HELP SESSION

TRACE

REQUIREMENTS

- You will implement essential components of a ray racer, including
  - Sphere Intersection
  - Triangle Intersection
    - Barycentric interpolation of Normals and UVs (for Trimesh)

REQUIREMENTS

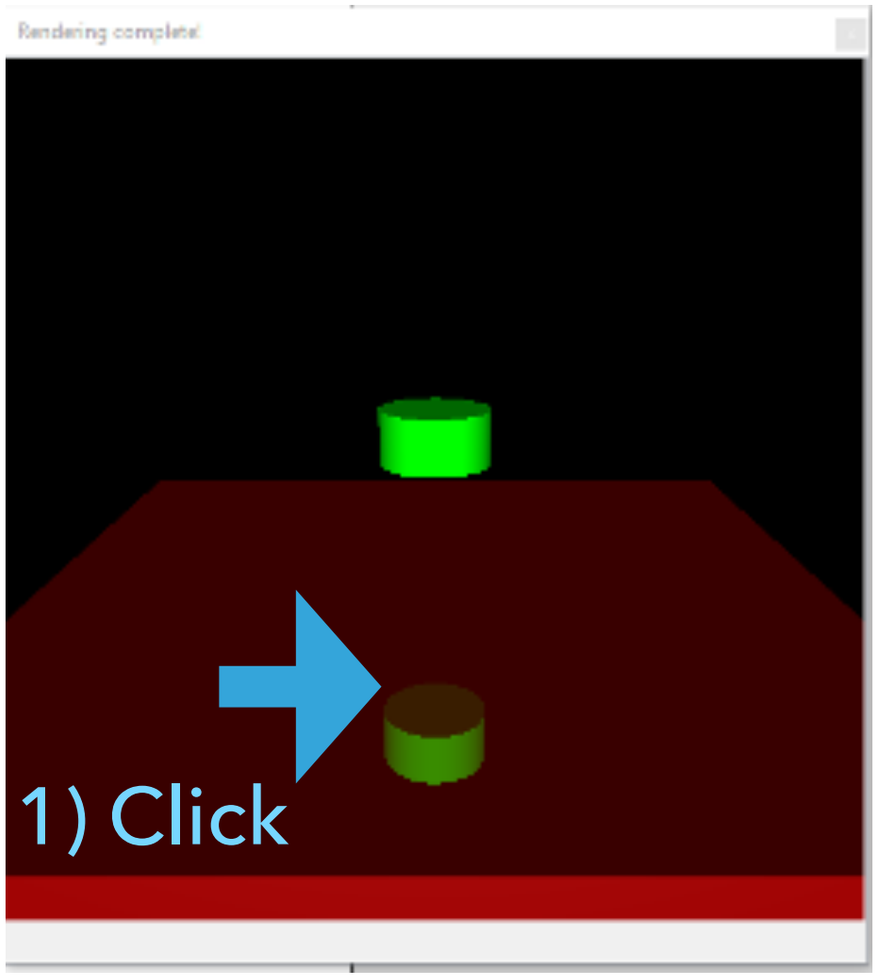
- You will implement essential components of a ray racer, including
  - Blinn-Phong Specular-Reflection Shading Model
    - Shading
    - Light Contributions
    - Shadow Attenuation
    - Reflection
    - Refraction
  - Anti-Aliasing

