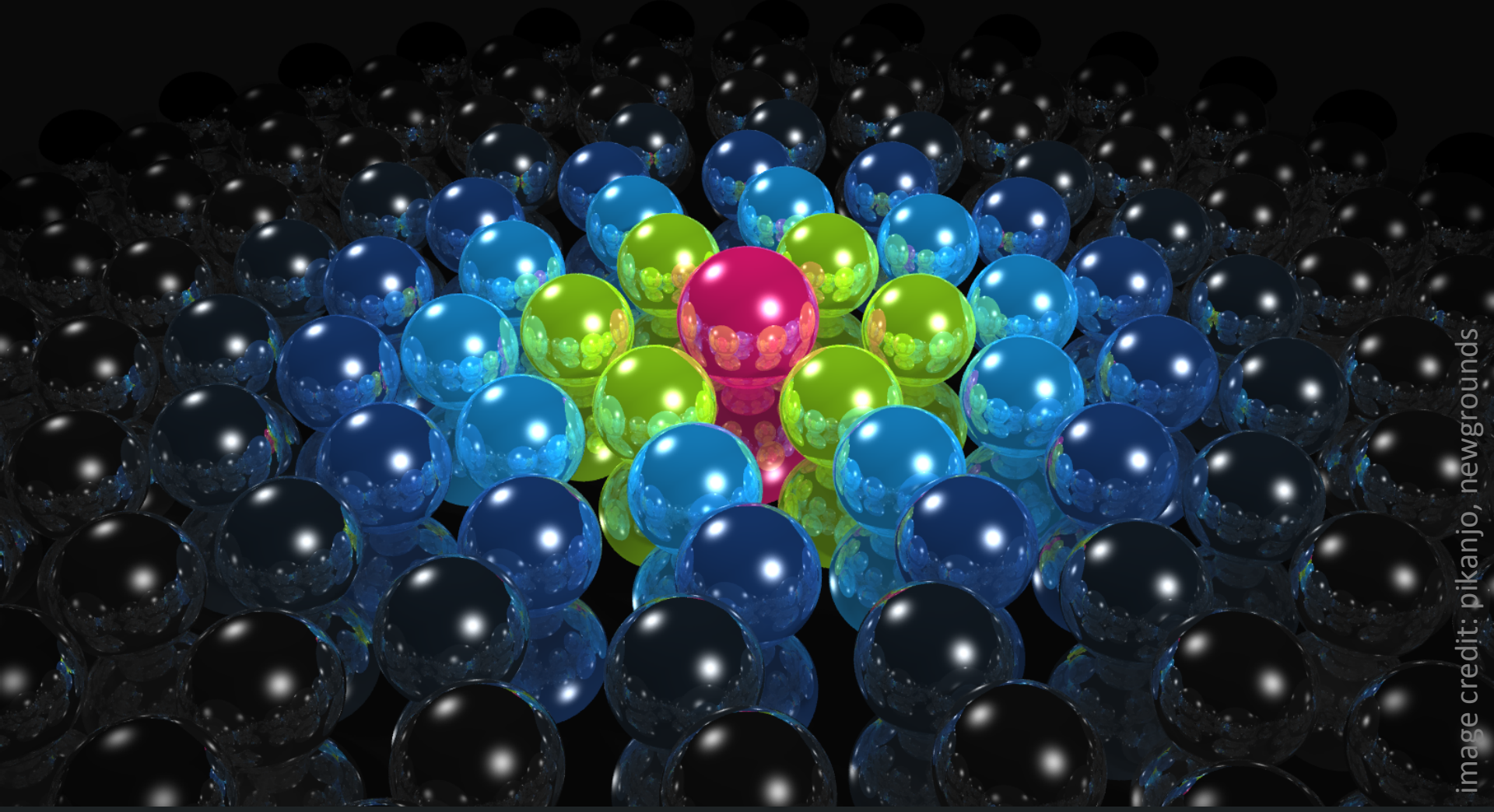


Introduction to Raytracing

also today: 2d review, project submission guidelines

CS 4300/5310

Computer Graphics

ANNOUNCEMENTS

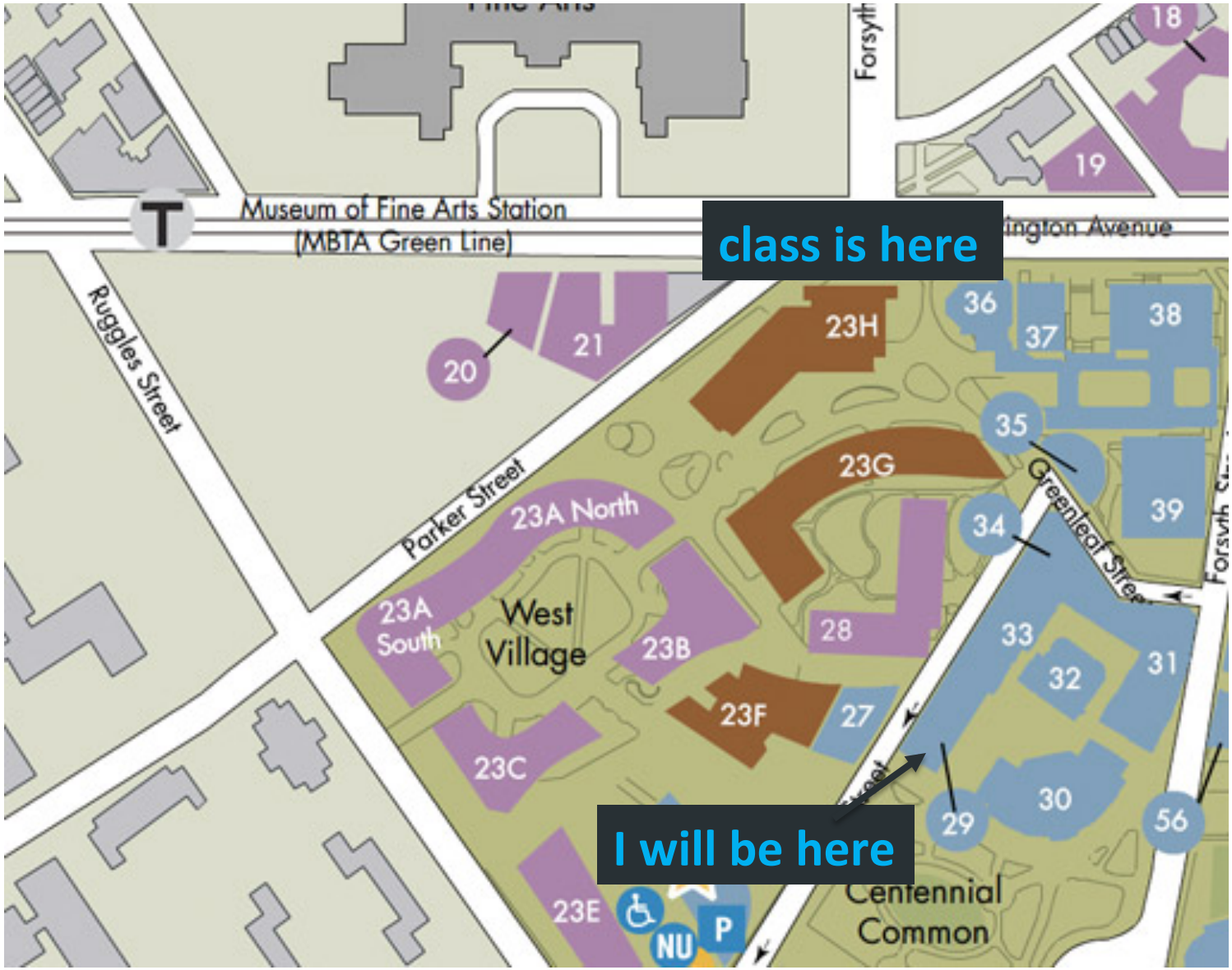
Deadline: 2D Project

- 2D Project main deadline: February 5th
  - That's in 7 days!

