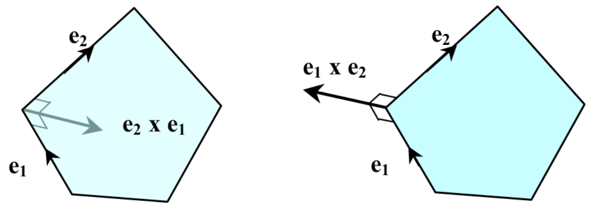
  n = (A, B, C)

Finding a normal vector

- The normal vector can be found from the cross product of two vectors on the plane, say two edge vectors



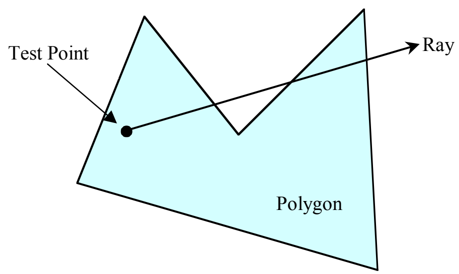
But which normal vector points inwards?



Concave Objects

- Containment and clipping can also be carried out with concave objects.
- Most algorithms are based on the ray containment test.

The Ray test in two dimensions



Find all intersections between the ray and the polygon edges.
If the number of intersections is odd the point is contained

Calculating intersections with rays

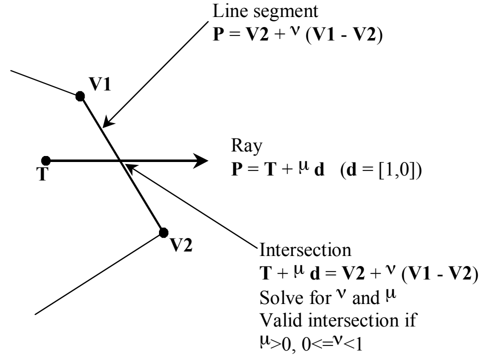
- Rays have equivalent equations to lines, but go in only one direction. For test point T a ray is defined as

$$\mathbf{R} = \mathbf{T} + \mu\, \mathbf{d} \quad \mu>0$$

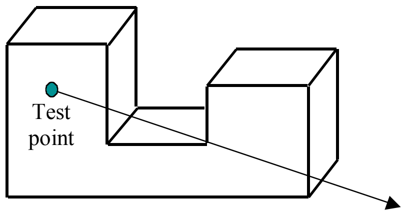
- We choose a simple to compute direction eg

$$\mathbf{d} = [1,0,0]$$

Valid Intersections



Extending the ray test to 3D



A ray is projected in any direction.

If the number of intersections with the object is odd, then the test point is inside

3D Ray test

- There are two stages:
  1. Compute the intersection of the ray with the plane of each face.
  2. If the intersection is in the positive part of the ray ($\mu>0$) check whether the intersection point is contained in the face.

The plane of a face

- Unfortunately the plane of a face does not in general line up with the Cartesian axes, so the second part is not a two dimensional problem.

- However, containment is invariant under orthographic projection, so it can be simply reduced to two dimensions.

Clipping to concave volumes

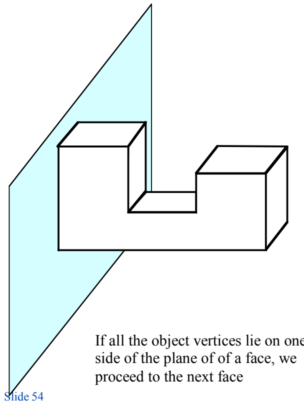
- Find every intersection of the line to be clipped with the volume.

- This divides the line into one or more segments.

- Test a point on the first segment for containment

- Adjacent segments will be alternately inside and out.

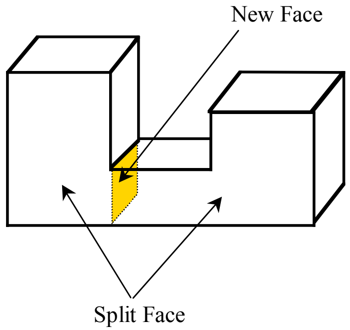
Splitting a volume into convex parts



If all the object vertices lie on one side of the plane of of a face, we proceed to the next face

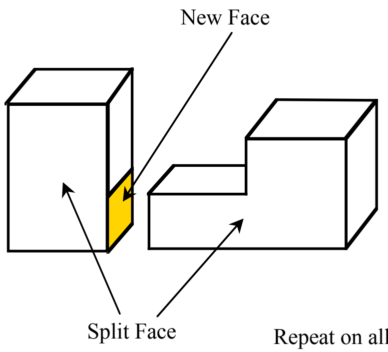
Graphics Lecture 4: Slide 54

If the plane of a face cuts the object:



Graphics Lecture 4: Slide 55

Split the Object



Repeat on all concave sub parts

Graphics Lecture 4: Slide 56