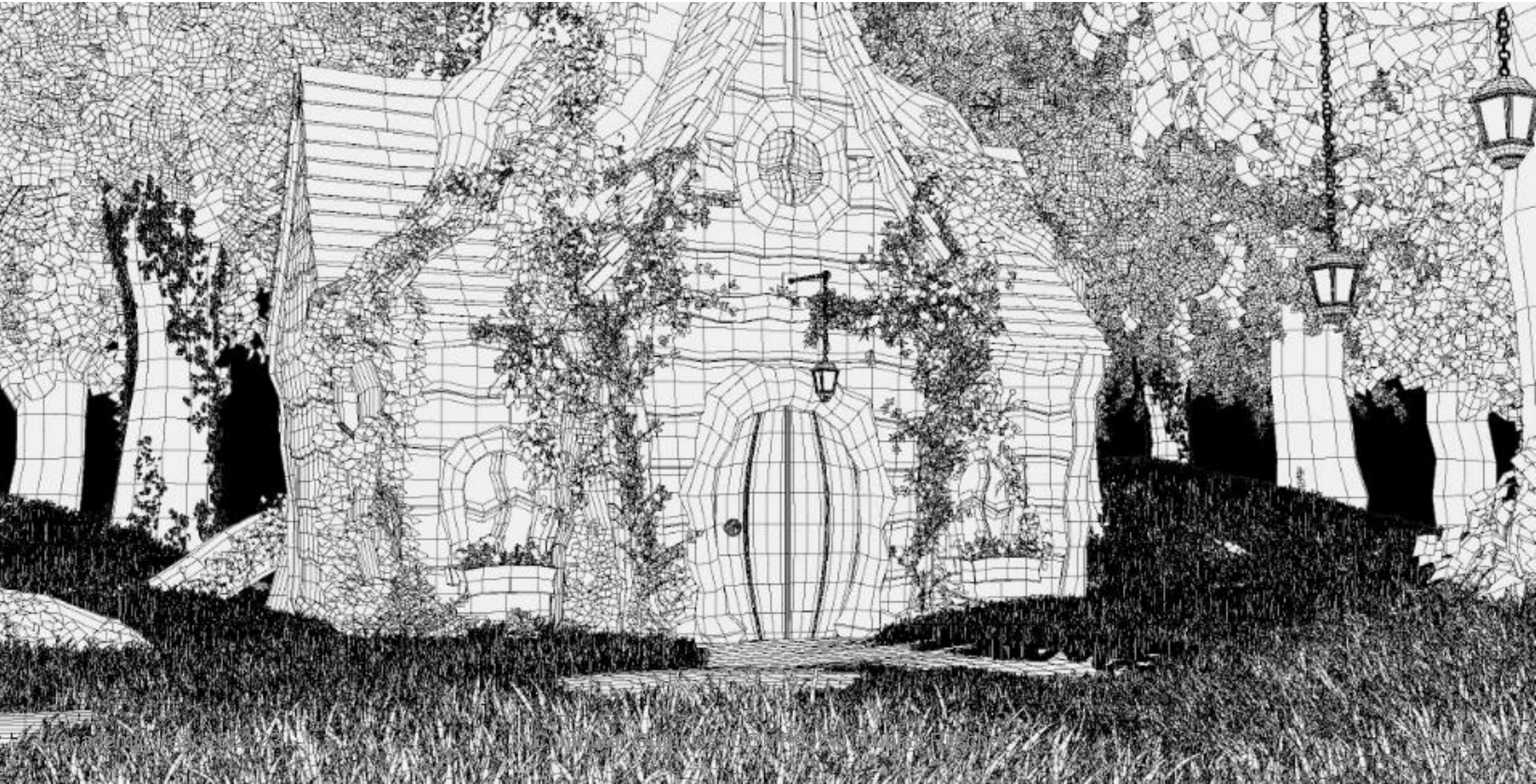
Computer Graphics (COMP0027) 2022/23

Polygon Intersection

Tobias Ritschel

Polygonal meshes



Overview

- Barycentric Coordinates
- Ray-Polygon Intersection Test

- This part:
  ray tracing **one** polygon
- Next part:
  ray-tracing objects with **many** polygons

Ray Tracing a Polygon

Three steps

1. Does the ray intersect the plane of the polygon?
   I.e., is the ray not orthogonal to the plane normal

2. Intersect ray with plane
   Also interval test: is the hit in the right interval

3. Test whether intersection point lies within polygon on the plane

Does the Ray Intersect the Plane?