DEBUGGER TOOLS

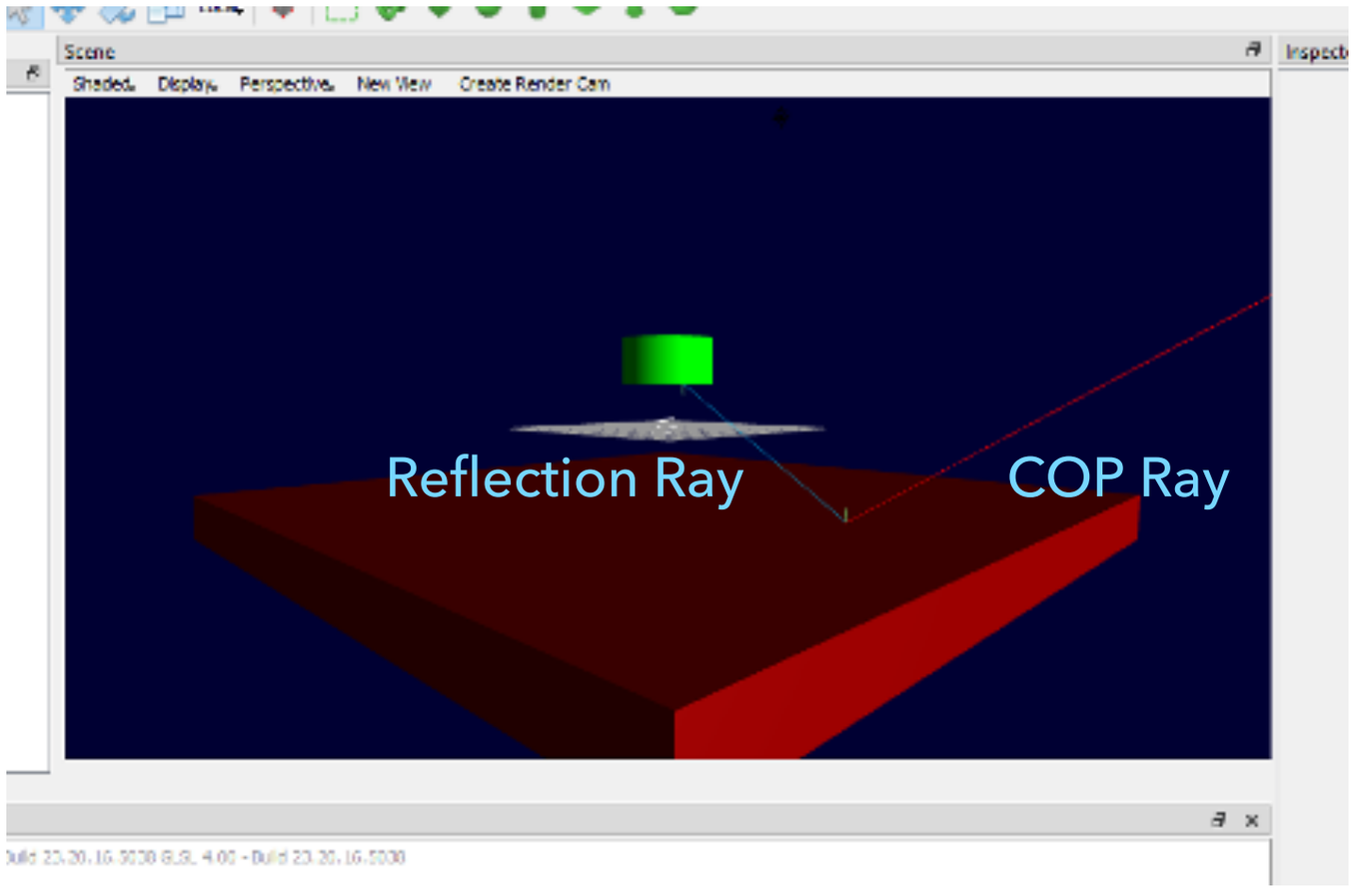
- **USE THIS, IT WILL SAVE YOUR LIFE!**

- 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)

DEBUGGER TOOLS – DEMO



2) Observe Scene View



TRACE

SPHERE INTERSECTION

SPHERE INTERSECTION

- Fill codes in Sphere::IntersectLocal()
- 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

TRIANGLE INTERSECTION

TRIANGLE INTERSECTION

- Fill in TriangleFace::IntersectLocal
  - See the triangle-intersection handout to get all equations you need.
- Access triangle vertices (class members)
  - glm::dvec3 a, b, c
- Interpolate normal and UV
  - Barycentric interpolation
- If the ray r intersects this triangle:
  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

BLINN-PHONG SHADING

BLINN-PHONG SPECULAR-REFLECTION MODEL

- Formula

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

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

LIGHT CONTRIBUTIONS (1/3)

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

LIGHT CONTRIBUTIONS (2/3)

- For Point Light: Get Light Position
  - TraceLight::GetTransformPos()
- For Directional Light: Get Light Direction
  - TraceLight::GetTransformDirection

LIGHT CONTRIBUTIONS (3/3)

- 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 coefficients a, b, and c

```
attenuating_light->AttenA.Get();
attenuating_light->AttenB.Get();
attenuating_light->AttenC.Get();
```