My office is moving on Wednesday

- Office hours today are cancelled!
- New office: Meserve Hall, Room 146
- Directions
  - Go through unmarked grey door facing Centennial Common
  - Go up a few stairs, left through the door at top
  - Follow signs for game design, along right hallway

New Office Location



Enter the Labyrinth Here

2D REVIEW

WELCOME TO 3D!

What is 3D Graphics?

3D Rendering: Rasterization

- Also called **object-order** rendering
- For each object, figure out what pixels it occupies
- Used in almost all realtime graphics, many movies

3D Rendering: Raytracing

- Also called **image-order** rendering
- For each pixel, figure out what color it is based on what objects it covers
- Realistic graphics with fancy lighting, reflections, refractions
  - Can be used in combination with rasterization

Examples: Rasterization

Examples: Rasterization

Examples: Raytracing

Examples: Raytracing

Examples: Raytracing

Tradeoffs Between Approaches

- Rasterization
  - Fast for simple scenes
  - One triangle at a time, interactions between them is harder
  - Specialized, “hacky” techniques for fancy lighting effects

Tradeoffs Between Approaches

- Raytracing
  - Physical simulation of the world
  - Interaction between objects is easy
  - Relatively simple model, easy to extend to many shapes
  - Slow, per pixel calculations
    - But very parallelizable!
    - Recently managed to do this in realtime
  - Used for photorealistic effects

Tradeoffs Between Approaches

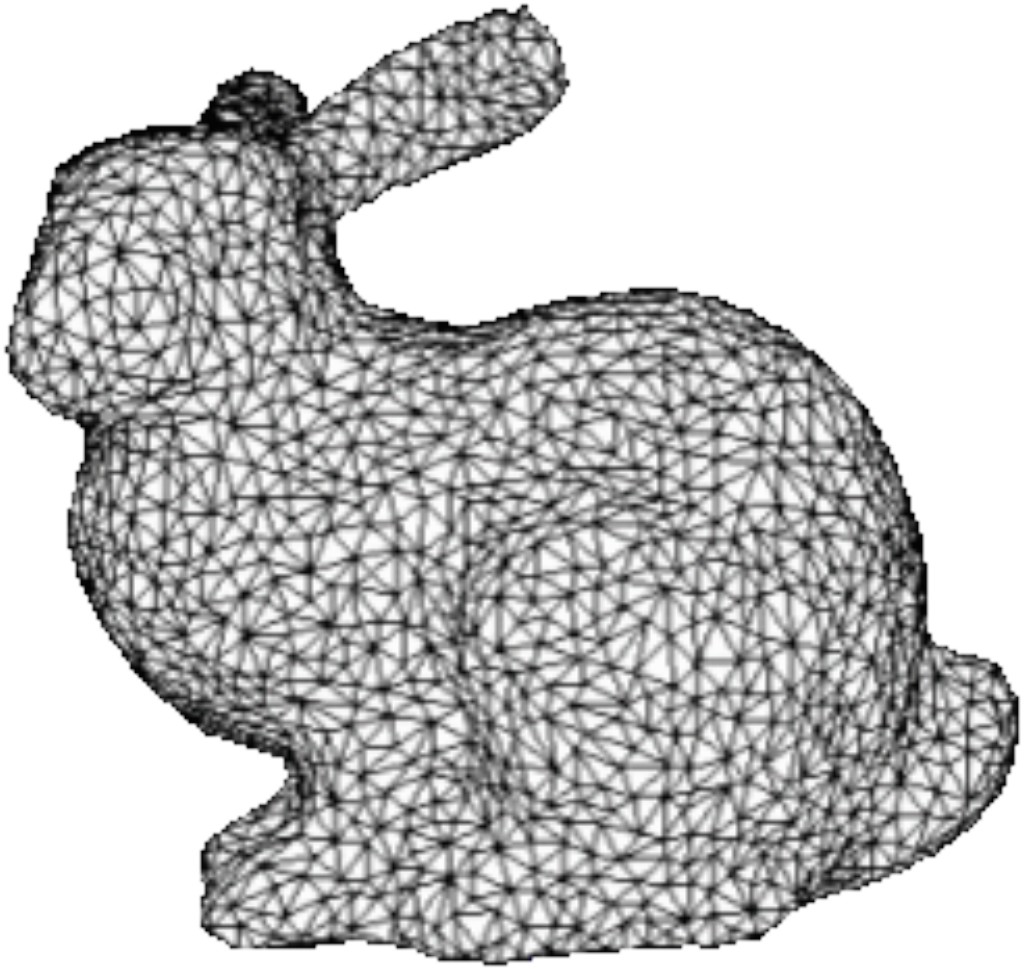
- A hybrid model?
  - Rasterization for faster rendering
  - Raytracing for lighting effects

Approaches to 3D Modeling

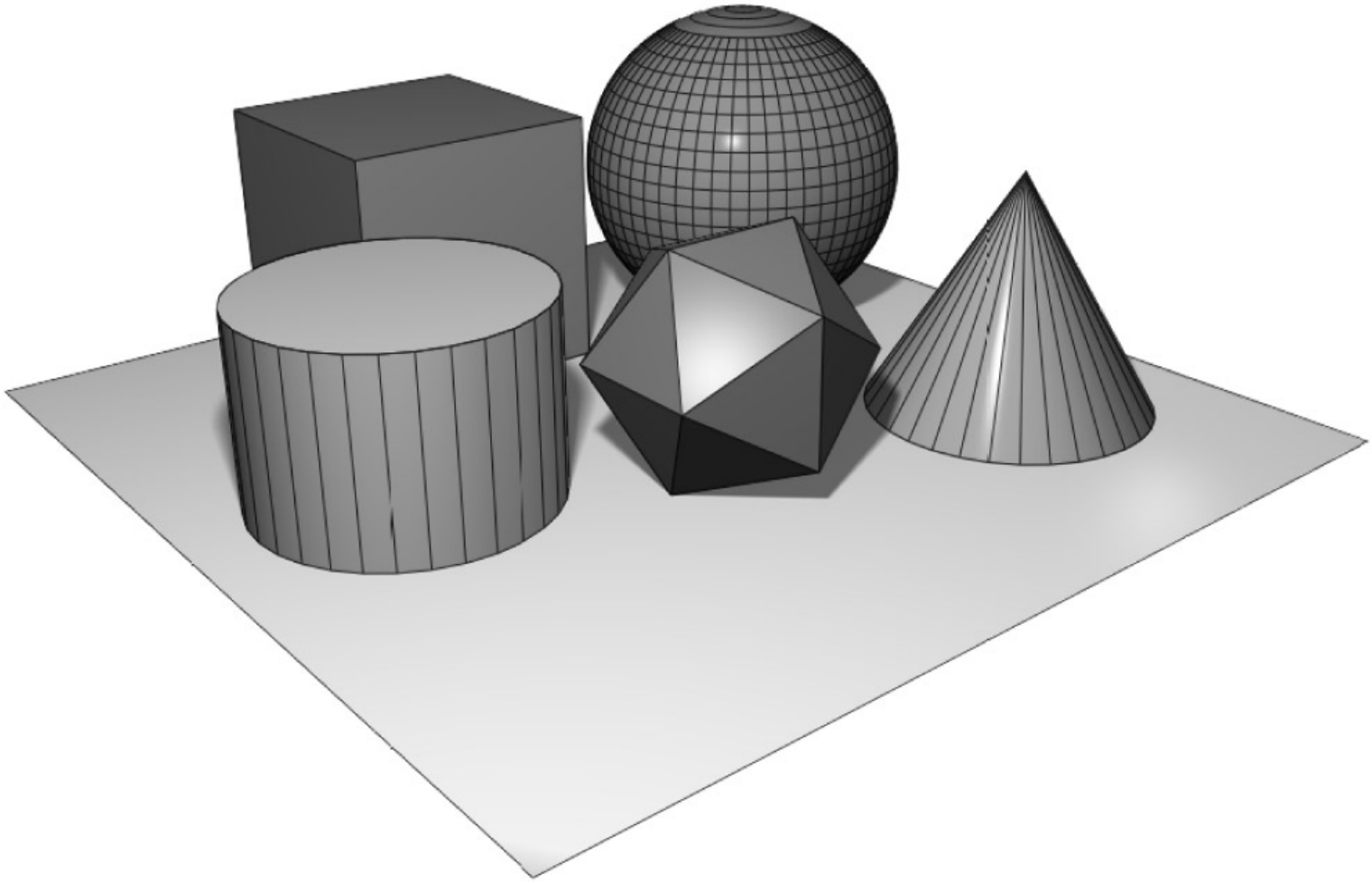
- Voxels
- Triangle meshes
- Arbitrary shapes

