Game Engines
CMPM 164, F2019

Prof. Angus Forbes (instructor)
angus@ucsc.edu

Montana Fowler (TA)
mocfowle@ucsc.edu

Website: creativecoding.soe.ucsc.edu/courses/cmpm164
Slack: https://ucsc_cmpm164.slack.com
Lab Sessions

Lab:
Ming Ong Computer Center, Windows Lab – Merrill 103

Tuesdays 11am-12noon
Wednesdays 3pm-4pm
Thursdays 1pm-2pm
(Montana’s office hours are also held during the Thursday lab)

Lab website / Direction:
https://its.ucsc.edu/computer-labs_descriptions/mingong.html
Homework 2

1: Create a simple scene using Unreal Engine (due 10/6)

2: Implement a classic Whitted recursive ray tracer (due 10/15)
Homework 2

1. Calculate Rays from Camera through each pixel of the Image Plane into 3D Scene

2. Calculate intersection point with closest 3D object

3. Depending on the object’s material:
   - Calculate color at point by casting a ray towards each light to determine diffuse and specular contributions
   - Cast a reflection ray, go to step 2
   - Cast a refraction ray, go to step 2

4. Combine information from recursively cast rays
Homework 2

PPM files:

P3
3 2
255
# The part above is the header
# "P3" means this is a RGB color image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value for each color
# The part below is image data: RGB triplets
255 0 0 0 255 0 0 0 255
255 255 0 255 255 255 0 0 0

You’ll see a lot of examples that use “P6” - an RGB color image in bytes
“If the PPM magic identifier is "P6" then the image data is stored in byte format, one byte per colour component (r,g,b)” - Paul Bourke

See https://www.scratchapixel.com/code.php?id=3 for an example C++ code to write out to a PPM file
RAY TRACING
(for one pixel up to first bounce)
1. Sphere equation: \((\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) = r^2\)
   Ray equation: \(\vec{r}(t) = \vec{o} + t\vec{d}\)
   Intersection:
   \[
   ((\vec{o} + t\vec{d} - \vec{c}) \cdot (\vec{o} + t\vec{d} - \vec{c}) = r^2
   \]
   \[
   t^2(\vec{d} \cdot \vec{d}) + 2(\vec{o} - \vec{c})t\vec{d} + (\vec{o} - \vec{c}) \cdot (\vec{o} - \vec{c}) - r^2 = 0
   \]

2. Illumination Equation (Blinn–Phong) with recursive Transmitted and Reflected Intensity:
   \[
   I = k_a I_a + I_i \left( k_d \left( \vec{L} \cdot \vec{N} \right) + k_s \left( \vec{V} \cdot \vec{R} \right)^n \right) + \underbrace{k_t I_t + k_r I_r}_{\text{recursion}}
   \]

3. Snell’s law:
   \[
   \frac{\sin \theta_1}{\sin \theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}
   \]
   \[
   n_{\text{air}} \sin \theta_i = n_{\text{glass}} \sin \theta_t
   \]
   refraction coefficients:
   \[
   n_{\text{air}} = 1, \quad n_{\text{glass}} = 1.5
   \]

4. Area Light Simulation:
   \[
   I_{\text{light}} \frac{\# \text{(visible shadow rays)}}{\# \text{(all shadow rays)}}
   \]
Pixels to World Space

Given an eye position and an image plane, how do you calculate the ray from the eye through each pixel?

Another way of saying this is: How do you move between raster space or pixel space to world space?

(in class explanation on chalkboard…)

Once you have the pixels in World Space, you can create a ray from the Camera through the pixel into the 3D scene.
Calculating Intersections

- Ray-Sphere intersection

- Ray-Triangle intersection

(in class explanation on chalkboard...)

See:


https://www.iquilezles.org/www/articles/intersectors/intersectors.htm
Calculating color at intersections

Once we’ve detected an intersection point, how do we figure out what color that point is?

**Phong shading**

A computationally inexpensive approximation of light that includes both diffuse and specular components
Ambient + Diffuse + Specular = Phong Reflection
Calculating color at intersections

Diffuse calculation:

Requires *Normal* vector and *Light* vector

\[
\text{DiffuseContribution} = \text{diffuseColor} \times \max( (\text{Normal} \cdot \text{Light}), 0.0 )
\]

The diffuse term is largest when the normal and the light are parallel. The diffuse term is 0 when the normal and the light are \(\geq\) perpendicular.
Calculating color at intersections

Specular calculation:

Requires *Reflect* vector and *View* vector. That is, it can change based on the position of the camera.

$$\text{SpecularContribution} = \text{specularColor} \times (\text{Reflect} \cdot \text{View})^\text{shininessFactor}$$

The specular term is large only when the viewer direction is aligned with the reflection direction. The highlights are sharper the greater the shininessFactor is.

*(Reflect vector is calculated using *Light* vector and *Normal* vector)*
Calculating color at intersections

Alternative Specular calculation:

Requires *Normal* vector and a *Half* vector that is in between the *View* and the *Light* vectors.

\[
\text{SpecularContribution} = \text{specularColor} \times (\text{Normal} \cdot \text{Half})^{\text{shininessFactor}}
\]

The specular term is large only when the viewer direction is aligned with the reflection direction. The highlights are sharper the greater the shininessFactor is.
Calculating color at intersections

Calculate diffuse and specular contribution for each light and add them together.

Check if light is not occluded by another object, otherwise move to next to light. If all lights are occluded, then the point is completely in shadow.
Next class

Reflection, Refraction, Recursion …