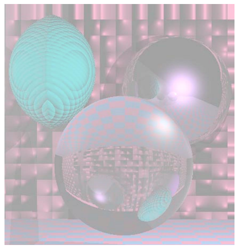
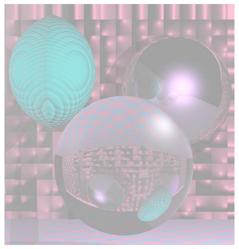


CS 4300 Computer Graphics

Prof. Harriet Fell
Fall 2011
Lecture 10 – September 28, 2011



3D Vectors

What we said about 2D vectors holds for 3D vectors too.

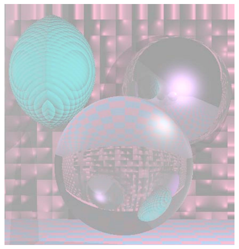
$\mathbf{a} = (x_a, y_a, z_a)$ $\mathbf{b} = (x_b, y_b, z_b)$

$$Length(\mathbf{a}) = Norm(\mathbf{a}) = \|\mathbf{a}\| = \sqrt{x_a^2 + y_a^2 + z_a^2}$$

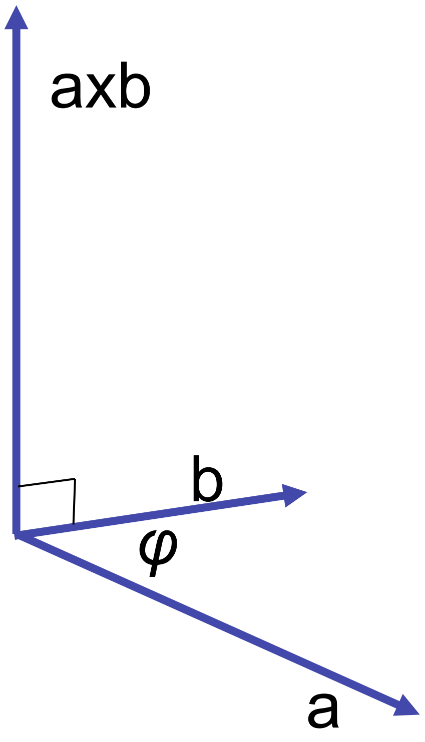
$$a \bullet b = x_a x_b + y_a y_b + z_a z_b$$

$$a \bullet b = ||a|| \bullet ||b|| \cos(\varphi)$$

$$\mathbf{a} \rightarrow \mathbf{b} = \|\mathbf{a}\| \cos(\varphi) = \frac{\mathbf{a} \bullet \mathbf{b}}{\|\mathbf{b}\|}$$



Vector Cross Product



$$\|\mathbf{a}\times\mathbf{b}\| = \|\mathbf{a}\|\,\|\mathbf{b}\|\sin\varphi$$

axb is perpendicular to a and b.

Use the right hand rule to determine the direction of axb.

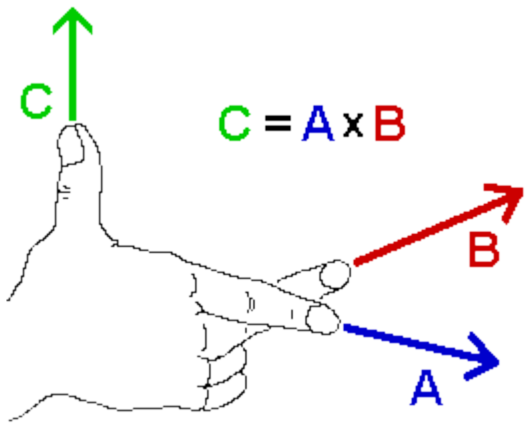
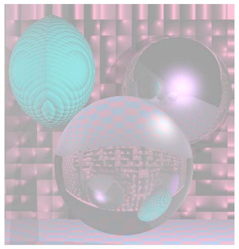
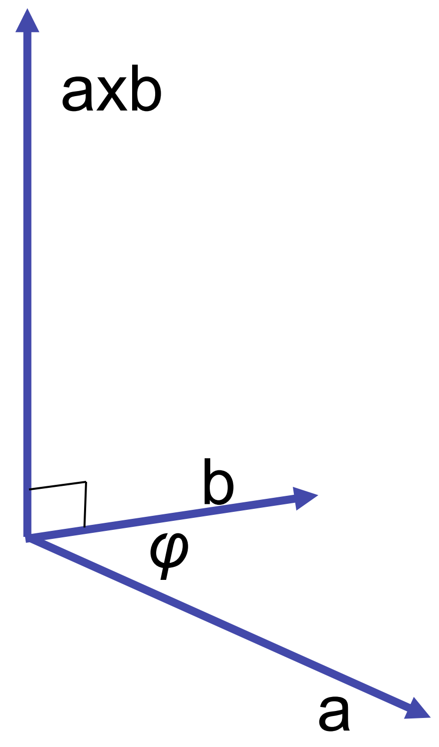


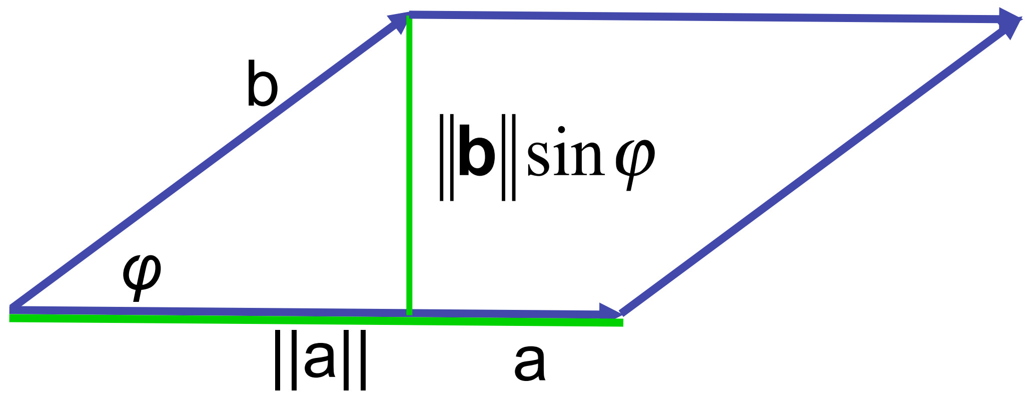
Image from www.physics.udel.edu



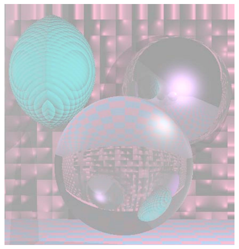
Cross Product and Area

$$\|\mathbf{a}\times\mathbf{b}\| = \|\mathbf{a}\|\,\|\mathbf{b}\|\sin\varphi$$





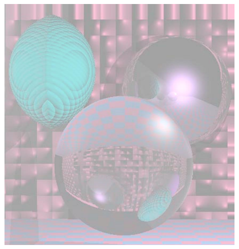
||a x b|| = area of the parallelogram.



Computing the Cross Product

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} i & j & k \\ \mathrm{a}_x & \mathrm{a}_y & \mathrm{a}_z \\ \mathrm{b}_x & \mathrm{b}_y & \mathrm{b}_z \end{vmatrix}$$

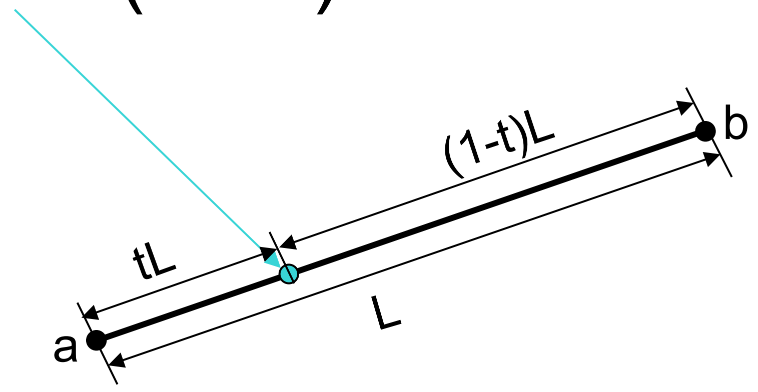
$$= \left(\mathrm{a}_y \mathrm{b}_z - \mathrm{a}_z \mathrm{b}_y\right) i + \left(\mathrm{a}_z \mathrm{b}_x - \mathrm{a}_x \mathrm{b}_z\right) j + \left(\mathrm{a}_x \mathrm{b}_y - \mathrm{a}_y \mathrm{b}_x\right) k$$

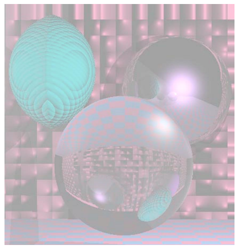


Linear Interpolation

- **LERP**: /lerp/, vi.,n.
  - Quasi-acronym for Linear Interpolation, used as a verb or noun for the operation. “Bresenham's algorithm lerps incrementally between the two endpoints of the line.”