Voxels

Triangle Meshes

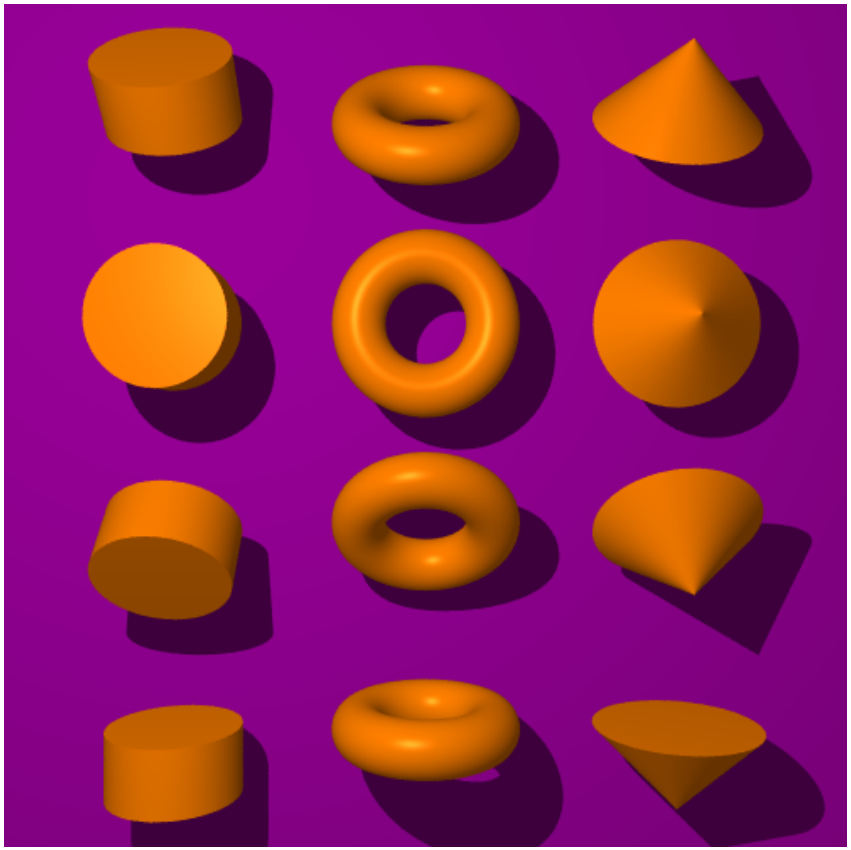


Arbitrary Shapes



using polygonal modeling

Arbitrary Shapes



using mathematical representation

RAYCASTING

Quick Math Review

- Points
- Vectors
- Rays

- Dot product
- Vector norm
- Cross product

Algorithm Overview

for every pixel, *p*, in the image

    let *r* be the ray from camera through pixel

    calculate intersection between *r* and scene objects

    if (!intersection)

        set *p* to “background color”

    else

        p = color calculated from first object it hits

Algorithm Overview

for every pixel, *p*, in the image

    let *r* be the ray from camera through pixel

    calculate intersection between *r* and scene objects

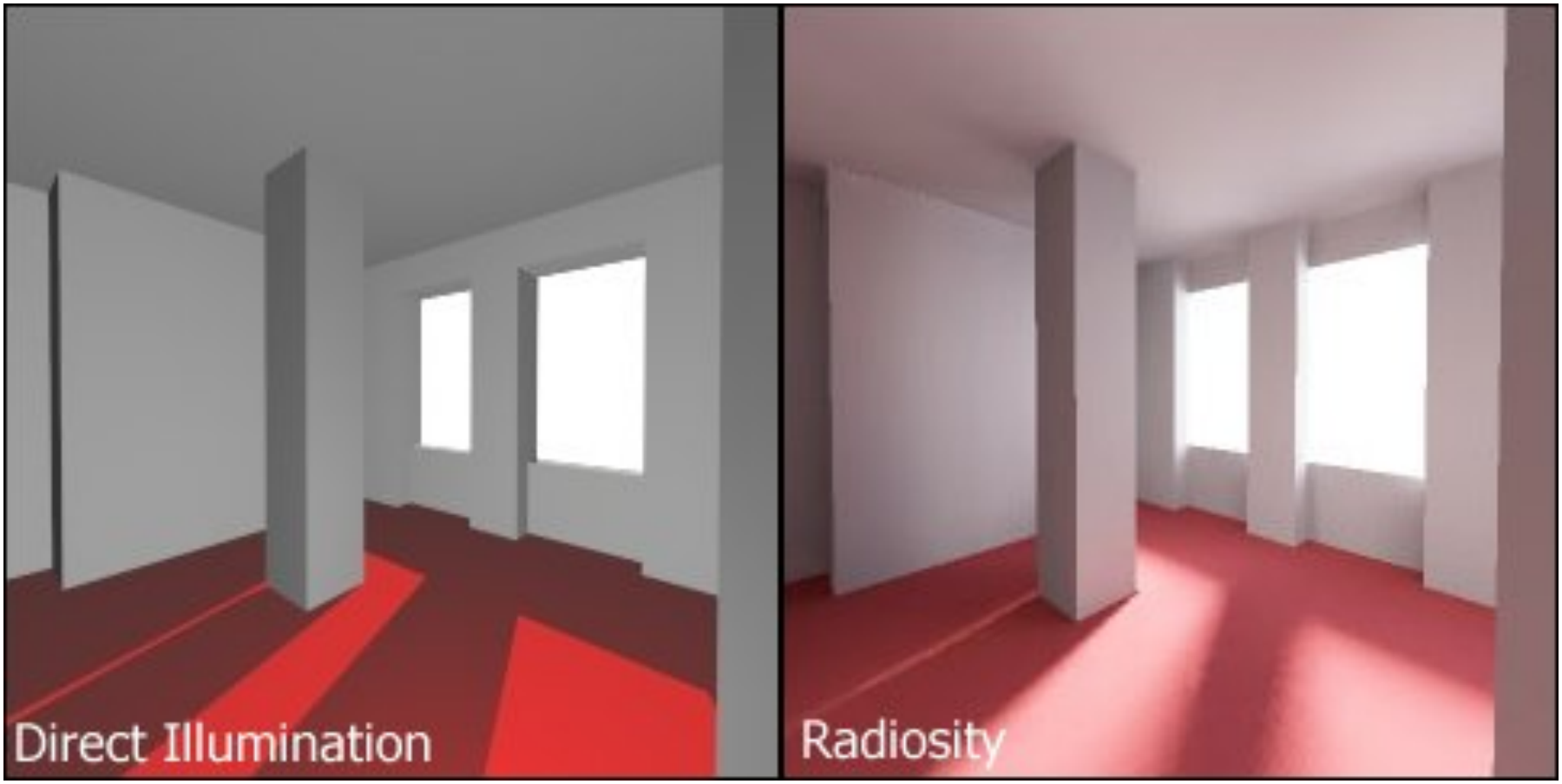
    if (!intersection)

        set *p* to “background color”

