Introduction

UCSD CSE 168
Rendering
Tzu-Mao Li
Rendering turns scene descriptions to images

3D scene
(light sources, geometry, materials, volumes, cameras, etc)

image
Photorealistic rendering v.s. non-photorealistic rendering
The line is blurry
Physically-based rendering: simulate light transport

- Scattering
- Indirect light
- Caustic
- Specular reflection
- Indirect light
- Volumentric scattering
- Shadow
- Defocus blur
- Direct light

figure from Wenzel Jakob
Modeling & simulating the reality with code and mathematics

- you’ll build a slightly more sophisticated version of this in this course!

99 lines C++ code, no dependencies from Kevin Beason
Quiz time!

• name an application of physically-based rendering
Physically-based rendering is widely adopted

“The Adventures of Tintin”, 2011

“Assassin’s Creed Odyssey”, 2018

https://www.youtube.com/watch?v=teyAfaZ1FwM
https://www.youtube.com/watch?v=eSw6JeHge1Y
Most things are CGI in modern films

“The Irishman”, 2019

“How The Irishman’s Groundbreaking VFX Took Anti-Aging To the Next Level”, Netflix
Digital Emily project (2008)

- no neural networks!!!

https://vgl.ict.usc.edu/Research/DigitalEmily/
Remote collaboration/conferencing

Graphics in product visualization

- At 2014, around 60-75% of all IKEA’s product images (images showing only a single product) are CG. 35% of non-product images are fully CG.

[Images: These are all computer generated!!](https://petapixel.com/2014/08/28/flip-ikea-catalog-75-photography-see-cgi/)
KeyShot: Henrik Jensen’s startup
Discover new dimensions in design.

Get it as part of Creative Cloud All Apps plan.

Free trial  Buy now
Fast, high-resolution renderings in the cloud

Create high-quality photorealistic images, panoramas, solar studies and illumination simulations using Autodesk® Rendering.

SIGN UP FOR FREE  SIGN IN
Daylighting simulation for architectural design

from Nathaniel Jones

https://nljones.github.io/Accelerad/
Simulation for autonomous driving

Adding physics to computer vision

- recover geometry and material properties from images

from our work

“Inverse Path Tracing for Joint Material and Lighting Estimation”
Adding physics to computer vision

• recover geometry and material properties from images

from our work
“Inverse Path Tracing for Joint Material and Lighting Estimation”
Image synthesis for manufacturing

Nimier-David et al.

(Left: real, right: fabricated)

Papas et al.

caustic design

https://rgl.epfl.ch/publications/NimierDavidVicini2019Mitsuba2
https://cs.dartmouth.edu/~wjarosz/publications/papas13fabricating.html
Image synthesis for medical imaging

computed tomography

tissue imaging

photo courtesy of daveynin from wikipedia

illustration courtesy of Escobet-Montalban from U. of St. Andrews
Logistics

• 4 homeworks, 1 final project — 20% each (START EARLY! ASK QUESTIONS!)
  • late penalty: score * \left( 1 - \frac{\text{seconds passed since deadline midnight}}{86400} \right)
  • no negotiation unless you get a letter from the dean
• collaboration: can form a group with at most 2 people
• TA: Menghe Zhang (mez071@ucsd.edu)
  • My office hour: Friday 3pm-4pm @ CSE 4116
  • TA’s office hour: Wednesday 11am-12pm @ CSE B240A
• Online discussion: Piazza
Prerequisites

• C++ programming
• undergrad-level calculus
• have taken CSE 167 or similar courses
Final project: rendering competition

- goal: render an image (or a video) by extending your renderer
- grade by both artistic values and technical sophistication
- write a report and submit code (we will publish the top reports and images)
- start early!

inspired by Stanford CS 348b
Homework submission

- submit the rendered images on Canvas
- you can test your results on UCSD Online
Homework 1: Whitted-style ray tracing

• from CSE 167: a very basic ray tracer that we’ll describe soon

• previous offerings provide a OptiX startup code: less recommended but still available

• [new this year] **Bonus**: design your own scenes and render them!

• due in 2 weeks, **START NOW and ASK QUESTIONS!**
Introduction to ray tracing
Review: rasterization

Quiz: what should be in the two question marks?

```
foreach ?
  foreach ?
    if pixel sample and object overlap
      if z < z_buffer
        update pixel color
```
Review: rasterization

foreach object
foreach pixel sample
    if sample and object overlap
        if $z < z\_buffer$
            update pixel color
Ray tracing: swap the loops!

foreach object
foreach pixel sample
  if sample and object overlap
    if z < z_buffer
      update pixel color

rasterization

foreach pixel sample
foreach object
  if ray hit object
    if z < min_z
      update pixel color

ray tracing
Ray tracing: swap the loops!