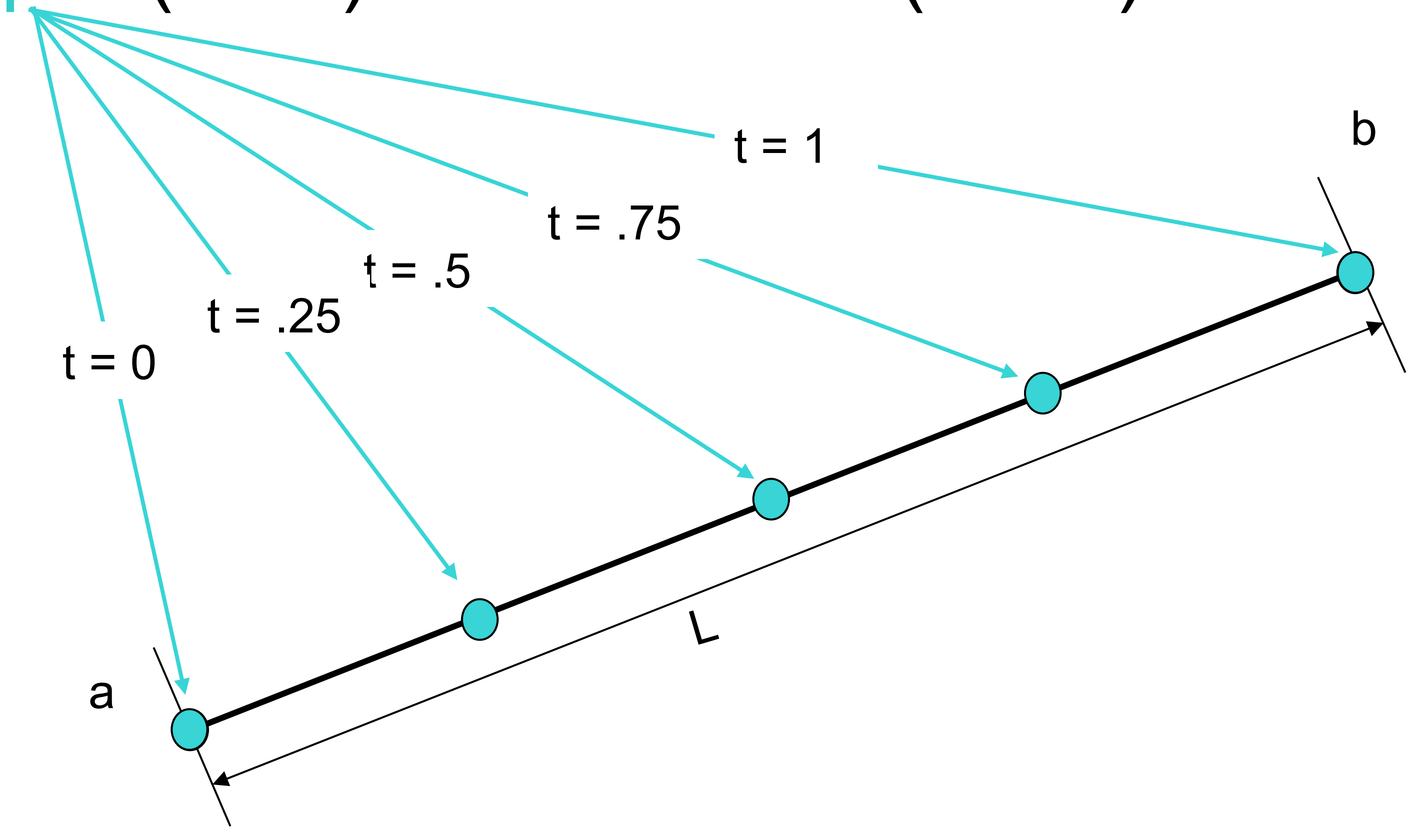
$$p = (1 - t)\, a + t\, b = a + t(b - a)$$

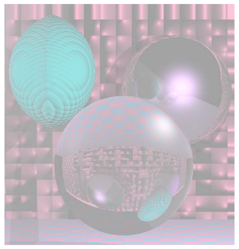




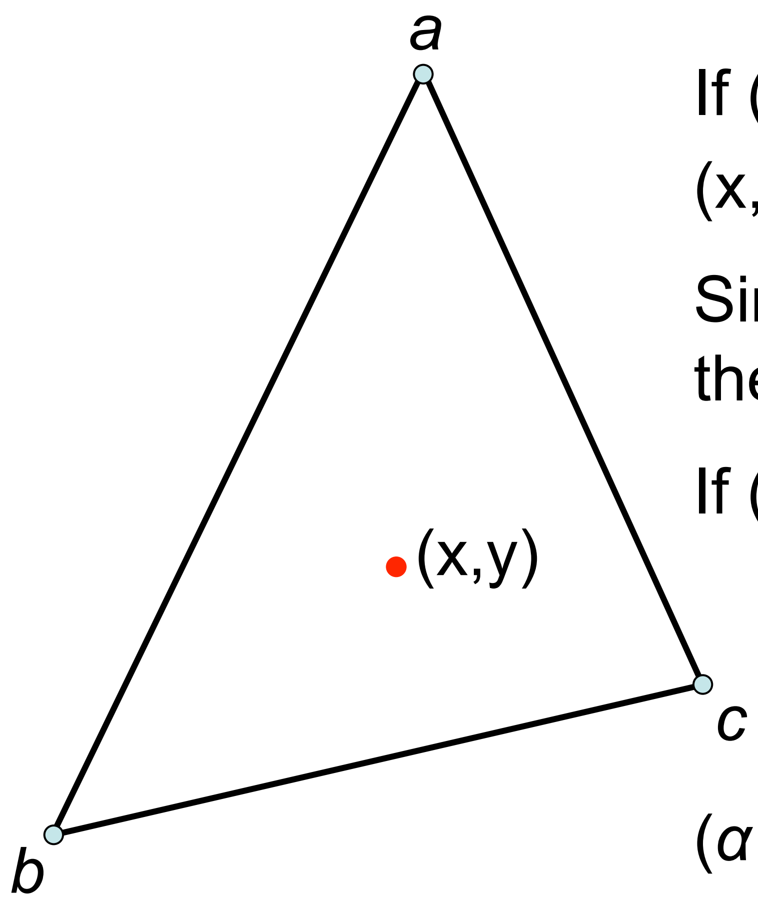
Lerping

$$p = (1 - t)\, a + t\, b = a + t(b - a)$$





Triangles



If (x, y) is on the edge ab,

(x, y) = (1 – t) a + t b = a + t(b – a).

Similar formulas hold for points on the other edges.

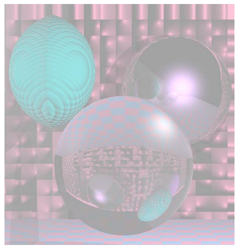
If (x, y) is in the triangle:

$$(x, y) = \alpha\, a + \beta\, b + \gamma\, c$$

$$\alpha + \beta + \gamma = 1$$

(α , β , γ) are the

Barycentric coordinates of (x, y).



