Ray Tracer

CSEP557 2019 Spring

Ray Tracer

- Given a ray “caster”, you have to implement:
  - Shading
  - Reflection and Refraction
  - Sphere Intersection
  - Triangle Intersection
    - Complex objects consist of a 3D mesh made up of triangles
  - Anti-Aliasing
- In this project, you will implement all of these!

Requirements

- Sphere intersection
- Triangle intersection
- Barycentric interpolation of Normals and UVs (for Trimesh)
- Blinn-Phong Specular-Reflection Shading Model
- Light Contribution
- Shadow attenuation
- Reflection
- Refraction
- Anti-Aliasing

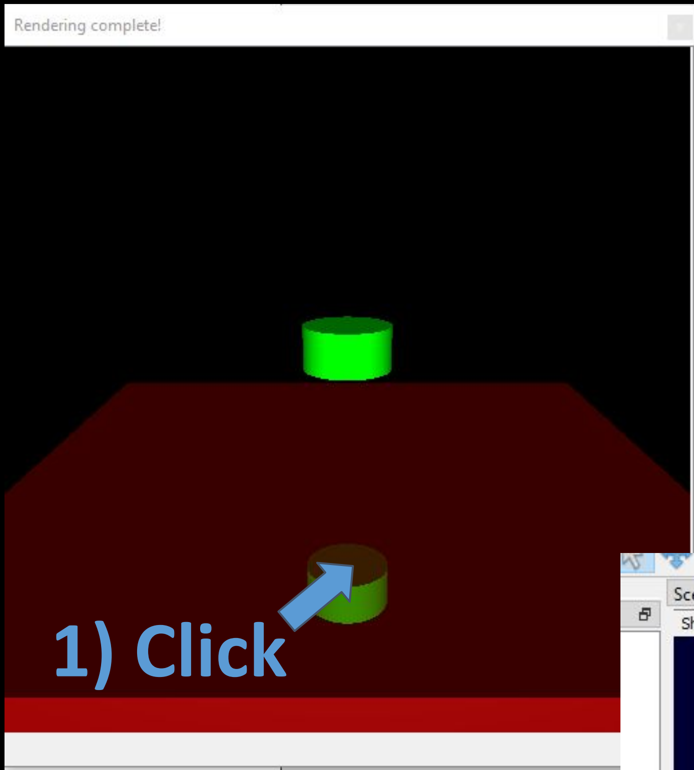
The Debugger Tool

- **USE THIS, IT WILL SAVE YOUR LIFE!**

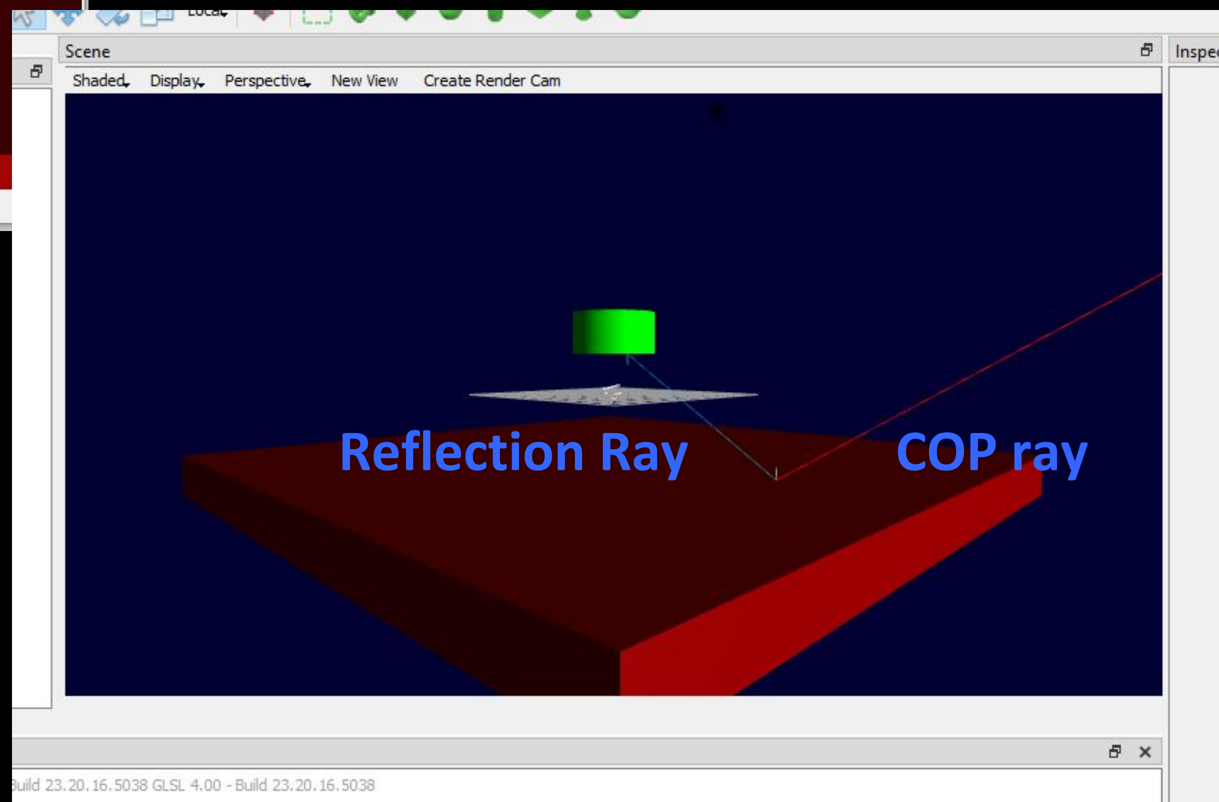
- Click a pixel in your rendered frame, and observe the scene view in the UI, it will show
  - Reflection Rays (if happened)
  - Refraction Rays (if happened)
  - Normal (at the intersection points)
  - Shadow/Light rays (intersection point to the light source)
  - COP ray (intersection point to the COP)