- Ray equation is: $\mathbf{r}(t) = \mathbf{p}_0 + t\,\mathbf{d}$
- Plane equation is: $<\mathbf{n}, (x, y, z)> = d$

- Then test is $<\mathbf{n}, \mathbf{d}> \,!= 0$

  → ray does intersect plane (ray direction and plane are not parallel)

Where Does It Intersect?

- Substitute line equation into plane equation

$$\mathbf{n} \times \begin{pmatrix} x_0 + td_x & y_0 + td_y & z_0 + td_z \end{pmatrix} = d$$

- Solve for t

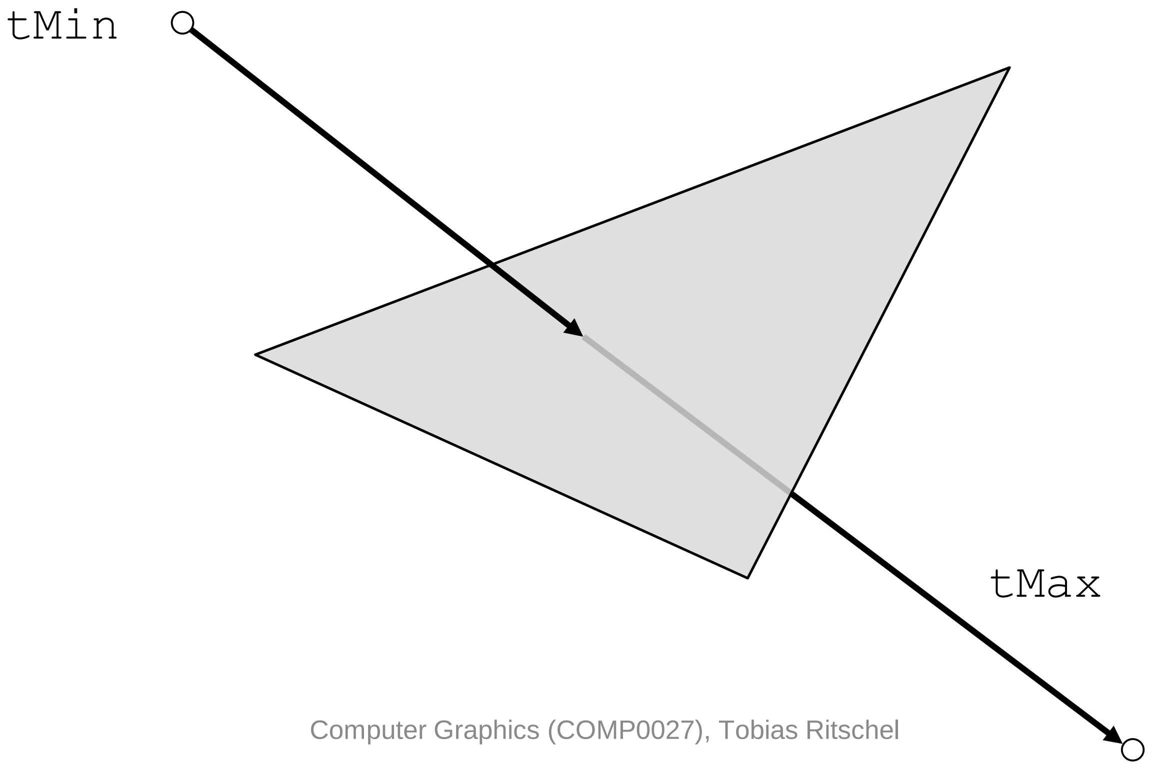
$$t = \frac{d - (\mathbf{n} \times \mathbf{p}_0)}{\mathbf{n} \times \mathbf{d}}$$