Triangles

$p = a + \beta(b-a) + \gamma(c-a)$

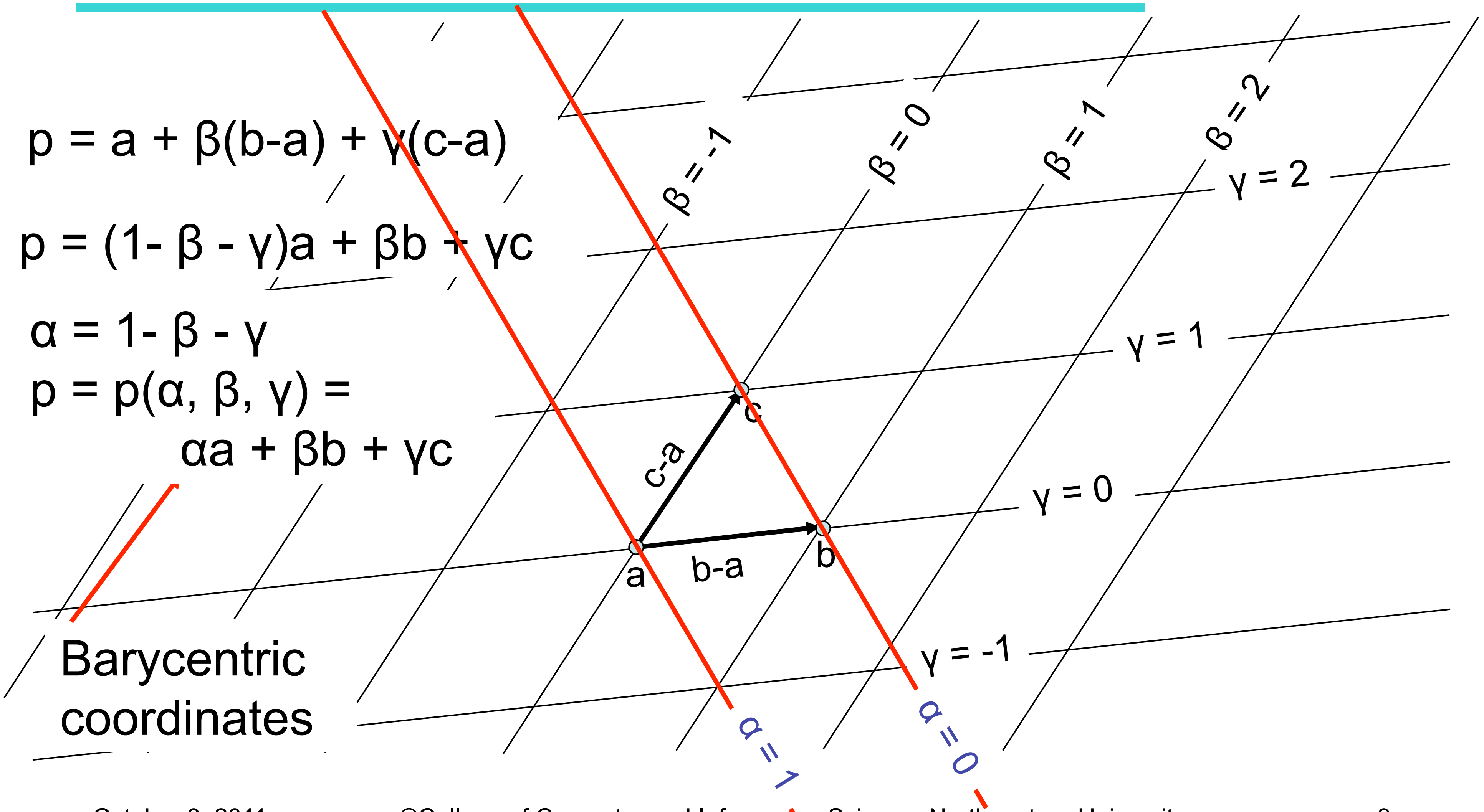
$p = (1- \beta - \gamma)a + \beta b + \gamma c$

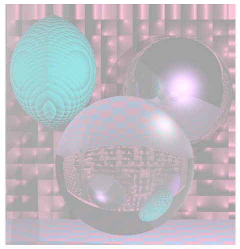
$\alpha = 1- \beta - \gamma$

$p = p(\alpha, \beta, \gamma) =$

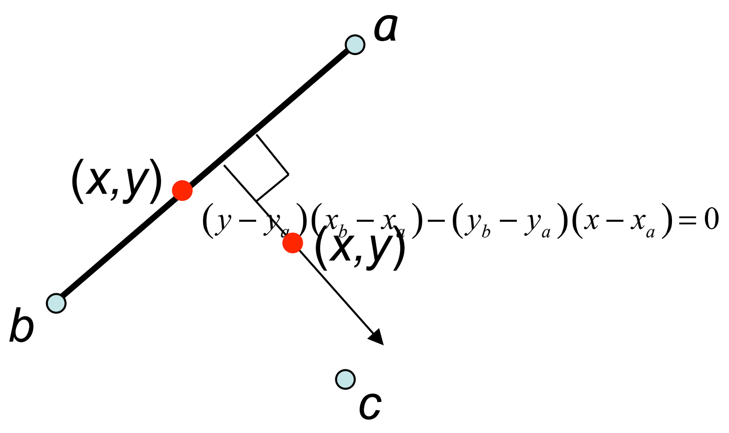
$\alpha a + \beta b + \gamma c$

Barycentric coordinates





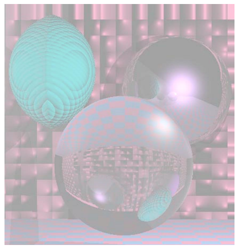
Computing Barycentric Coordinates



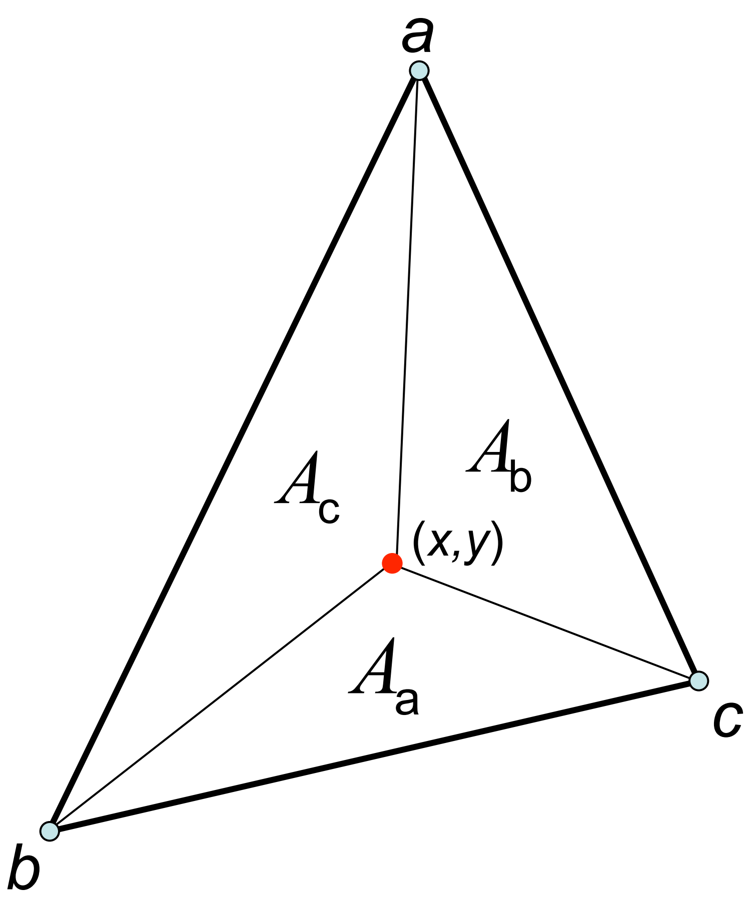
$$\frac{y-y_a}{x-x_a}=\frac{y_b-y_a}{x_b-x_a}$$