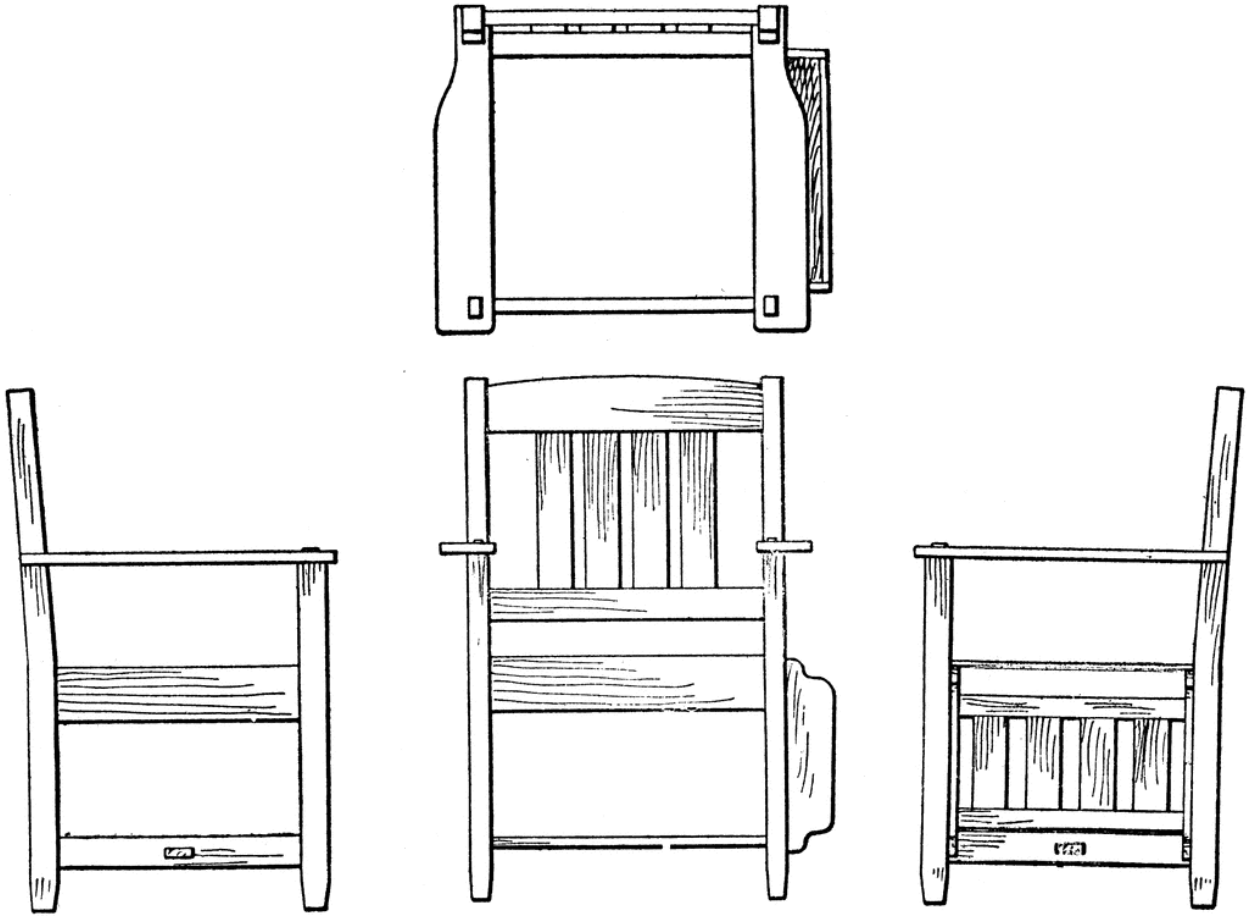
    else

        p = color calculated from first object it hits

Radiosity (global illumination)