- Intersection:

$$\mathbf{p}_{\text{int}} = \mathbf{p}_0 + t\mathbf{d}$$

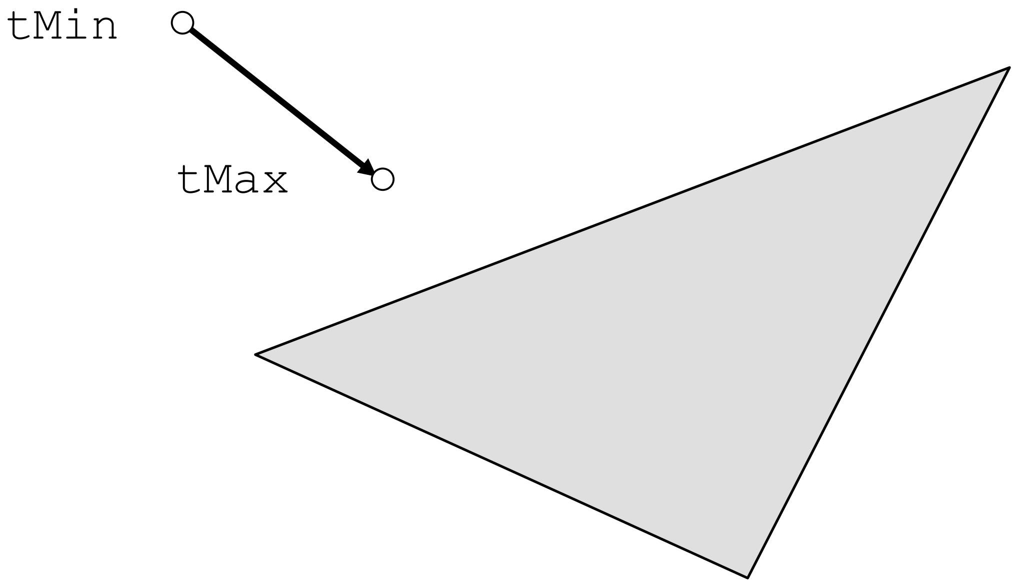
Interval test

```
intersect(tri, ray, tMin, tMax)
```



Interval test

```
intersect(tri, ray, tMin, tMax)
```

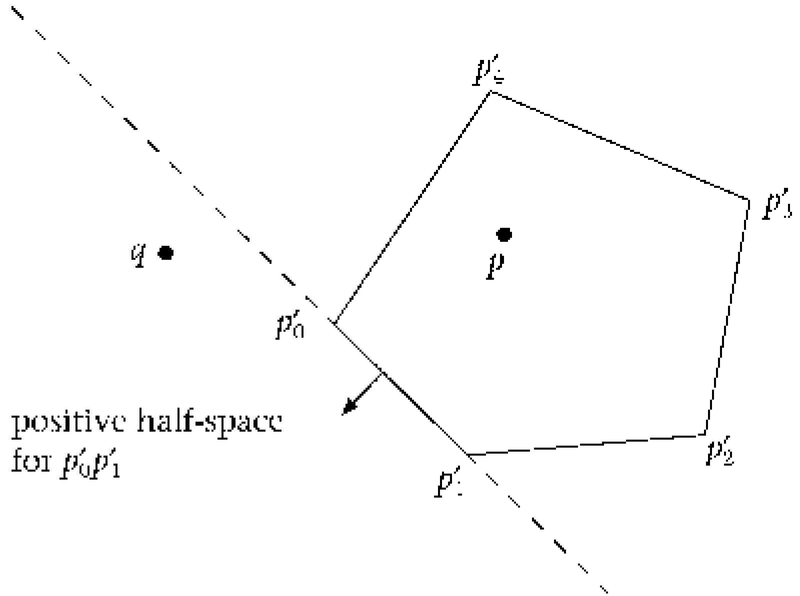


Is This Point Inside the Polygon?