$$f_{ab}(x,y)=(y-y_a)(x_b-x_a)-(y_b-y_a)(x-x_a)$$

$$\gamma=\frac{f_{ab}(x,y)}{f_{ab}(x_c,y_c)}$$



Barycentric Coordinates as Areas



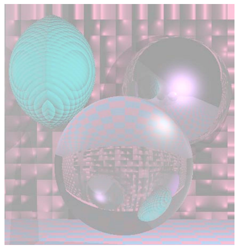
$$\alpha = A_a / A$$

$$\beta = A_b / A$$

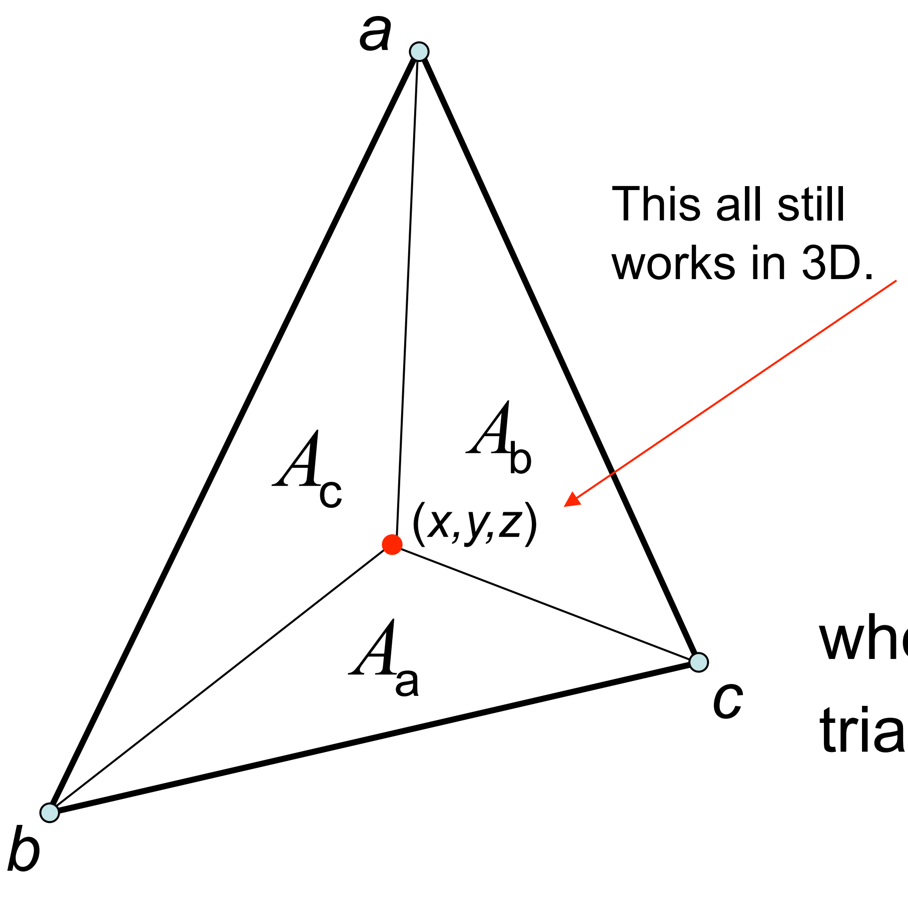
$$\gamma = A_c / A$$

where *A* is the area of the triangle.

$$\alpha + \beta + \gamma = 1$$



3D Triangles



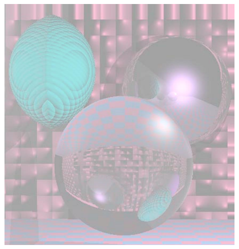
$$\alpha = A_a / A$$

$$\beta = A_b / A$$

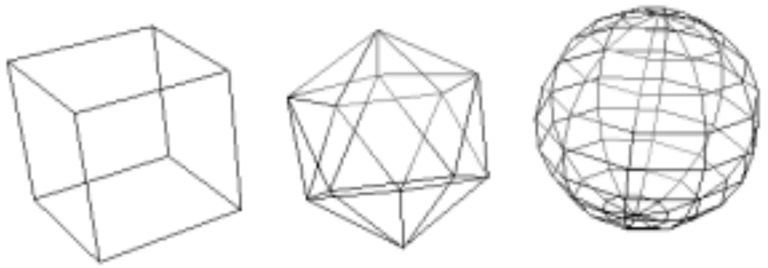
$$\gamma = A_c / A$$

where A is the area of the triangle.

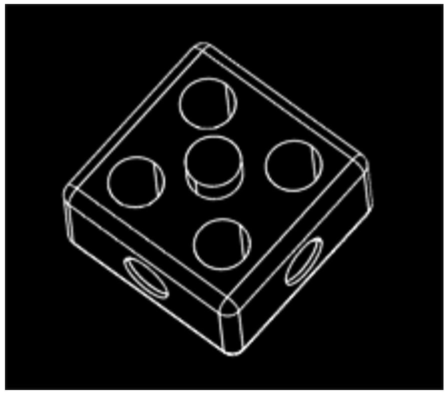
$$\alpha + \beta + \gamma = 1$$



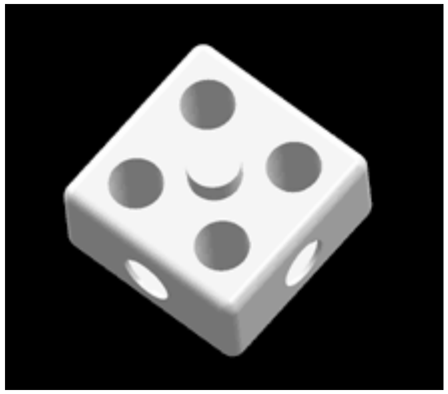
Wireframe Rendering

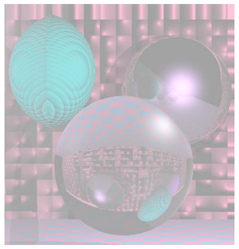


Hidden-Line Removal

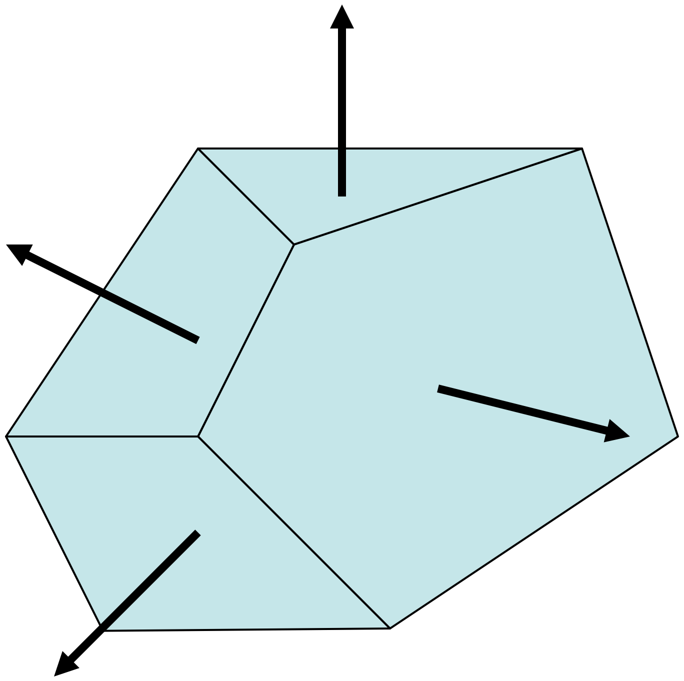


Hidden-Face Removal





Convex Polyhedra

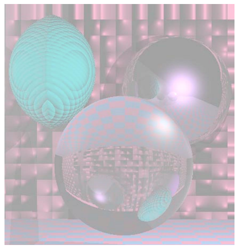


We can see a face if and only if its normal has a component toward us.

$$\mathbf{N}\cdot\mathbf{V} > 0$$

V points from the face toward the viewer.

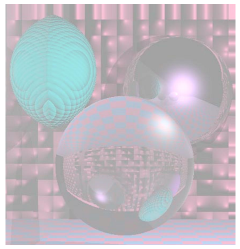
N point toward the outside of the polyhedra.



