Ray tracing intro + camera

COMP575

Overview

- Homework
- So far...
- Ray tracing intro

So far...

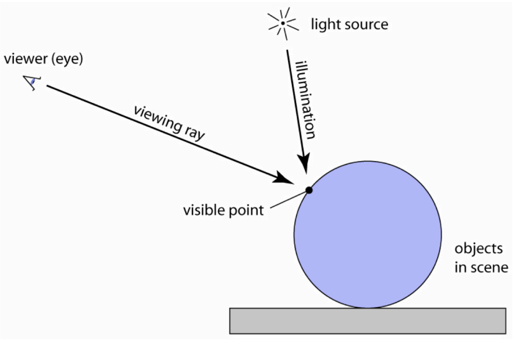
- Color representation
  - RGB floats internally
  - RGB bytes stored
- Mesh representation
  - Wavefront OBJ files
  - Triangle index + more
- Scene representation
  - Flat list
- Camera and sampling
  - Simple sampling and reconstruction

Ray tracing overview

- Visibility algorithm
  - Often used for rendering
- Input: objects, lights, camera
- Output: 2D image

Ray tracing overview

- Simplified overview
- For each pixel...
  - Generate ray
  - Check if ray hits any objects
  - If ray hits, generate a color
  - Store color for the pixel



Ray tracing overview

Ray generation

- Position camera in scene
- Create image plane
- Sample positions on image plane
- Create ray for each position

Ray tracing overview

Check ray hits

- Loop over all objects
- Test ray object intersection

- Divide objects into groups
- Test group intersection, then object intersection

Ray tracing overview

Get pixel color

- Record object hit data
- Use hit data and object color to get color

- Check if hit point is in shadow
- Reflect new ray if surface is mirror
- Other shader operations

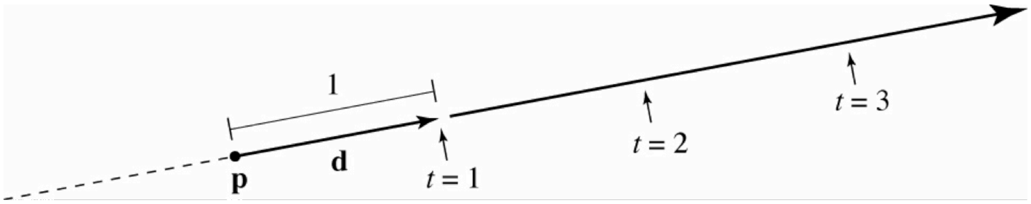
Ray tracing overview

Store pixel color

- Tone map color
- Apply gamma if desired
- Store image in memory/disk

Ray definition

- Half line from point
- Has origin and direction
- Helpful to reference distances on ray



Ray definition

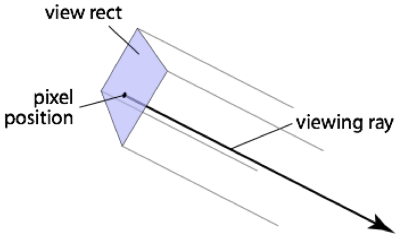
3D parametric line
r = **p** + *t***d**
r(*t*) = **e** + *t*(**s**-**e**)

- **r** is the set of points on the ray
- **p** is the origin (camera)
- **d** is ray direction *(s-e)*
- **e** and **s** are related to the camera (more later)
- *t* is the ray parameter ('length')

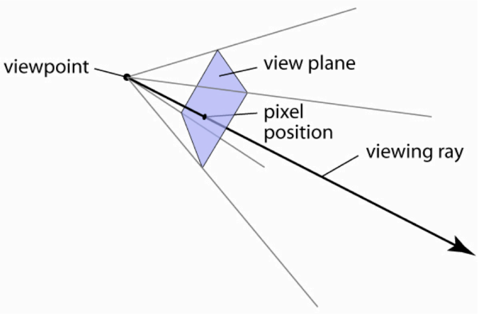
Ray generation

- From camera discussion
    - Orthographic, perspective
    - Image plane
    - View direction

Ray generation



Orthographic



Perspective

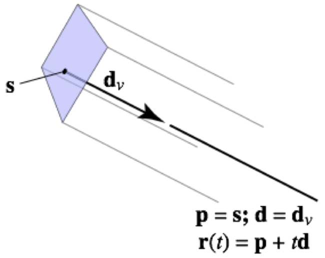
Ray generation

- Orthonormal basis
  - Represents camera frame in 3D
  - 3 orthonormal vectors: **u**, **v**, **w**
  - Camera **across**, **up**, and **look** vectors
- Using right hand rule, **look** may be backwards

Ray generation

Orthographic

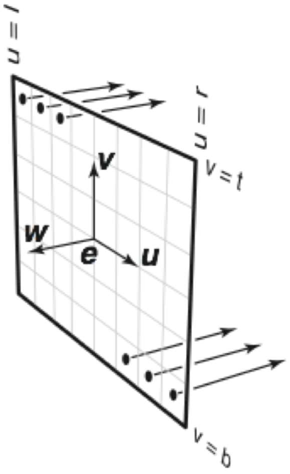
- Compute point **s** on image plane
- Create ray using **s** as origin