- Many tests are possible
  - Winding number (can be done in 3D)
  - Infinite ray test (done in 2D)
  - Half-space test (done in 2Dish for convex poylgons)
  - Barycentric coordinates (in 3D, good for triangles)

Half-Space Test (Convex)

- A point **p** is inside a polygon if it is in the negative half-space of all the line segments

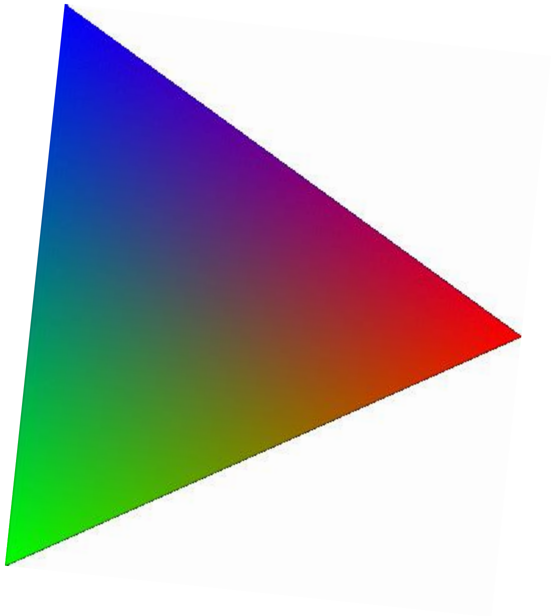


Polygon intersection, pseudocode

```
HitInfo intersectPoly(ray, poly, min, max) {
 HitInfo hitInfo = intersectPlane(ray, tri, min, max);
 if(! hitInfo.is) return emptyHitInfo();
 vec3 normal = getNormal(poly);
 for(i from 0 to numebrOfEdges) {
  vec3 edge = getEdge(poly, i, i+1);
  vec3 halfSpaceNormal = cross(normal, edge);
  float d = dot(poly.positions[i], edge);
  float d2 dot(hitInfo, edge);
  if(d < d2) return emptyHitInfo();
 }
 return hitInfo;
}
```

Method 2: Barycenters

- Need to learn something new to do this: Barycentric Coordinates



Line Equation

- Recall that given $\mathbf{p}_1$ and $\mathbf{p}_2$ in 3D space, the straight line that passes between is, for any real number t

$$\mathbf{p}(t) = (1 - t)\mathbf{p}_1 + t\mathbf{p}_2$$

- We can consider t a **weighting**
- This is a simple example of a *barycentric combination*

Barycentric Combinations

- A barycentric combination is a weighted sum of points, where the weights sum to 1.
  - Let $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ be points
  - Let $a_1, a_2, \ldots, a_n$ be weights

$$\mathbf{p} = \sum_{i=i}^{n} a_i \mathbf{p}_i$$

$$\sum_{i=1}^{n} a_i = 1$$

Implications

- If $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ are co-planar points then $\mathbf{p}$ as defined will be inside ~~the polygon~~ (convex hull) defined by the points, if and only if