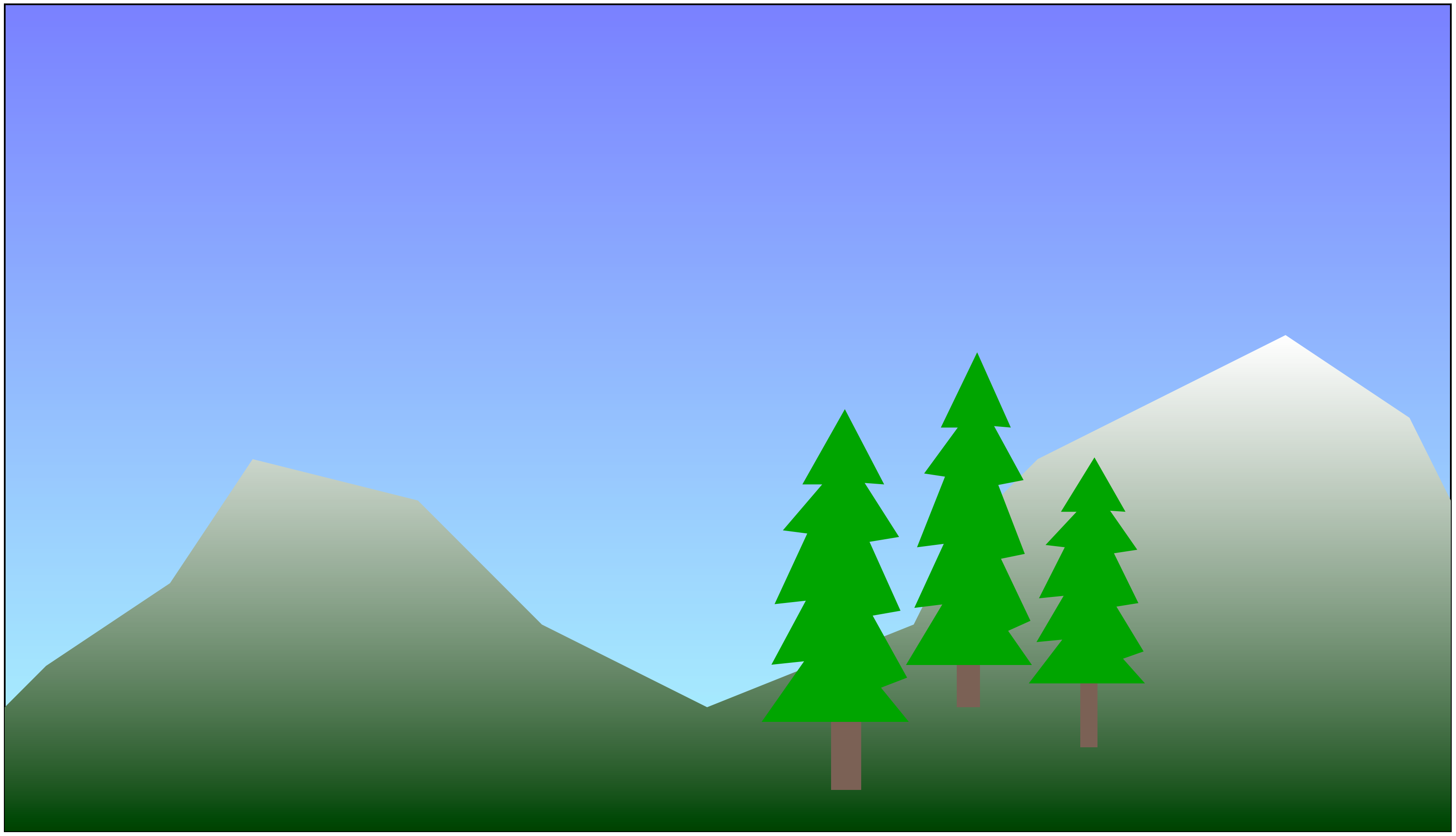
Hidden Surface Removal

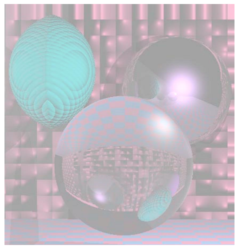
- Backface culling
  - Never show the back of a polygon.
- Viewing frustum culling
  - Discard objects outside the camera’s view.
- Occlusion culling
  - Determining when portions of objects are hidden.
    - Painter’s Algorithm
    - Z-Buffer
- Contribution culling
  - Discard objects that are too far away to be seen.

http://en.wikipedia.org/wiki/Hidden_face_removal



Painter’s Algorithm

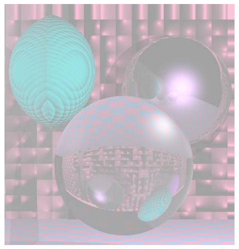




Painter’s Algorithm

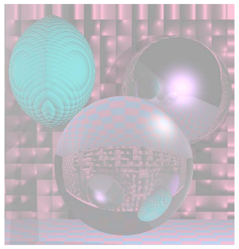
```
Sort objects back to front relative to the
    viewpoint.
for each object (in the above order) do
    draw it on the screen
```

-- more on this later



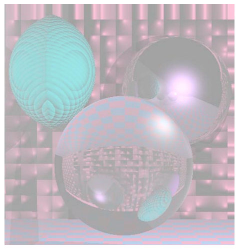
Visible Surface Determination

- If surfaces are invisible, don’t render them.
  - Ray Tracing
    - We only render the nearest object.
  - Binary Space Partitioning (BSP)
    - Recursively cut up space into convex sets with hyperplanes.
    - The scene is represented by a BSP-tree.



Rendering a Polymesh

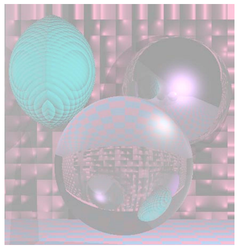
- Scene is composed of triangles or other polygons.
- We want to view the scene from different view-points.
  - Hidden Surface Removal
    - Cull out surfaces or parts of surfaces that are not visible.
  - Visible Surface Determination
    - Head right for the surfaces that are visible.
    - Ray-Tracing is one way to do this.



Polygon **T**able

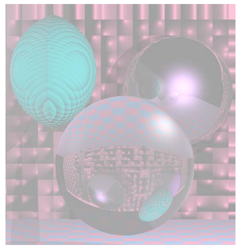
Polygon **T**able

- A, B, C, D of the plane equation
- shading or color info (e.g. color and N)
- *in* (*out*) boolean
  - initialized to false (= *out*) at start of scanline
- z – at lowest y, x



Coherence

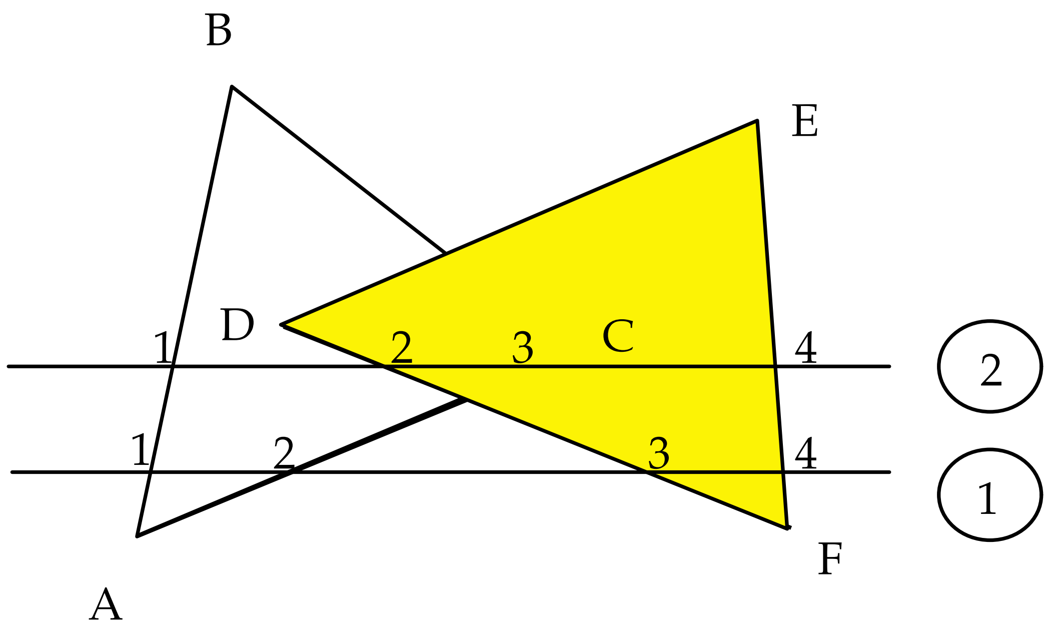
- Non-penetrating polygons maintain their relative z values.
  - If the polygons penetrate, add a false edge.
- If there is no change in edges from one scanline to the next, and no change in order wrt x, then no new computations of z are needed.

