Making Virtual Worlds

Computer Graphics Lecture 1

Nicholas Dwork

Making Virtual Worlds

We must make the objects in our virtual world.
We must color the objects in our virtual world.

We can add light sources in our virtual world.
E.g. the sun, a light bulb
We can give our objects reflectances.
We can specify how much specular and diffuse reflectivity each object has.

We'll make videos of our virtual world!

Creating a Coordinate System

Our world will be a 3D Euclidean Space.

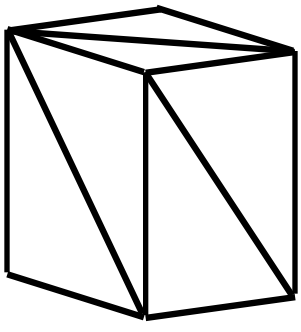
This space has an origin.

Everything will be relative to this point.

Making Objects

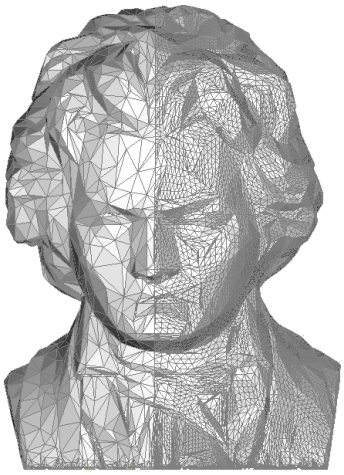
Objects are constructed from triangles.

Here I've made a box from triangles.



We "make" the triangles by specifying the coordinates of each vertex.

Face From Triangles



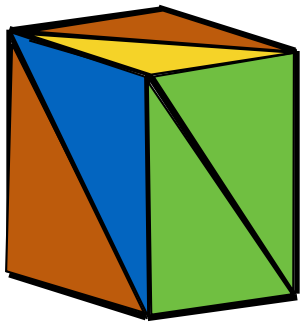
http://www.final-surface.com/refining.php

Adding Color

We can add color to our objects.

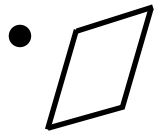
The simplest way of adding color is to specify an RGB value for each triangle.

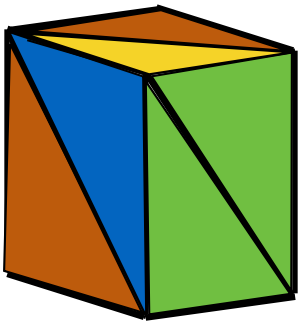
This is how we'll start.



Placing the Camera

Once we have constructed our objects from triangles, we can image them with our virtual camera.





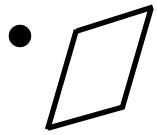
Adding Color to Mesh

Video

http://izismile.com/2014/02/12/beautiful_photorealistic_human_eye_in_3d.html

Placing the Camera

Once we have constructed our objects from triangles, we can image them with our virtual camera.

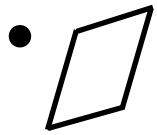


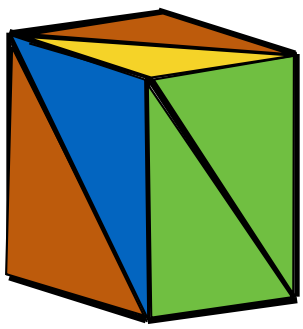
We must specify the intrinsic parameters of the camera, its location, and its roll, pitch, and yaw.

This completely specifies the camera matrix.

We already know how to do everything in this lecture so far.

We'll now make the image, which will be the focus of the rest of this lecture.





Ray Tracing

For each pixel of the image

1. **Create a line that connects the pixel and the camera center**
   - **Point the line forward**

2. **For each triangle in the virtual world**
   - **Intersect the line with the plane containing the triangle**
   - **See if the intersection lies within the triangle**
   - **If it lies within the triangle, store the color and distance**

3. **Color the pixel with the color of the closest triangle**

Suppose P is the camera matrix. Then we know the relationship between 3D points and image locations.

$$x = P\,X$$

Up until now, we've understood this equation by saying "we can map a point in the world X to a location in the image x".