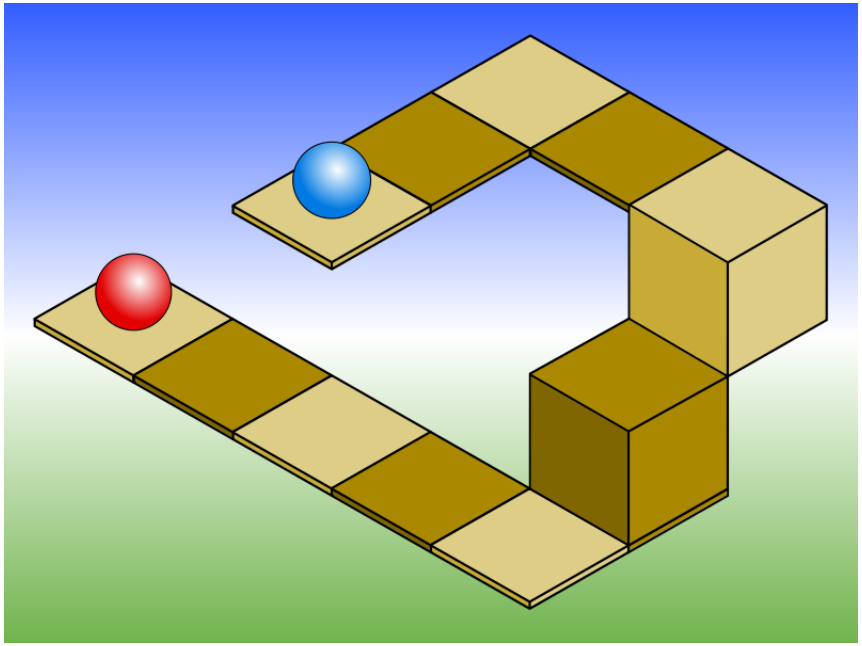


The Camera: Projections



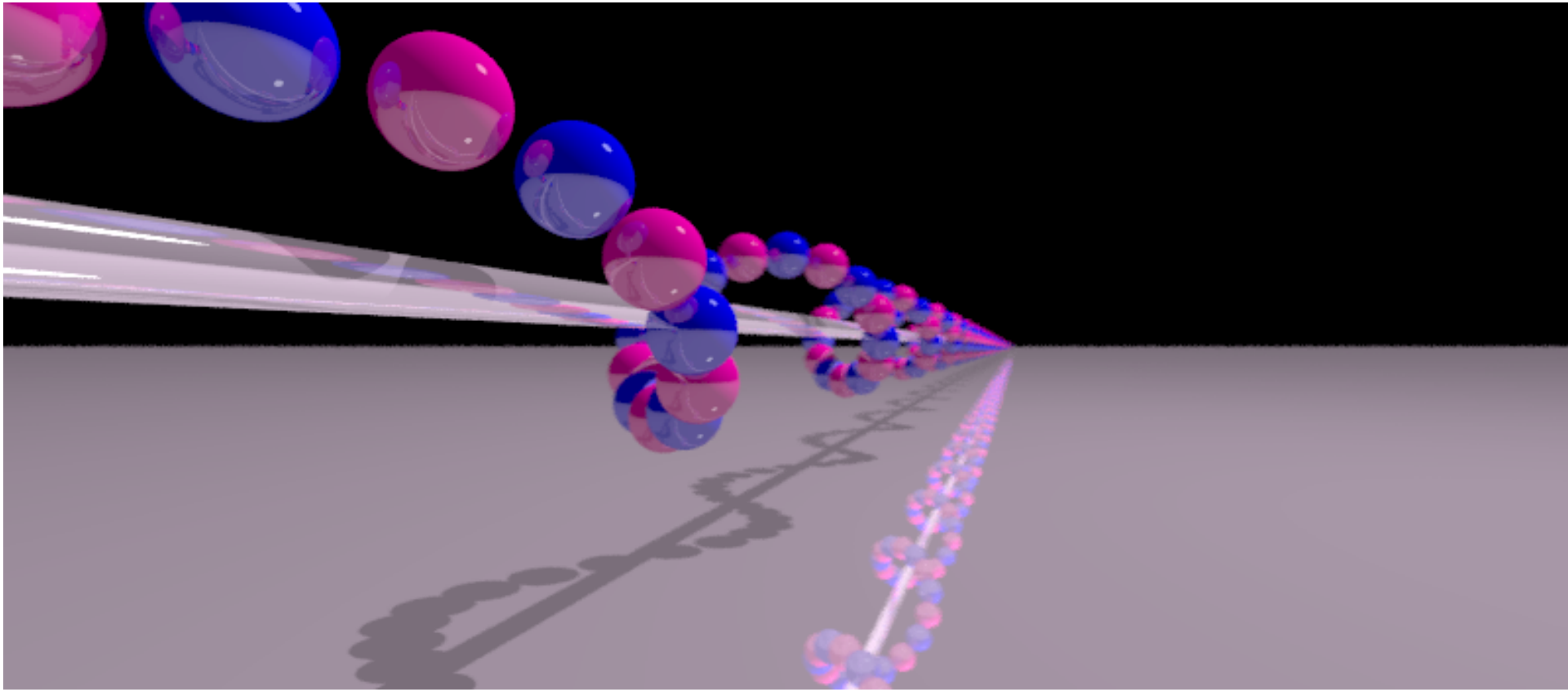
orthographic projection

The Camera: Projections



isometric projection

The Camera: Projections



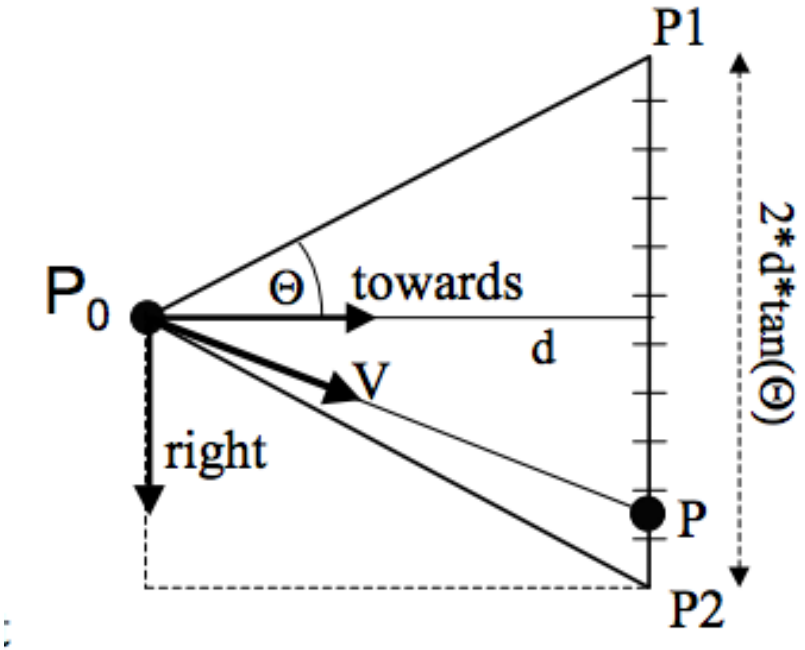
perspective projection

Orthographic Projection

- No single camera position
  - Or, camera is infinitely far away from viewing plane
- All rays through the pixel are parallel to each other and the z-axis

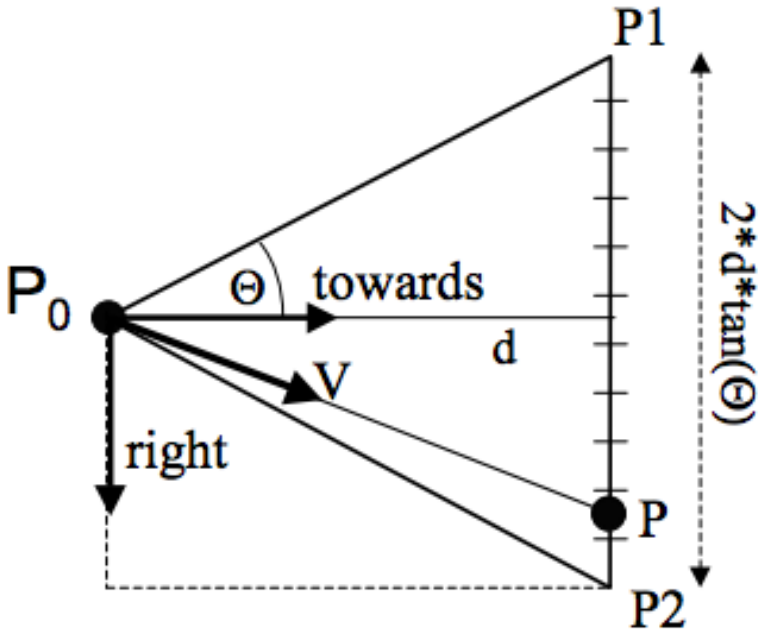
Perspective Projection

- Camera position: P_0
- Frustum angle: θ
- Camera vectors:
  - Up (*U*)
  - Towards (*T*)
  - Right (*R*)



Perspective Projection

- Camera position: P_0
- Frustum angle: θ
- Camera vectors:
  - Up (*U*)
  - Towards (*T*)
  - Right (*R*)

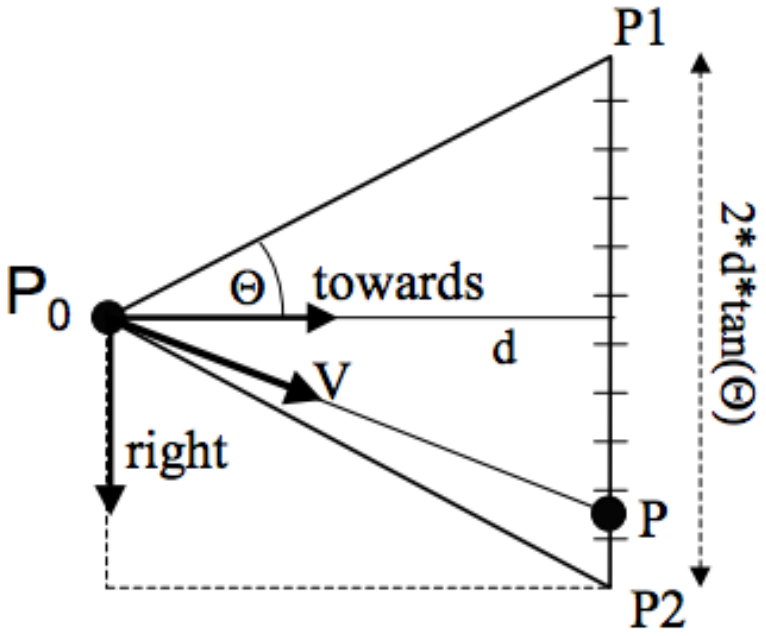


$P1 = P_0 + d*towards - d*tan(\theta)*right$

$P2 = P_0 + d*towards + d*tan(\theta)*right$

Perspective Projection

- Camera position: P_0
- Frustum angle: θ
- Camera vectors:
  - Up (*U*)
  - Towards (*T*)
  - Right (*R*)