The Debugger Tool

- Demo



2) Observe scene view



Requirement: Sphere Intersection

- Fill in Sphere::IntersectLocal in scene\components\sphere.cpp
- The sphere is centered at the origin with radius **0.5**
- If the ray r intersects this sphere:
  1) Put the hit parameter in i.t
  2) Put the normal in i.normal
  3) Put the texture coordinates in i.uv (Not a Requirement; You will get 1 whistle if you implement this)
  4) Return true

Requirement: Triangle Intersection

- Fill in TriangleFace::IntersectLocal in Scene\components\triangleface.cpp

- See the triangle-intersection handout to get all equations you need.

Requirement: Triangle Intersection

- Access triangle vertices (class members)
  - glm::dvec3 a, b, c
- Interpolate normal and UV
  - Barycentric interpolation
- If the ray r intersects this sphere:
  1) Put the hit parameter in i.t
  2) Put the normal in i.normal
  3) Put the texture coordinates in i.uv
  4) Return true

Requirement: Blinn-Phong Specular-Reflection Model

- Refer to the lecture slides to get the formula

$$I_{\text{direct}} = k_e + \sum_j k_d I_{La,j} + A_j^{\text{shadow}} A_j^{\text{dist}} I_{L,j} B_j \left(k_d (\mathbf{N} \cdot \mathbf{L}_j) + k_s (\mathbf{N} \cdot \mathbf{H}_j)_+^{n_s} \right)$$

$$A_j^{\text{dist}} = \min\left(1, \frac{1}{a_j r_j^2 + b_j r_j + c_j}\right)$$

Requirement: Light Contributions

- To sum over the light sources, use a for loop to iterate all light sources as described in the code

- Access the light
  - Light* scene_light = trace_light->light
- Determine the type of light
  - Use dynamic casting

```
if (PointLight* point_light = dynamic_cast<PointLight*>(scene_light)) {
    // do something
} else if (DirectionalLight* directional_light = dynamic_cast<DirectionalLight*>(scene_light)) {
    // do something
}
```

Requirement: Light Contributions

- For Point Light: Get Light Position
  - TraceLight::GetTransformPos()

- For Directional Light: Get Light Direction
  - TraceLight::GetTransformDirection

Requirement: Light Contributions

- For Point Light:
  - Consider Distance Attenuation
  - First, check if the light type is AttenuatingLight

```
if (AttenuatingLight* attenuating_light = dynamic_cast<AttenuatingLight*>(scene_light))
```

  - Second, get a, b, and c

```
a = attenuating_light->AttenA.Get();
b = attenuating_light->AttenB.Get();
c = attenuating_light->AttenC.Get();
```

Requirement: Shadow Attenuation

- Rather than simply setting the attenuation to zero if an object blocks the light, accumulate the product of k_t's for objects which block the light

- See lecture slides to get more details

Requirement: Reflection

- Modify RayTracer::TraceRay in raytracer.cpp to implement recursive ray tracing
- Get reflection direction

$$\mathbf{R} = 2(\mathbf{V} \cdot \mathbf{N})\mathbf{N} - \mathbf{V}$$

- Consider UI setting in your implementation
    -

```
if (settings.reflections)
{
    // Put your reflection codes here
}
```

Requirement: Refraction

- Apply Snell's law

- Get refraction direction

$$\eta = \frac{\eta_i}{\eta_t}$$

$$\cos\theta_i = \mathbf{N} \cdot \mathbf{V}$$

$$\cos\theta_t = \sqrt{1 - \eta^2(1 - \cos^2\theta_i)}$$

$$\mathbf{T} = (\eta\cos\theta_i - \cos\theta_t)\mathbf{N} - \eta\mathbf{V}$$

Note that Total Internal Reflection (TIR) occurs when the square root term above is negative.