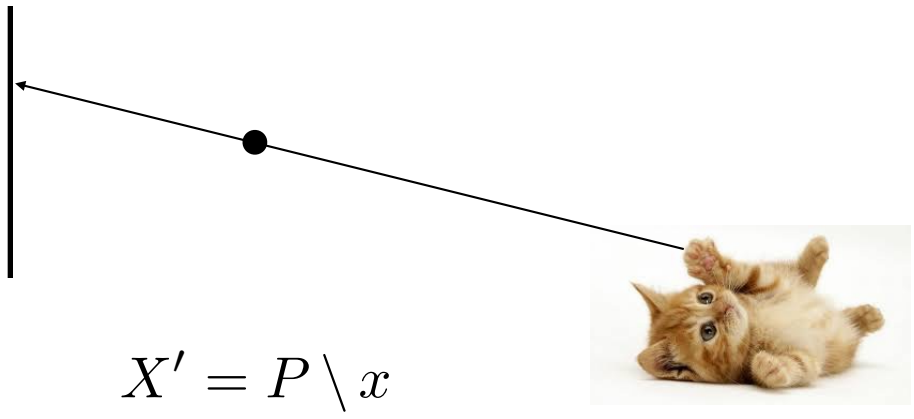
We can use P to map an image point to a world point.

$$X' = P \setminus x$$

Note that P is not square, and so it can't be invertible.

This makes sense!

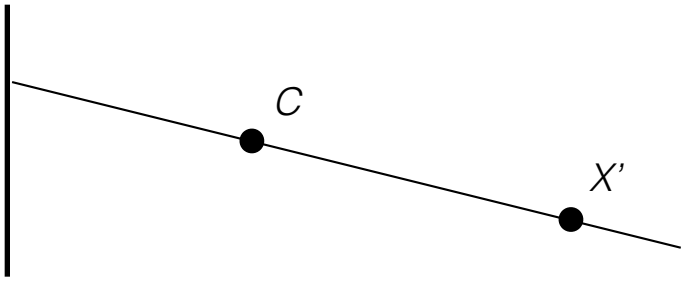
Any point on the line between the cat's paw and the image plane maps to the same image point.



$$X' = P \setminus x$$

But then, what is X' ?

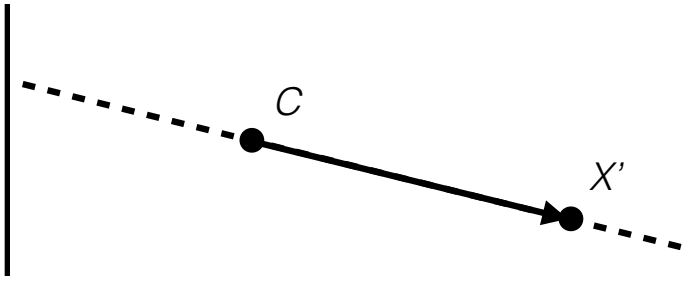
X' **is some point on the line, but we don't know which one.**



C **is the camera center.**

X' - C **is a vector on the line (or ray) that intersects the specified pixel.**

Now that we have a vector that is parallel to the line and a point on the line (*C*), we can write the equation of the "ray".



$$f(t) = C + t\,(X' - C)$$

Ray Tracing

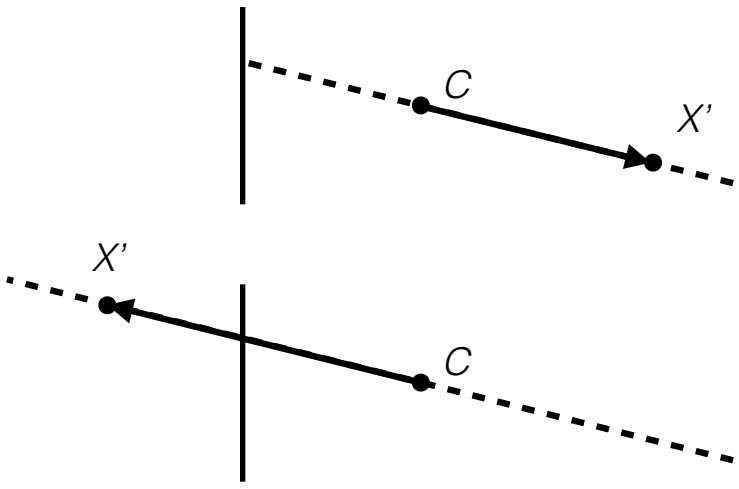
For each pixel of the image

1. **Create a line that connects the pixel and the camera center**
   - **Point the line forward**

2. **For each triangle in the virtual world**
   - **Intersect the line with the plane containing the triangle**
   - **See if *f(t)* lies within the triangle**
   - **If it lies within the triangle, store the color and distance**