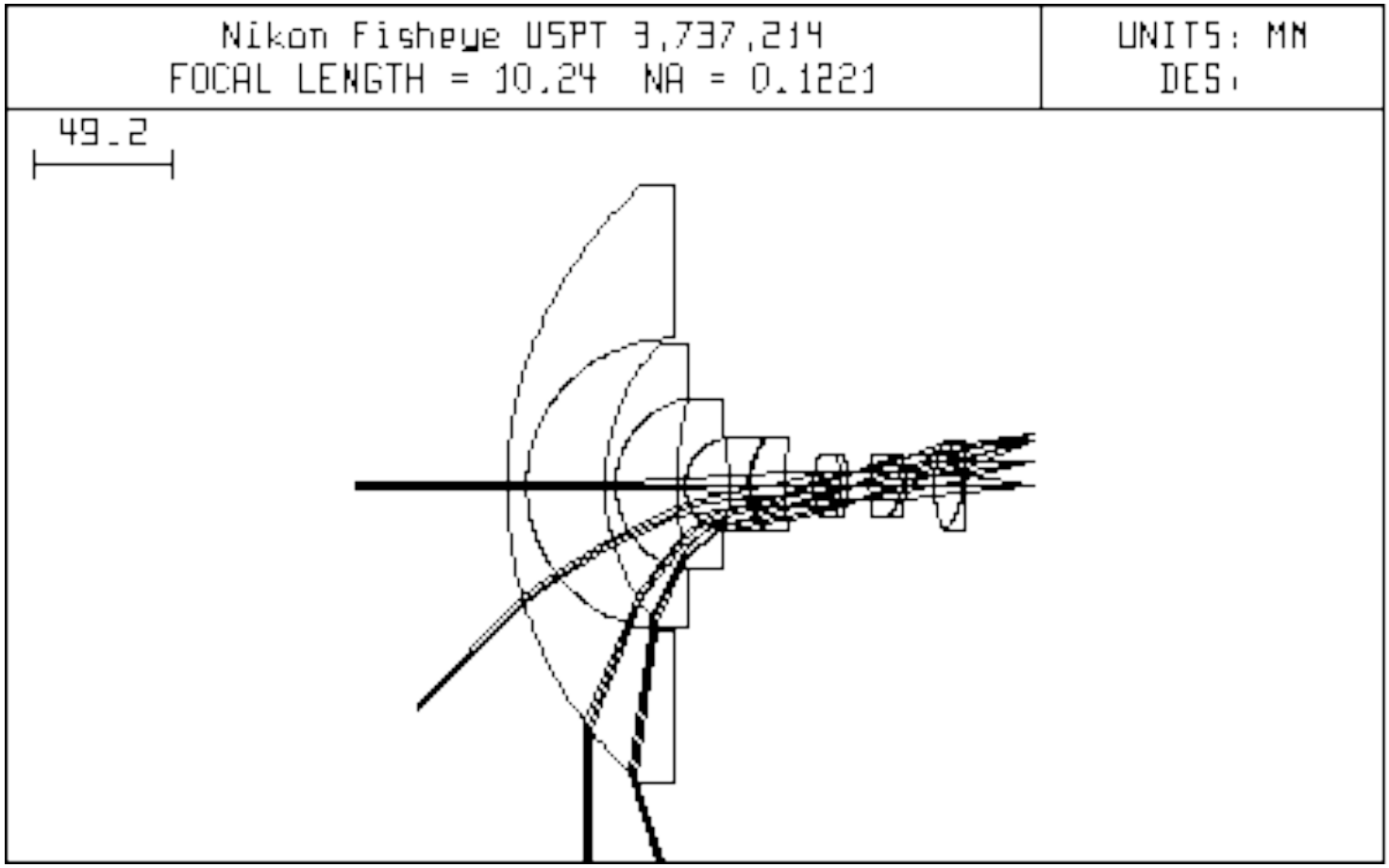
$P1 = P_0 + d*towards - d*tan(\theta)*right$