Requirement: Refraction

- Watch out for total internal refraction
- Consider the case when the ray is exiting a material into air (think about the direction of the normal)
- Consider UI setting in your implementation
  -

```
if (settings.refractions)
{
    // Put your refraction codes here
}
```

Direct + Indirect Illumination

- Direct Illumination + Reflection + Refraction

$$I_{\text{total}} = I_{\text{direct}} + k_r I_{\text{reflectedRay}} + k_t I_{\text{transmittedRay}}$$

Use Ks (specular coefficients) in our case

Requirement: Anti-Aliasing

- Gets rid of jaggies
- Implement using oversampling.
  - Equally divide each pixel, trace the ray, and average the results

Requirement: Anti-Aliasing

- Fill code in Raytracer::ComputePixel(...)

```
switch (settings.samplecount_mode) {
    case Camera::TRACESAMPLING_CONSTANT:
        // Put Your anti-alasing code here
        color = SampleCamera(x_corner, y_corner, settings.pixel_size_x, settings.pixel_size_y, debug_camera);
        break;
    default:
        break;
}
```

- Get the number of samples you need to shoot in each pixel
  - Settings.constant_samples_per_pixel
- Call SampleCamera() when shooting a ray at different positions of a pixel.

Data Structure: Ray

- Direction: r.direction
- Position: r.position

- r.at(t) – r.position + (t * r.direction)
  - Returns the end position of the ray r after going a distance of t from its start position

Take Care of Normals

- Interpolated Normal
  - Used for shading

```
glm::vec3 N = i.normal;
```

- True/Geometric Normal
  - Used for **everything except for shading**, like entering/leaving a object, computing reflection/refraction rays

```
glm::vec3 GeometricN = i.GetTrueNormal();
```

Take Care of Normals

- Flip both (interpolated and true) normals if you are on the inside of an object, for any shading, reflection, or refraction.
  - As we said before, use True Normal to determine if you’re a on the inside/outside the object (i.e. use the sign of glm::dot(-r.direction, GeometricN))

Test Your Implementation

- Start from simpler case: assets/trace/simple
  - Sphere: sphere_xxx.yaml
  - Trimesh: box_xxx.yaml, cube_xxx.yaml
    - Texture: texture_reflection.yaml
  - Distance attenuation: box_dist_atten.yaml
  - Opaque shadow: box_cyl_opaque_shadow.yaml
  - Transparent shadow:
    - box_cyl_trans_shadow.yaml, cube_transparent.yaml
  - Reflection
    - box_cyl_reflect.yaml, texture_reflection.yaml
  - Refraction
    - box_cyl_trans_shadow.yaml, cube_transparent.yaml
    - cylinder_refract.yaml, sphere_refract.yaml

Test Your Implementation

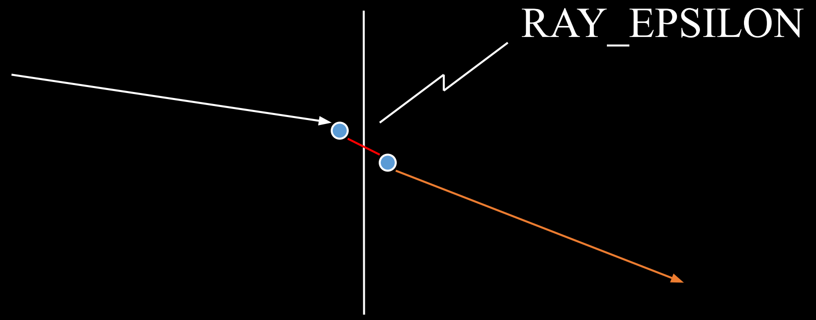
- Then test more complicated case in
  - **assets/trace/trimeshes**
  - **assets/trace/more**

  - In particular, try
    - trimeshes/**revolution_texture.yaml** to see your trimesh texture
    - more/**lecture.yaml** to see the effect of direct illumination + reflection + refraction
    - trimeshes/**dragon.yaml** to test your anti-aliasing

Tips and Tricks

- Don't write too much code without testing!
  - Lots of dependencies, think carefully before writing any codes

- Use RAY_EPSILON (which is defined as 0.00001) to account for computer precision error when checking for intersections



Memory Leaks

- A memory leak can (and probably will) ruin your night hours before your artifact is due
- To test, try to ray trace a complex model (the dragon) with depth 10, anti-aliasing, HUGE Image
- Cause: not calling free after allocating memory
  - Object constructors, vector (array) creation
- Solution: free stuff!
  - Call the “delete [object]” on ANYTHING you create that is temporary
    - i.e. 3 byte temporary vectors in the rayTrace function
- It is HIGHLY RECOMMENDED you have no memory leaks

Comparison Tool

- We will be using this tool (link on the course webpage) to evaluate your solution versus the sample. So you should check too !!

- See the class announcement letter and course page for the details
  - Will announce this soon ...

That’s all. Good luck!