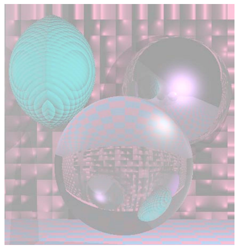


Active Edge Table

Keep in order of increasing x.

At (1) AET ➔ AB ➔ AC ➔ DF ➔ EF

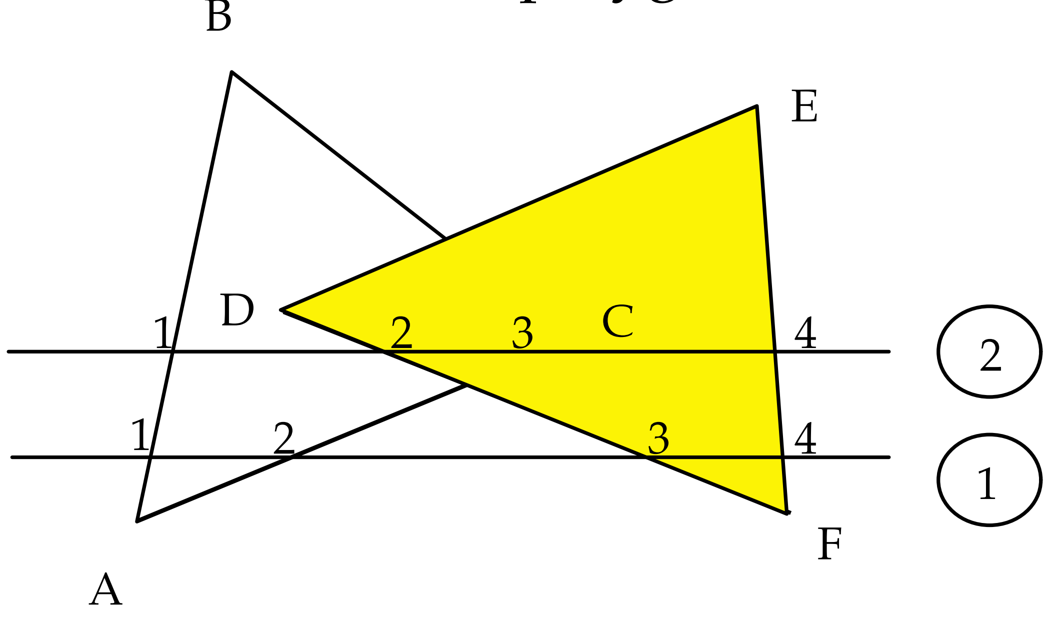


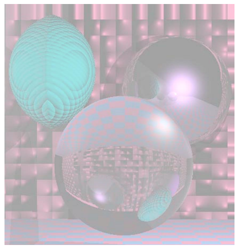


Running the Algorithm 1

If more than one in is true, compute the z values at that point to see which polygon is furthest forward.

If only one in is true, use that polygon's color and shading.





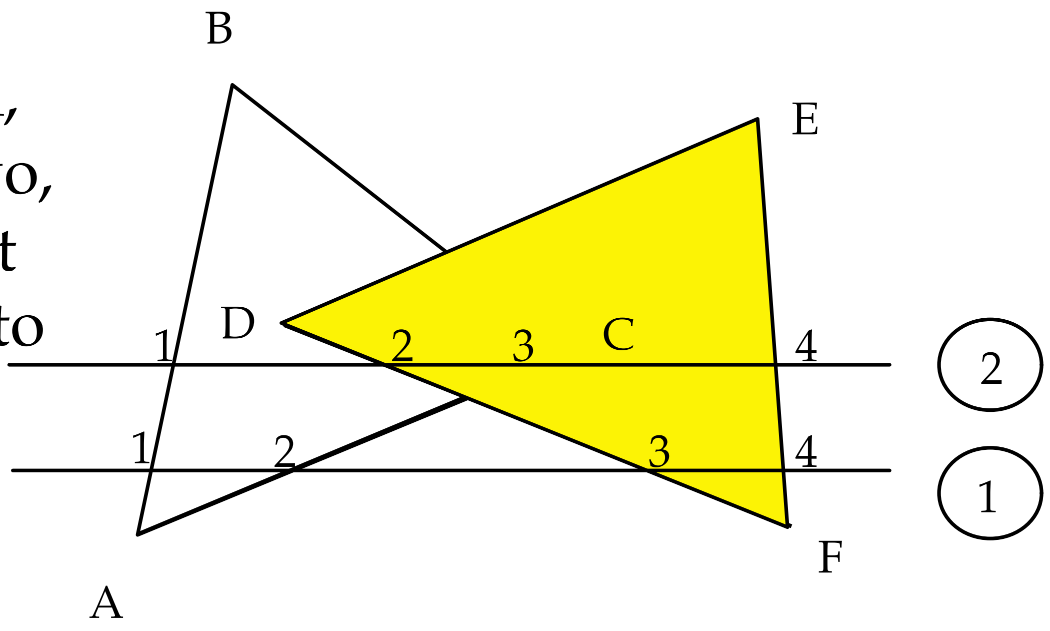
Running the Algorithm 2

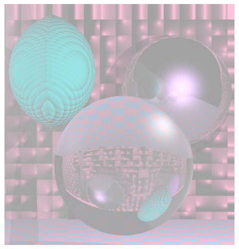
On crossing an edge

set in of polygons with that edge to **not** in.

At (2) AET ➔ AB ➔ DF ➔ AC EF

If there is a third polygon, GHIJ behind the other two, after edge AC is passed at level (2) there is no need to evaluate z again - if the polygons do not pierce each other.