3. **Color the pixel with the color of the closest triangle**

Since any point on the line projects to the same pixel, we don't know which way *(X' - C)* is pointed.



$$f(t) = C + t\,(X' - C)$$

$$P = K\,R\,\begin{bmatrix} I & -C \end{bmatrix}$$

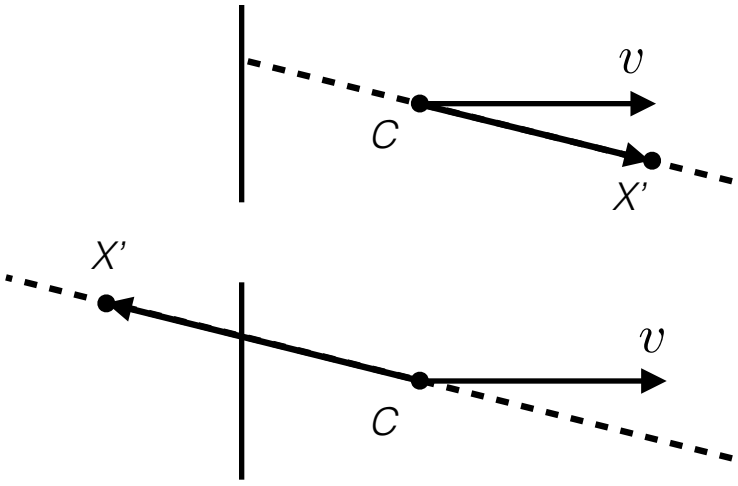
The pointing direction of the camera is completely specified by the matrix *R*.

The roll of the camera doesn't matter because changing the roll doesn't change the principal axis of the camera.

So intuitively, we don't even need all of *R*.

In fact, the third row of *R* is a vector on the principal ray.

$$v = r^{3T}$$



Recall: The dot product can be used to determine whether or not the angle between two vectors is acute or obtuse.

If $v \cdot (X' - C) > 0$ **then** $r = X' - C$

If $v \cdot (X' - C) < 0$ **then** $r = C - X'$

$$f(t) = C + t\,r$$

Now we know that if t is positive, then the point is in front of the camera. Otherwise, the point is behind the camera.

Ray Tracing

For each pixel of the image

1. **Create a line that connects the pixel and the camera center**
   - **Point the line forward**

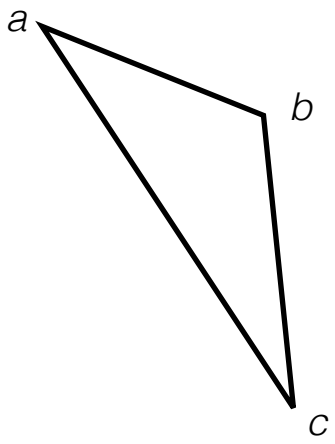
2. **For each triangle in the virtual world**
   - **Intersect the line with the plane containing the triangle**
   - **See if the intersection lies within the triangle**
   - **If it lies within the triangle, store the color and distance**

3. **Color the pixel with the color of the closest triangle**

Equation of Plane Containing Triangle

For the equation of a plane, we need:
- **A normal vector**
- **A point on the plane**

We have:
- **The vertices of the triangle**



Equation of Plane Containing Triangle

Any vertex can be a point in the plane.

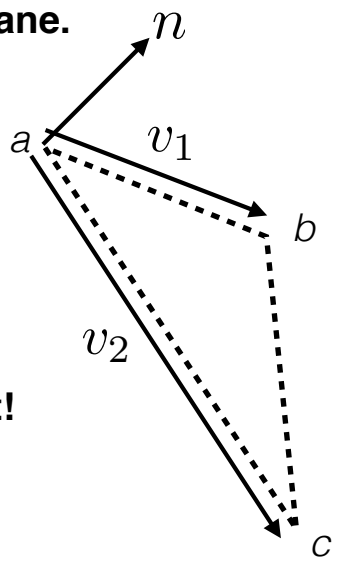
We still need a normal vector.

$$v_1 = b - a$$

$$v_2 = c - a$$

Now we can use the cross product!

$$n = v_2 \times v_1$$



Line-Plane Intersection

The ray is represented as $f(t) = C + t\,r$

The plane containing the triangle is represented as

$$n \cdot (p - a) = 0$$

where *a* **is any vertex, and** *n* **is the normal vector.**

We can combine these equations into one.