$P2 = P_0 + d*towards + d*tan(\theta)*right$

$P = P1 + (i/width + 0.5) * 2 * d * tan(\theta)*right$

$V = (P - P_0)/||P - P_0||$

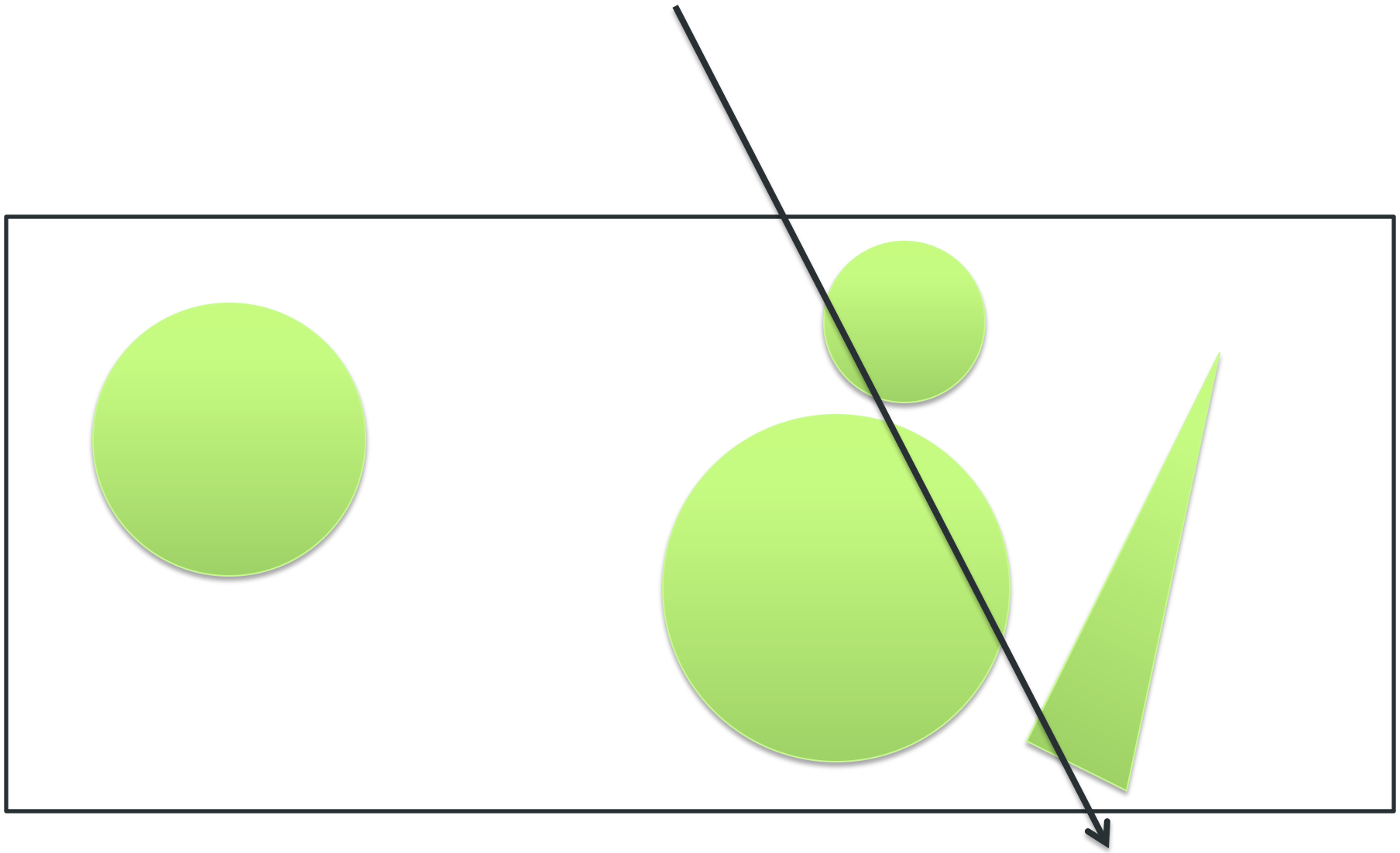
Special Camera Effects



Algorithm Overview

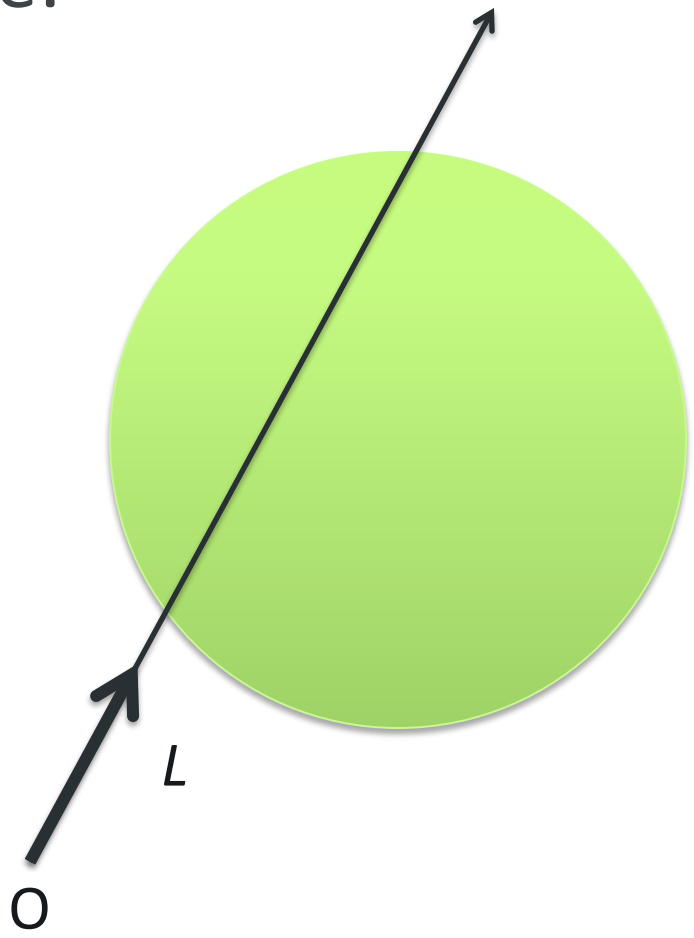
```
for every pixel, p, in the image
        let r be the ray from camera through pixel
        calculate intersection between r and scene objects
        if (!intersection)
                set p to "background color"
        else
                p = color calculated from first object it hits
```

Intersecting with Scene Objects



Spheres

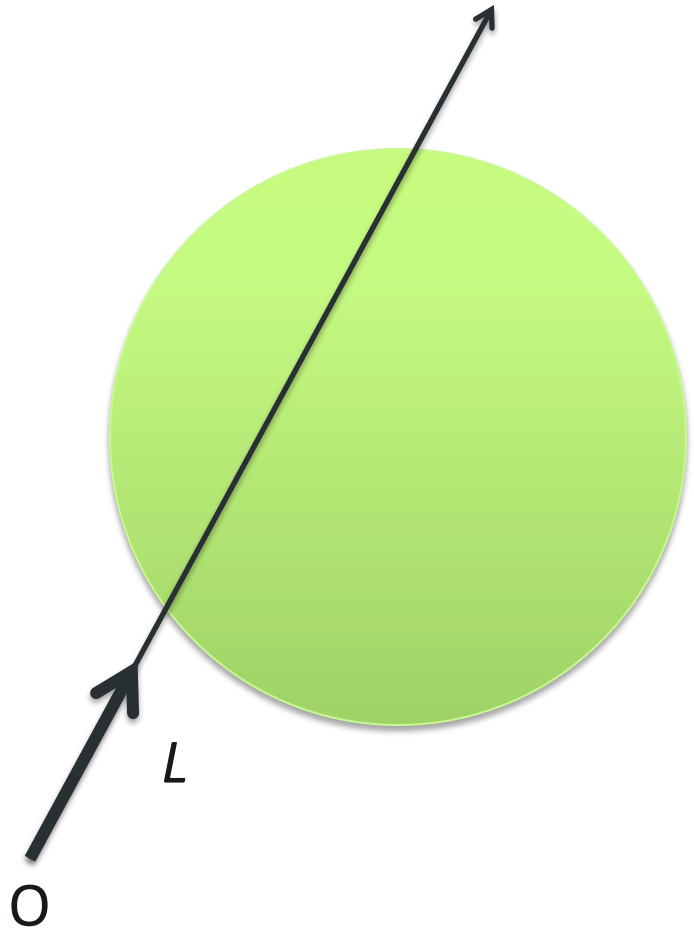
- Representation of a sphere:
  - Center, *C*
  - Radius, *r*
  - $||x - c||^2 = r^2$
- Ray:
  - Origin, *O*
  - Direction, *L*
  - Distance along line, *d*
  - $x = O + dL$



Spheres

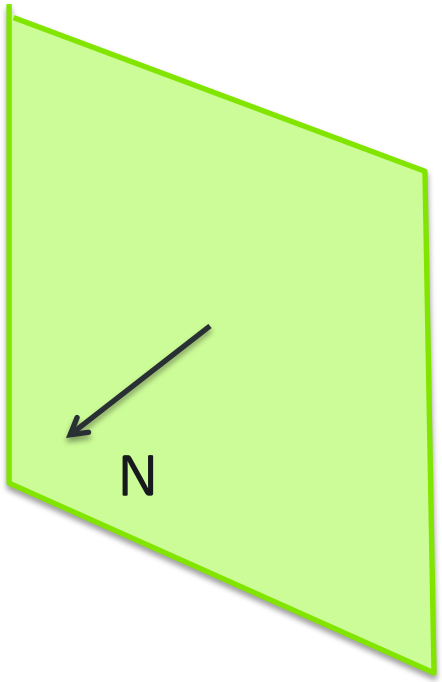
- $||O + dL - C||^2 = r^2$

- Solve for *d*….



Planes

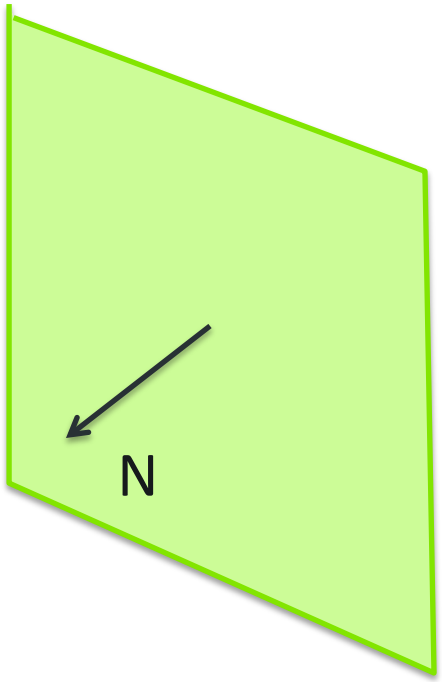
- Plane:
  - $x \cdot N + depth = 0$
- Ray:
  - Origin, O
  - Direction, L
  - Distance along line, d
  - $x = O + dL$



Planes

- $(O + dL) \cdot \mathrm{N} + \mathrm{depth} = 0$

- Solve for d?

