Raytracing

Motivation

- We'll examine raytracing. Several reasons:
  - Practical problem
  - Allows us to see several techniques we can use for optimization
  - Becoming more widely used for real-time rendering

Raytracing

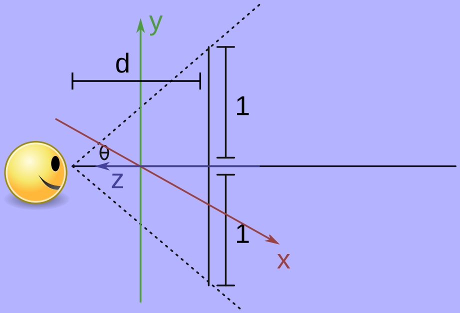
- For every pixel p:
  - Generate ray from eye through p
  - Find closest object intersected by ray
  - if there is one:
    - Shade pixel accordingly
  - else:
    - Use background color

Rays

- How to generate the rays?
- Suppose we are given a camera with these pieces of information:
  - Right: Vector
  - Up: Vector
  - Look: Vector
  - Eye: Point

Rays

- Next, we need to define the projection "screen"
  - To make it easier, we'll use square screen
- Given: Half-angle field of view, θ
- $d = \frac{1}{tan\,\theta}$



Rays

```
void render( Camera& cam ){
    float fov = radians(45.0);
    float d = 1.0 / tan(fov);
    float dy = 2.0/(height-1);
    float dx = 2.0/(width-1);
    float y=1.0;
    for(int pixy=0;pixy<height;++pixy,y-=dy){
        float x=-1.0;
        for(pixx=0;pixx<width;++pxx,x+=dx){
            vec3 rayDir = x*cam.right + y*cam.up - d*cam.look;
            color = traceRay(cam.eye,rayDir);
            setPixel( pixx, pixy, color );
        }
    }
}
```

Rays

- Now we know how to generate the rays
- How to intersect them with scene objects?
- We'll consider two types of objects: Spheres and triangles

Spheres

- Sphere is defined as all points p that satisfy equation: $||\vec{p} - \vec{c}|| = r$
  - Where $\vec{c}$ = center of sphere and r = radius
- How can we make use of this?

Sphere

- Write the formula in a little more convenient form: $\sqrt{(p_x - c_x)^2 + (p_y - c_y)^2 + (p_z - c_z)^2} = r$
- How do we relate this to our ray?

Ray

- If ray starts at s and has direction v: Any point on ray can be expressed as:
  $\vec{s} + t\vec{v}$
  - Where the scalar $t \geq 0$
- Substitute this for $\vec{p}$ in our previous equation

Ray

- We get:
$$\sqrt{(s_x + tv_x - c_x)^2 + (s_y + tv_y - c_y)^2 + (s_z + tv_z - c_z)^2} = r$$
- That radical sign is irritating. Let's get rid of it:
$$(s_x + tv_x - c_x)^2 + (s_y + tv_y - c_y)^2 + (s_z + tv_z - c_z)^2 = r^2$$
- To make things a little shorter, let $\vec{q} = \vec{s} - \vec{c}$:
$$(q_x + tv_x)^2 + (q_y + tv_y)^2 + (q_z + tv_z)^2 = r^2$$
- Nothing to do now but FOIL it out...

FOIL

- $q_x^2 + 2q_xv_xt + t^2v_x^2 + q_y^2 + 2q_yv_yt + t^2v_y^2 + q_z^2 + 2q_zv_zt + t^2v_z^2 = r^2$
- Collect like terms:
  $\vec{q} \cdot \vec{q} - r^2 + 2(\vec{q} \cdot \vec{v})t + t^2(\vec{v} \cdot \vec{v}) = 0$
  - Note: $\vec{q} \cdot \vec{q} = q_x^2 + q_y^2 + q_z^2$ (same idea for $\vec{v} \cdot \vec{v}$ and $\vec{q} \cdot \vec{v}$)
- Quadratic formula:
  $t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$
  - A=$\vec{v} \cdot \vec{v}$
  - B=$2(\vec{q} \cdot \vec{v})$
  - C= $\vec{q} \cdot \vec{q} - r^2$
  - If discriminant < 0: No intersection

Note...