- **Rasterization**
  ```
  foreach object
  foreach pixel sample
    if sample and object overlap
      if $z < z_{buffer}$
        update pixel color
  ```

- **Ray Tracing**
  ```
  foreach pixel sample
  foreach object
    if ray hit object
      if $z < min_z$
        update pixel color
  ```

Quiz: which one is faster?
Rasterization vs ray tracing

- rasterization
  - memory coherent access
  - simpler acceleration data structures

- ray tracing
  - allows arbitrary visibility query
  - enables more complex effects

from Gruen 2020
1080p, ~19M triangles
raster: 2.7 ms
raytrace: 8.6 ms (2.5 ms for animation)
We’ll focus on ray tracing

```plaintext
foreach pixel sample
    foreach object
        if ray hit object
            if z < min_z
                update pixel color
```
We’ll focus on ray tracing

foreach pixel sample
  foreach object
    if ray hit object
      if z < min_z
        update pixel color
Ray-sphere intersection

quiz: how do we know if a ray intersects with a sphere? if it hits, how do we know where?

\[ x = o + t \cdot d \]

\[ \| x - x_c \|^2 = r \]
Ray-triangle intersection

quiz: how do we know if a ray intersects with a triangle? if it hits, how do we know where?

\[ \mathbf{x} = \mathbf{o} + t \cdot \mathbf{d} \]

\[ \mathbf{x} = (1 - b_1 - b_2)\mathbf{P}_0 + b_1\mathbf{P}_1 + b_2\mathbf{P}_2 \]

https://en.wikipedia.org/wiki/Barycentric_coordinate_system
Ray-triangle intersection

\[ \mathbf{o} + t \cdot \mathbf{d} = (1 - b_1 - b_2)\mathbf{P}_0 + b_1\mathbf{P}_1 + b_2\mathbf{P}_2 \]

\[ o_x + t \cdot d_x = (1 - b_1 - b_2)P_{0x} + b_1P_{1x} + b_2P_{2x} \]
\[ o_y + t \cdot d_y = (1 - b_1 - b_2)P_{0y} + b_1P_{1y} + b_2P_{2y} \]
\[ o_z + t \cdot d_z = (1 - b_1 - b_2)P_{0z} + b_1P_{1z} + b_2P_{2z} \]

3 unknowns (t, b1, b2), 3 linear equations

"Moller-Trumbore algorithm"
We’ll focus on ray tracing.

foreach pixel sample
  foreach object
    if ray hit object
      if $z < \text{min}_z$
        update pixel color
Why do we see color?

photon/energy
Why do we see color?

color = light * reflectance
Handling visibility using ray tracing

color = light * reflectance if visible else 0
Shadow ray details

• we want to mark occluded if $\text{hit\_distance} < ||\text{light\_position} - \text{ray\_org}||$

• quiz: what if the shadow ray hits the red sphere itself?
Shadow ray details

• we want to mark occluded if $\text{hit\_distance} < ||\text{light\_position} - \text{ray\_org}||$

• quiz: what if the shadow ray hits the red sphere itself?

in practice, test

\[
\text{hit\_distance} > \text{eps} \quad \&\& \\
\text{hit\_distance} < (1 - \text{eps}) \times ||\text{light\_position} - \text{ray\_org}||
\]
Mirror reflections

• continue tracing the ray in mirror reflection direction if we hit a mirror surface

\[ r = d - (2d \cdot n) n \]

quiz 1: why?
quiz 2: how do we know the normal?

https://fabiensanglard.net/rayTracing_back_of_business_card/
Whitted-style ray tracer

foreach pixel sample
    ray = primary_ray(camera, sample)
    color = trace(ray)

def trace(ray):
    min_z = infty
    hit = None
    foreach object
        if ray hit object
            if z < min_z
                hit = ...
    if hits mirror:
        r = ...
        return trace()
    else:
        color = ...
        return color
Whitted-style ray tracer

```python
foreach pixel sample
    ray = primary_ray(camera, sample)
    color = trace(ray)

def trace(ray):
    min_z = infty
    hit = None
    foreach object
        if ray hit object
            if z < min_z
                hit = ...
    if hits mirror:
        r = ...
        return trace()
    else:
        color = ...
        return color

quiz: how do we parallelize this?
```
Business card ray tracers

from Paul Heckbert, 1984

https://fabiensanglard.net/rayTracing_back_of_business_card/

from Andrew Kensler somewhere between 2005-2009
Next: can we make this faster?

foreach pixel sample
  foreach object
    if ray hit object
      if $z < \text{min}_z$
        update pixel color

HW1 due in 2 weeks! START NOW!!!