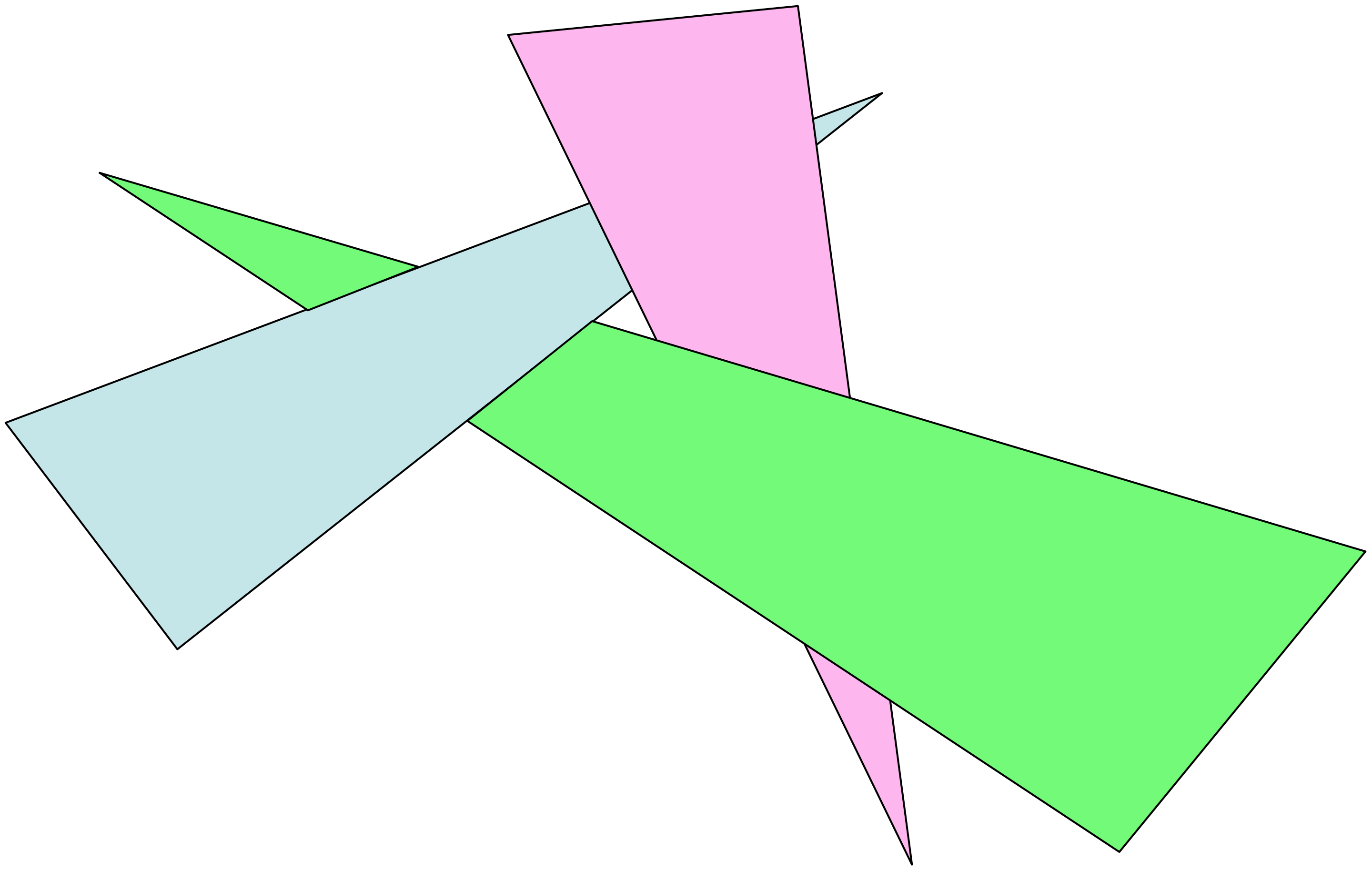


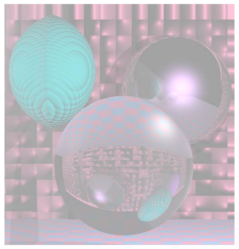


Painter's Algorithm

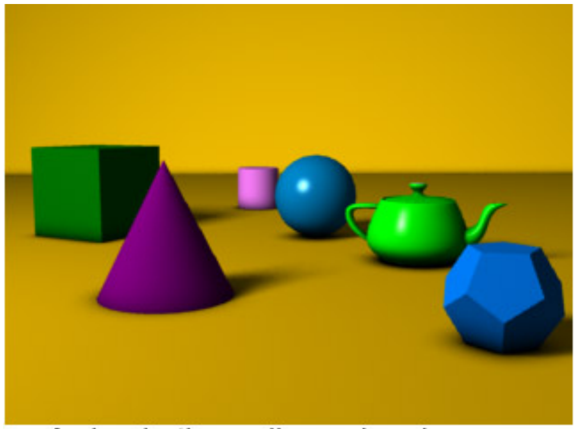
```
Sort objects back to front relative to the
    viewpoint.
**for** each object (in the above order) **do**
    draw it on the screen
```

Painter’s Problem

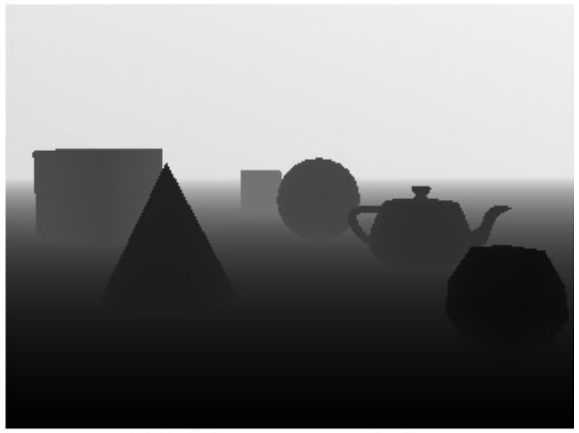




Z-Buffer



A simple three dimensional scene

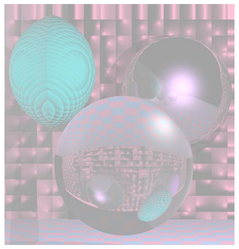


Z-buffer representation

The **Z-Buffer** is usually part of graphics card hardware. It can also be implemented in software. The **Z-Buffer** is a 2D array that holds one value for each pixel.

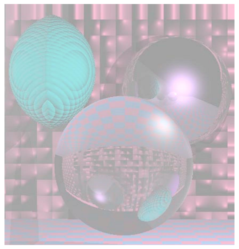
The depth of each pixel is stored in the z-buffer.

An object is rendered at a pixel only if its z-value is higher(lower) than the buffer value. The buffer is then updated.



Visible Surface Determination

- If surfaces are invisible, don’t render them.
  - Ray Tracing
    - We only render the nearest object.
  - Binary Space Partitioning (BSP)
    - Recursively cut up space into convex sets with hyperplanes.
    - The scene is represented by a BSP-tree.



Sorting the Polygons

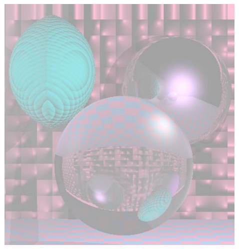
The first step of the Painter's algorithm is:

Sort objects back to front relative to the viewpoint.

The relative order may not be well defined.

We have to reorder the objects when we change the viewpoint.

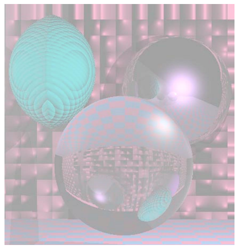
The BSP algorithm and BSP trees solve these problems.



Binary Space Partition

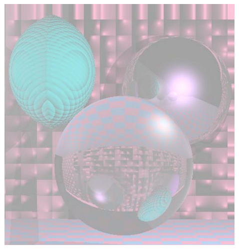
- Our scene is made of triangles.
  – Other polygons can work too.
- Assume no triangle crosses the plane of any other triangle.
  – We relax this condition later.

following Shirley *et al.*



BSP – Basics

- Let a plane in 3-space (or line in 2-space) be defined implicitly, i.e.
  - $f(\boldsymbol{P}) = f(x, y, z) = 0$ in 3-space
  - $f(\boldsymbol{P}) = f(x, y) = 0$ in 2-space
- All the points $\boldsymbol{P}$ such that $f(\boldsymbol{P}) > 0$ lie on one side of the plane (line).
- All the points $\boldsymbol{P}$ such that $f(\boldsymbol{P}) < 0$ lie on the other side of the plane (line).
- Since we have assumed that all vertices of a triangle lie on the same side of the plane (line), we can tell which side of a plane a triangle lies on.



BSP on a Simple Scene

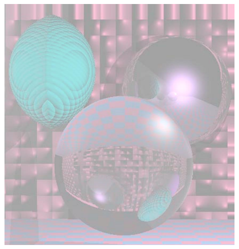
Suppose scene has 2 triangles

$T1$ on the plane $f(\boldsymbol{P}) = 0$

$T2$ on the $f(\boldsymbol{P}) < 0$ side

e is the eye.

```
if $f(e) < 0$ then
        draw $T1$; draw $T2$
else
        draw $T2$; draw $T1$
```



The BSP Tree

Suppose scene has many triangles, *T1, T2, ...* .

We still assume no triangle crosses the plane of any other triangle.

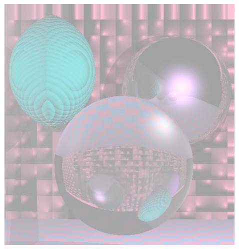
Let $f_i(\boldsymbol{P}) = 0$ be the equation of the plane containing *Ti.*

The *BSPTREE* has a node for each triangle with *T1* at the root.