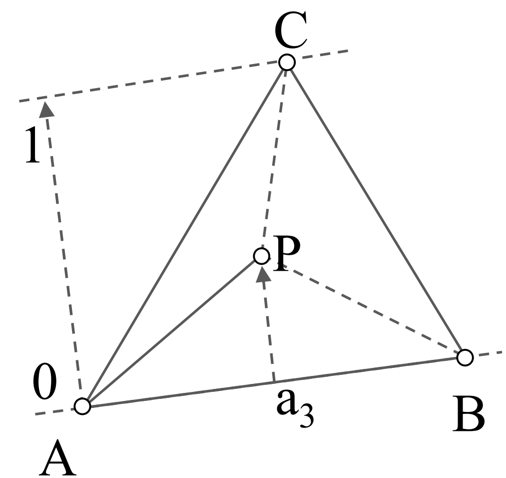
$$0 \le \alpha_i \quad \forall i$$

Barycenters, Computation

- Compute barycentric coordinates, and check if

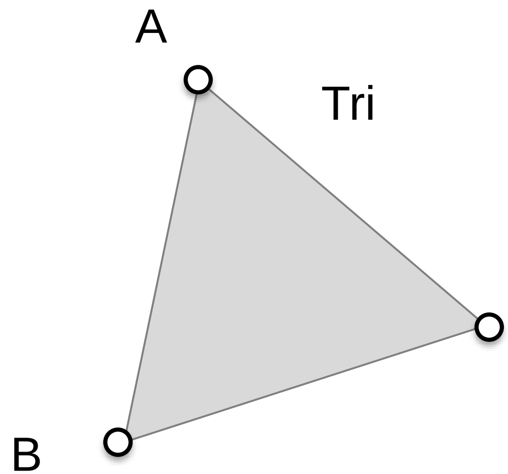
$$0 \le a_i \quad \forall i$$

- Compute barycentric coords with:
  - $a_1 = \Delta(PBC) / \Delta(ABC)$
  - $a_2 = \Delta(APC) / \Delta(ABC)$
  - $a_3 = \Delta(ABP) / \Delta(ABC)$
  - $\Delta(ABC)$ is the *signed* area of a triangle ABC
  - Computed using the determinant



Signed area

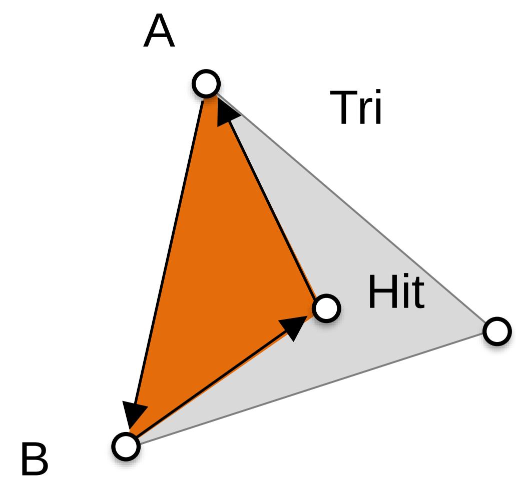
$$D(ABC) = \frac{1}{2} \begin{vmatrix} Ax & Bx & Cx \\ Ay & By & Cy \\ 1 & 1 & 1 \end{vmatrix}$$



Signed area

$$D(ABC) = \tfrac{1}{2}\begin{vmatrix} Ax & Bx & Cx \\ Ay & By & Cy \\ 1 & 1 & 1 \end{vmatrix}$$

```
area(A, B, hit)
```