- Quadratic formula gives two solutions: t_1 and t_2
- Desired value for t:
  - If $t_1 < 0$ and $t_2 < 0$: No intersection (behind starting point)
  - If $t_1 < 0$ and $t_2 \geq 0$: t_2
  - If $t_1 \geq 0$ and $t_2 < 0$: t_1
  - If $t_1 \geq 0$ and $t_2 \geq 0$: $\min(t_1, t_2)$

Intersection

- Once we have t: Compute intersection point ip:
  $\vec{ip} = \vec{s} + t\vec{v}$
- We also know the normal at the intersection point (useful for shading):
  $\vec{N} = \vec{ip} - \vec{c}$

Triangle

- What about ray-triangle intersection?
- Two parts:
  - Find intersection of ray with plane that contains triangle
  - See if intersection point is inside triangle

Plane

- Recall definition of a plane:
  - Set of all points (x,y,z) such that Ax + By + Cz + D = 0
    - Where A,B,C = Plane normal
    - D = Distance of plane from origin
- If we know triangle T is made up of points p,q,r:
  - $\vec{N} = (\vec{q} - \vec{p}) \times (\vec{r} - \vec{p})$
  - $D = -(\vec{N} \cdot p)$

Plane

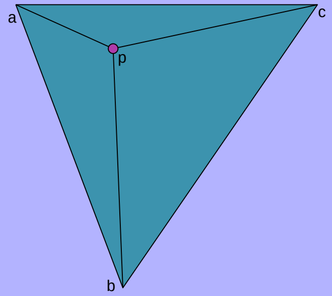
- Plug ray equation into planar equation:
  $A(s_x + tv_x) + B(s_y + tv_y) + C(s_z + tv_z) + D = 0$
- Multiply out:
  $As_x + Av_xt + Bs_y + Bv_yt + Cs_z + Cv_zt + D = 0$
- Collect terms:
  $(\vec{N} \cdot \vec{v})t = -(D + (\vec{N} \cdot \vec{s}))$
- Divide:
  $t = \frac{-(D+(\vec{N}\cdot\vec{s}))}{\vec{N}\cdot\vec{v}}$
  - If denominator is zero: No intersection
  - If t < 0: No intersection

Intersection

- Now that we know t, we can find the intersection point:
  $\vec{ip} = \vec{s} + t\vec{v}$
- But is this point inside the triangle?
- Several ways to do this test. We'll look at *barycentric coordinates*

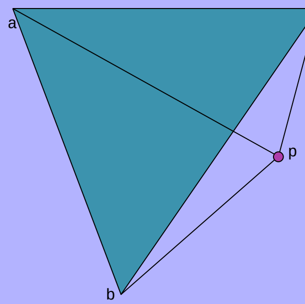
Barycentric

- Suppose we have a point p inside a triangle T
- Connect p to each vertex of T. This creates three triangles
- The area of the three little triangles sums up to the area of T



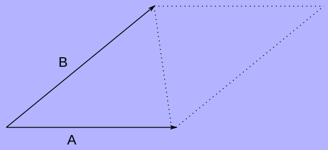
Barycentric

- Suppose we have a point p outside triangle T
- If we consider the three triangles generated by connecting p to each vertex of T, notice the area of those three triangles is now *greater* than that of T
  - Triangles: abp, apc, pcb



Area

- How do we find area of triangle?
- Can use cross product
- If $C = A \times B$:
  - length(C) = area of parallelogram defined by the two vectors A and B
- And area of parallelogram = 2 * area of triangle defined by A and B



Code

- Let $\vec{p}_i$ be vertex i of the triangle (i=0,1,2), let $\vec{e}_i$ be edge i of the triangle ($\vec{e}_i = \vec{p}_{i+1} - \vec{p}_i$), let $\alpha = \frac{1}{2 \cdot triangleArea}$
  - $\vec{ip} = \vec{s} + t\vec{v}$
  - $\vec{q}_i = \vec{e}_i \times (\vec{ip} - \vec{p}_i)$
  - $A = \left(\sum_{i=0}^{2} ||\vec{q}_i|| \right) \cdot \alpha$
  - If $A \leq 1.001$: We have an intersection
    - 1.001 accounts for floating point error

Code

- What if we need to do lighting?

```
vec3 shadePixel(vec3 objColor, vec3 lightPosition, vec3 ip, vec3
    N){
     L = normalize(lightPosition - ip);
     dp = dot(N,L);
     dp = max(dp,0.0)
     return dp * objColor;
}
```

Assignment

- Finish the raytracer: Write the routines to do the tracing of spheres and triangles
- Codebase

Sources

- https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org