Line-Plane Intersection

$$f(t) = C + t\,r$$

$$n \cdot (p - a) = 0$$

We need the value of *t* **so that** $n \cdot (f(t) - a) = 0$

Equivalently, $n \cdot (C + t\,r - a) = 0$

$$t = \frac{n \cdot (a - C)}{n \cdot r}$$

Now that we know *t*, *f(t)* **is the point where the ray intersects the plane.**

If *t* **is negative, we discard this intersection and go on to the next triangle.**

Ray Tracing

For each pixel of the image

1. **Create a line that connects the pixel and the camera center**
   - **Point the line forward**

2. **For each triangle in the virtual world**
   - **Intersect the line with the plane containing the triangle**
   - **See if the intersection lies within the triangle**
   - **If it lies within the triangle, store the color and distance**

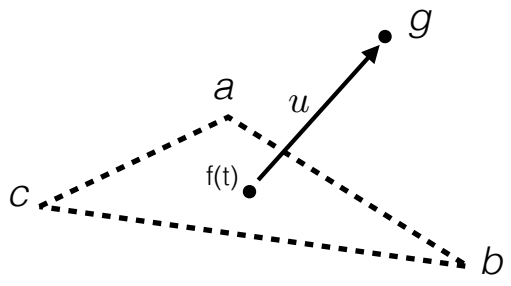
3. **Color the pixel with the color of the closest triangle**

Intuition

We will create a vector from $f(t)$ to a point outside the triangle.

We will count the number of times this vector crosses and edge.

If the number is odd, then the point lies inside the triangle. Otherwise, it's outside.

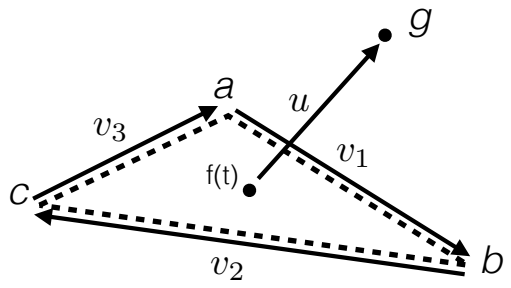


A Distant Point

Calculate the three edge vectors v_1, v_2, v_3 **.**
Note that edge vectors are tip-to-tail.

Add v_3 **to** a **to get a point** g **outside the triangle.**

Calculate vector $u = g - f(t)$ **.**



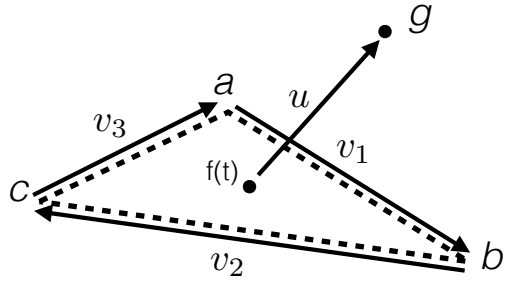
Crossings

Set number of crossings to *0*.
For each edge vector

Find the intersection between the line containing the edge vector and u **.**

If the intersection point lies between *f(t)* **and** g**, add** *1* **to the number of crossings.**

If the number of crossings is odd, then the point is inside the triangle.



Ray Tracing

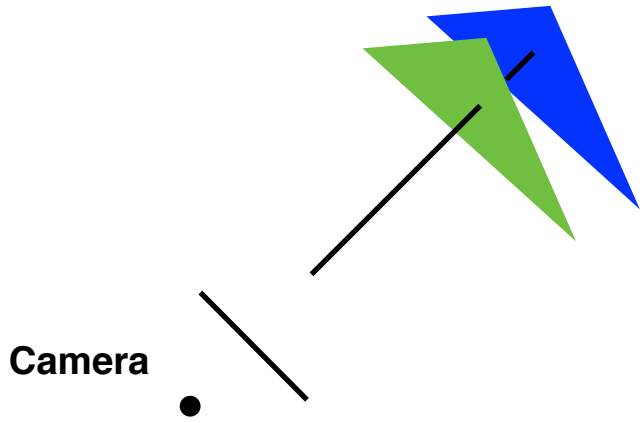
For each pixel of the image

1. Create a line that connects the pixel and the camera center

2. For each triangle in the virtual world
   - Intersect the line with the plane containing the triangle
   - Make sure the intersection is in front of the camera
   - See if the intersection lies within the triangle
   - If it lies within the triangle, store the color and distance

3. Color the pixel with the color of the closest triangle

The ray shown would see the green triangle but not the blue triangle.



We can now make a virtual world!