At the node for *Ti,*

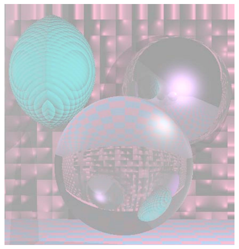
the minus subtree contains all the triangles whose vertices have $f_i(\boldsymbol{P}) < 0$

the plus subtree contains all the triangles whose vertices have $f_i(\boldsymbol{P}) > 0$.



BSP on a non-Simple Scene

```
function draw(bsptree tree, point $e$)
if (tree.empty) then
    return
if ($f_{tree.root}(e) < 0$) then
    draw(tree.plus, $e$)
    render tree.triangle
    draw(tree.minus, $e$)
else
    draw(tree.minus, $e$)
    render tree.triangle
    draw(tree.plus, $e$)
```



2D BSP Trees Demo

http://www.symbolcraft.com/graphics/bsp/index.php

This is a demo in 2 dimensions.

The objects are line segments.

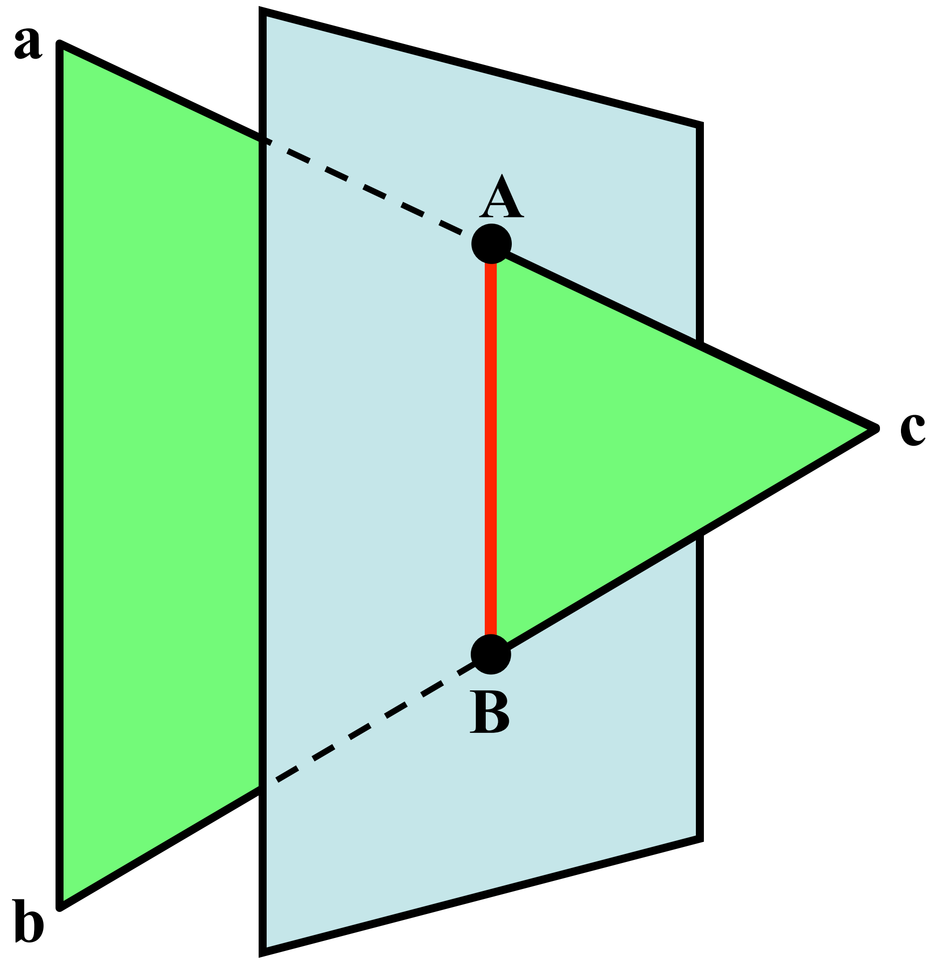
The dividing hyperplanes are lines.

Building the BSP Tree

We still assume no triangle crosses the plane of another triangle.

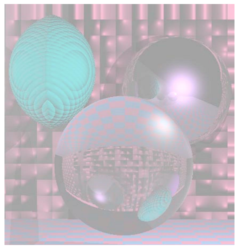
```
tree = node(T1)
for  I in {2, …, N} do tree.add(Ti)

function add (triangle T)
if (f(a) < 0 and f(b) < 0 and f(c) < 0) then
   if (tree.minus.empty) then
      tree.minus = node(T)
   else
       tree.minus.add(T)
else if (f(a) > 0 and f(b) > 0 and f(c) > 0) then
    if (tree.plus.empty) then
       tree.plus = node(T)
   else
      tree.plus.add(T)
```

Triangle Crossing a Plane

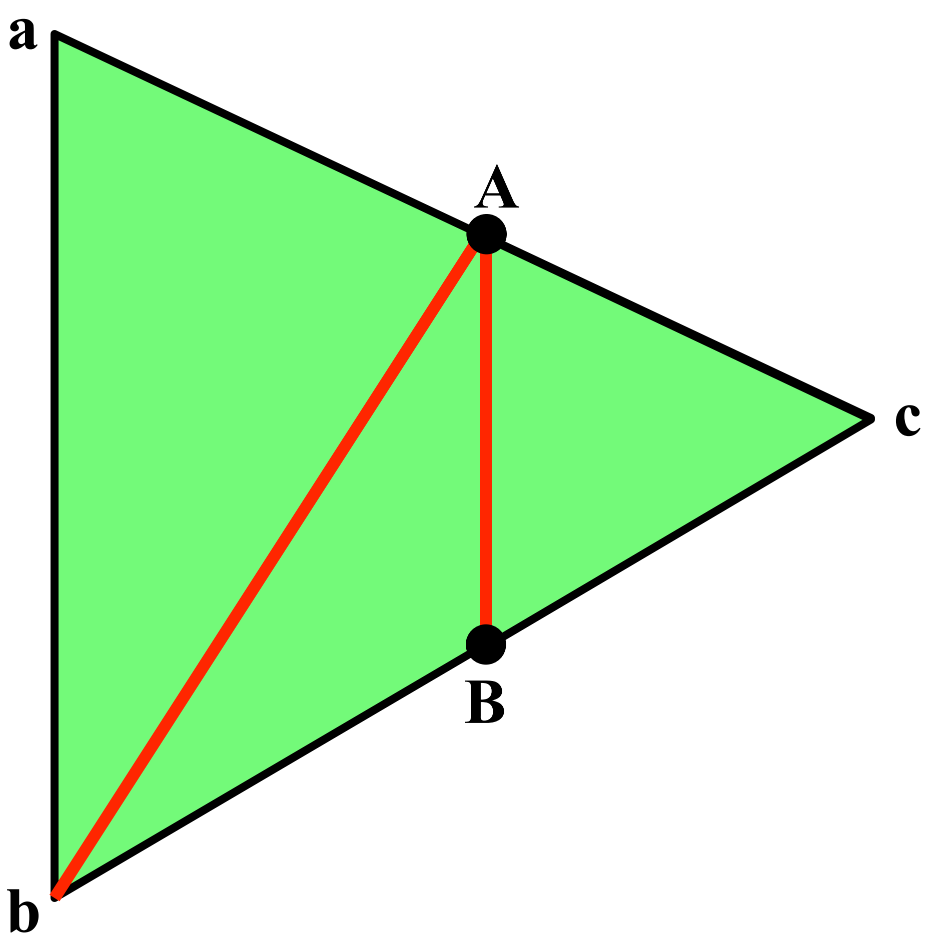


Two vertices, **a** and **b**, will be on one side and one, **c**, on the other side.

Find intercepts , **A** and **B**, of the plane with the 2 edges that cross it.



Cutting the Triangle



Cut the triangle into three triangles, none of which cross the cutting plane.

Be careful when one or more of **a**, **b**, and **c** is close to or on the cutting plane.