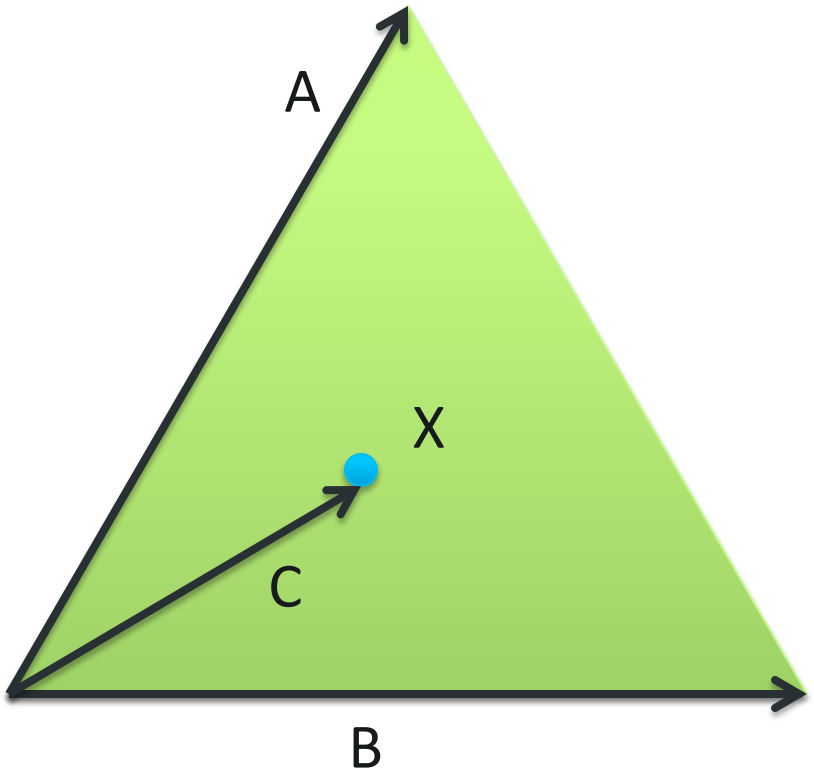


Triangles (Strategy I)

Triangles (Strategy I)

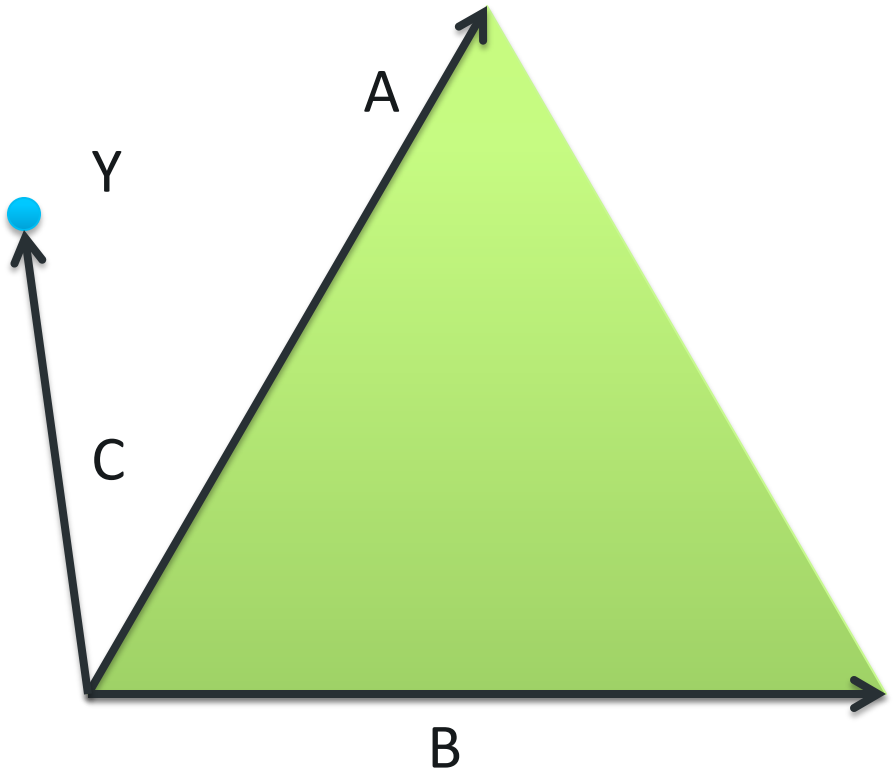
- Check if point on the plane is inside the triangle

- AxC should be same direction as AxB



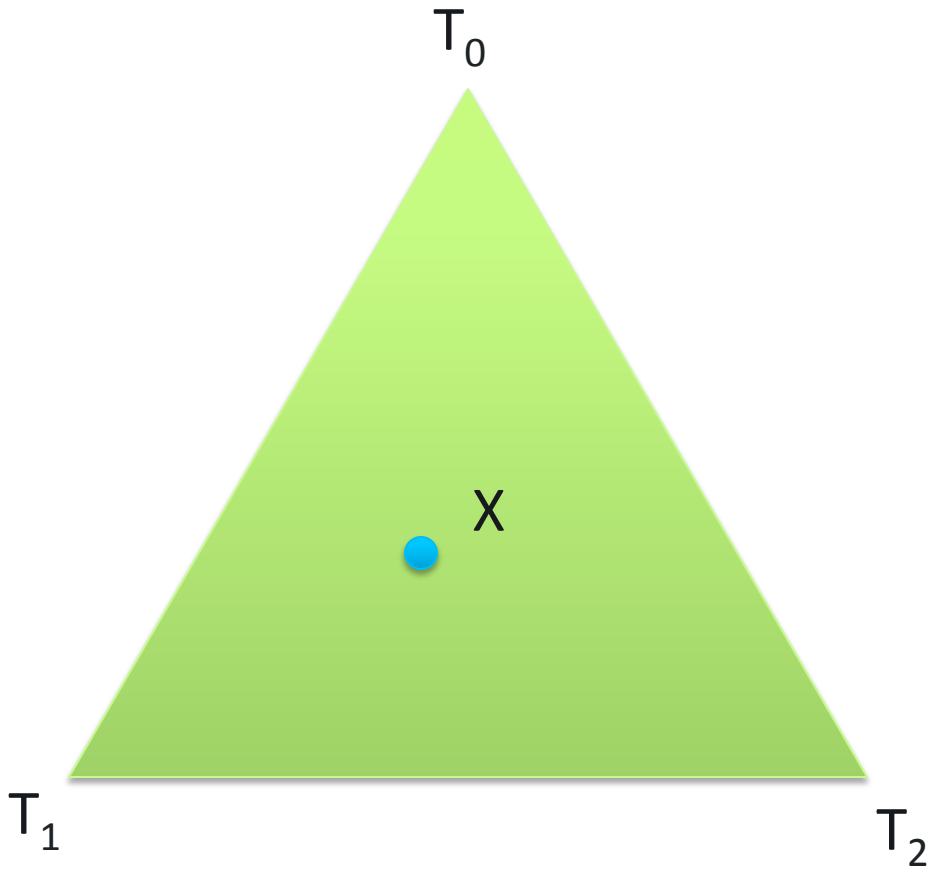
Triangles (Strategy I)

- Check if point on the plane is inside the triangle
- AxC should be same direction as AxB
  - And its not!



Triangles (Strategy II)

- Barycentric coordinates
- $X = \alpha(T_2 - T_1) + \beta(T_0 - T_1)$
- Solve for α and β
  - if $\alpha > 1$ or $\beta > 1$, fail
  - if $\alpha + \beta > 1$, fail



More complicated things?

- Can intersect with any mathematical representation of a 3D structure!
  - But sometimes the math gets kinda complicated
- Examples
  - Cones, cylinders
  - Boxes
  - Polygons (convex, concave)
  - Compound geometry

Optimization Approaches

- Bounding volumes
- Uniform grid
- Octrees
- BSP Trees

- “Embarrassingly parallel”

Up Next!

- What color is each pixel?
- What about reflective surfaces?
- What about refractive surfaces?

submission requirements

2D PROJECT

What do I turn in?

- An executable of your project
- Well-commented source code
- A *short* report giving instructions for how to play your game/use your application
  - What are the controls?
  - What are the features you implemented?
    - And how do they line up to project requirements?
  - What's an example use case?

- Deadline: 11:59pm, February 5th

What happens in class?

- Each group has **2 minutes** to give a lightning presentation of their work
  - The time limit is very strict
  - The next group must be lined up behind the current group to take over immediately
  - All presentations will run from one computer
  - Video encouraged, live demo unlikely
- (18 groups x 2 minute presentations) + time for chaos = 50 minutes
- Remaining time: in-class project critique

Presentation Logistics

- All slides, video, supporting materials must be submitted to Blackboard by 10am Tuesday, February 5th
- Presentation computer is a CCIS desktop
- Presentation order randomly determined, will be posted on Piazza