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

REFLECTION

- Fill codes in 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
}
```

REFRACTION (1/2)

- 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.

REFRACTION (2/2)

- Be aware of Total Internal Refraction
- Consider the case when the ray is exiting a material into air
- Consider UI setting in your implementation

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

DIRECT + INDIRECT ILLUMINATION

- Formula

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

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

TRACE

ANTI-ALIASING

ANTI-ALIASING

- Gets rid of jaggies
- Implement using oversampling.
  - Equally divide each pixel, trace the ray, and average the results
- Fill code in Raytracer::ComputePixel
- Enable anti-aliasing
  - Goto property of RenderCamera

TRACE

TESTING & TRICKS

SIMPLE TEST SCENES

- 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

SIMPLE TEST SCENES

- More scenes in simpler case: assets/trace/simple
  - 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

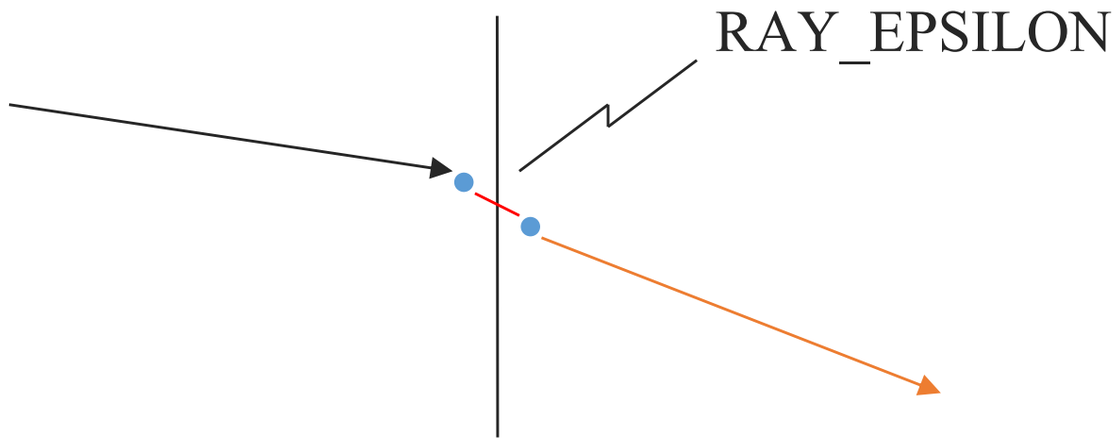
MORE COMPLICATED TEST SCENES

- 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

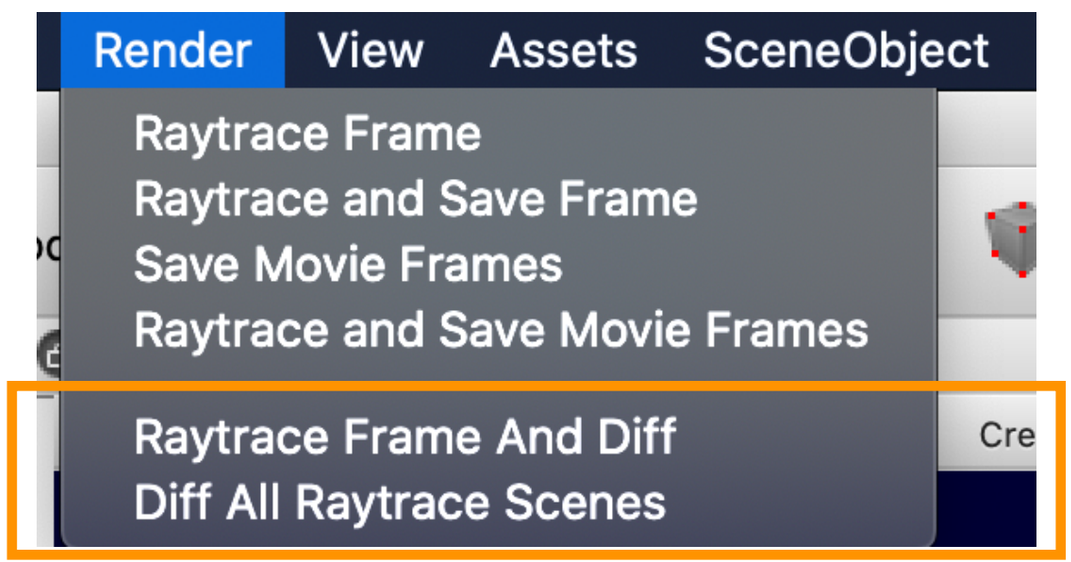
- Don’t write too much code without testing!
  - Lots of dependencies, think carefully before writing any codes
- Use RAY_EPSILON (which is defined as 0.00001) to account for computer precision error when checking for intersections



AUTO DIFF TOOL

- Two ways
  - **Diff Selected Scenes**: Output errors on the console output and automatically display the visual diff image (mark unmatched pixels as red)
  - **Diff All Scenes**: Render all test scenes sequentially and output all of the errors on the console output.

- The tool will also store a diff image, named as [scene_name]_[render_depth]_diff.png, in the same folder as the scene .yaml file.



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
- It is HIGHLY RECOMMENDED you have no memory leaks

THE END

GOOD LUCK