$\mathbf{s}$
$\mathbf{d}_v$
$\mathbf{p} = \mathbf{s};\ \mathbf{d} = \mathbf{d}_v$
$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$

Ray generation

Orthographic camera frame

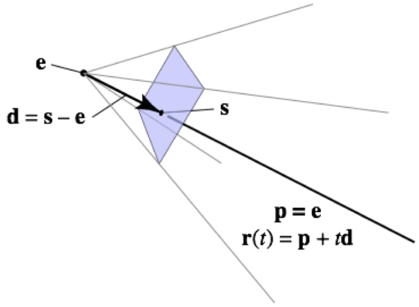
$$\mathbf{s} = \mathbf{e} + u\mathbf{u} + v\mathbf{v}$$
$$\mathbf{p} = \mathbf{s};\ \mathbf{d} = -\mathbf{w}$$
$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



Ray generation

Perspective

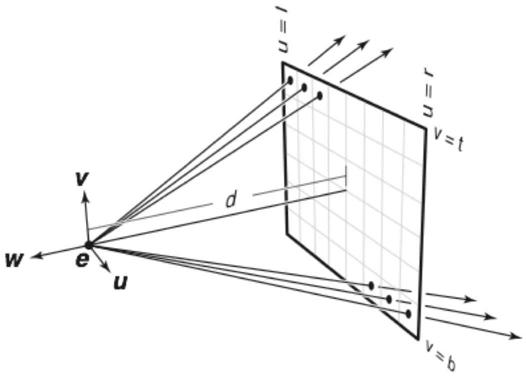
- Image plane is not at camera position
- Distance controls focal length/field of view
- **e** is origin, **s** controls direction



Ray generation

Perspective camera frame

$$\mathbf{s} = \mathbf{e} + u\mathbf{u} + v\mathbf{v} - d\mathbf{w}$$
$$\mathbf{p} = \mathbf{e};\ \mathbf{d} = \mathbf{s} - \mathbf{e}$$
$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



Ray generation

- Image to camera mapping (u, v)
    - l and r are the distance of the left and right edges
    - t and b are the distance of the top and bottom edges
    - (i, j) is the position in the image

$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

Ray generation

Image to camera mapping (u, v)

$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

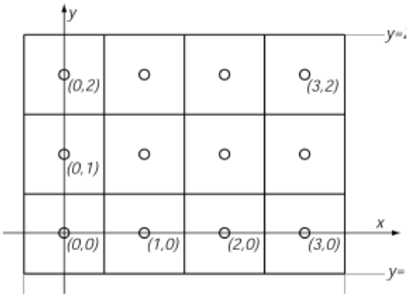
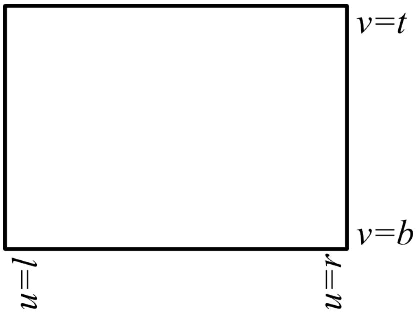


Image pixels



View plane positions

Object intersection

- Intersect ray with sphere
    - Use quadratic formula to solve equation

$$t = \frac{-\mathbf{d} \cdot (\mathbf{p} - \mathbf{c}) \pm \sqrt{(\mathbf{d} \cdot (\mathbf{p} - \mathbf{c}))^2 - (\mathbf{d} \cdot \mathbf{d})((\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2)}}{(\mathbf{d} \cdot \mathbf{d})}$$

- **d** is the ray direction
- **p** is the ray origin
- **c** is the sphere center
- *R* is the sphere radius
- *t* is the ray parameter of the hit

Code overview

- Basic C++ code will be posted
    - OBJ loader
    - Starting vector class

    - Starting color class
    - SDL frontend

Code overview

- Helpful classes
    - Vector
    - Ray
    - Hit point data
    - Camera
    - Ray generator
    - 2D image buffer
    - Shape: spheres...
    - Material: surface color...
    - Light: intensity...
    - Color
    - Shader
    - Scene data
    - Shape collection
    - Material collection
    - Light collection
    - Ray tracer: single ray
    - Ray renderer: ray loop
    - Shape intersection
    - Model loader
    - Option loader
    - Image save code

Code Overview

- Where to start?
    - Model camera
    - Generate rays
    - Print x, y, z as image
    - Must be able to load camera!
    - Use print outs or image dump to check

Code Overview

- Write small functions!
- Test each part as you go

Code Overview

- Load scenes
- Generate rays
- Sphere intersect
- Triangle intersect
- Color shading
- Shadows
- Reflections
- Image output