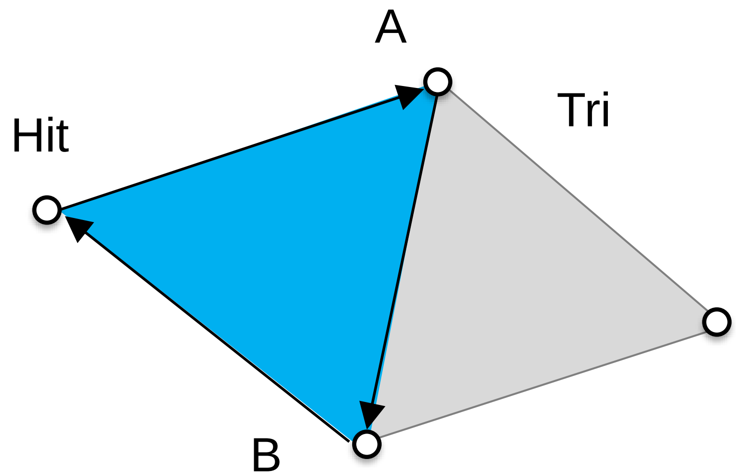


Positive

Signed area

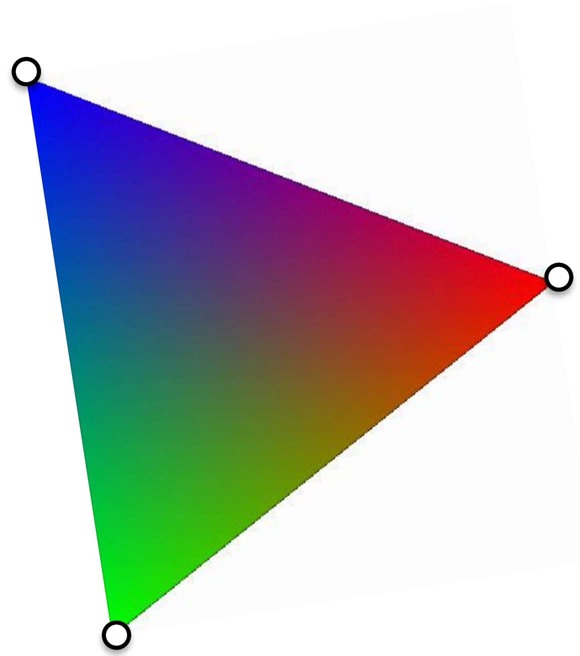
$$D(ABC) = \tfrac{1}{2}\begin{vmatrix} Ax & Bx & Cx \\ Ay & By & Cy \\ 1 & 1 & 1 \end{vmatrix}$$

`area(A, B, hit)`

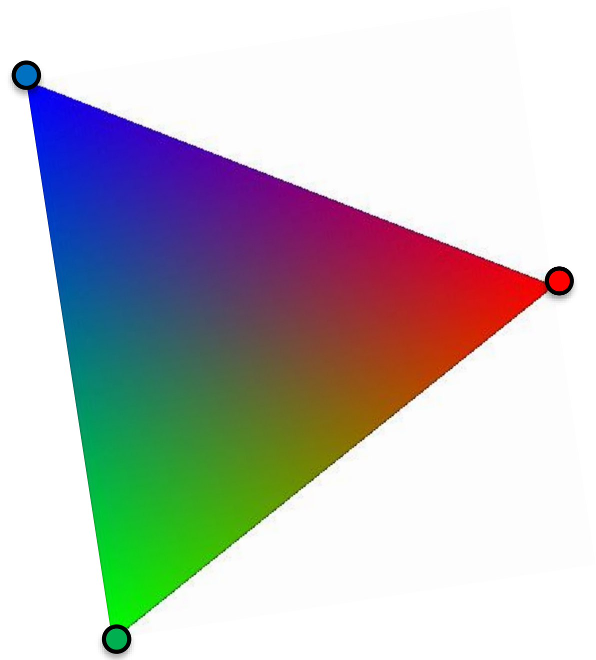


Negative

What is that tri again?

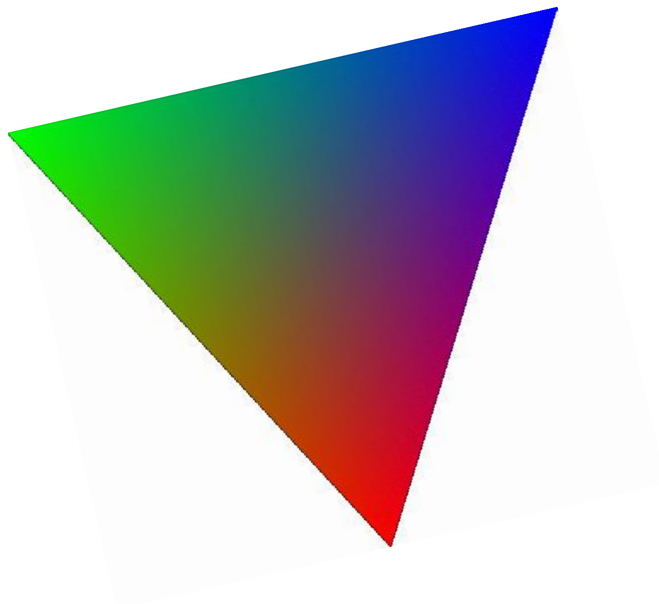


What is that tri again?



Barycentric coords are useful

- You get the test
- You also get weights that can be used to interpolate any other per-vertex quantity
  - Colours (what is on this slide)
  - Normals
  - Texture Coordinates
  - Anything



Recap

- We have seen how to ray-trace polygons, by turning the problem into a 2D problem
- We saw that we have to be clear what is inside a polygon
- The different tests are suitable in different situations: whether or not you need to know if the polygon was hit or not
  - E.G. if are doing collision detection you don't need to know where, but if you are doing you need texture coordinates
- The different algorithms have different efficiencies depending on whether you expect the ray to hit