Exercises - Programming

Simple ray-tracer

The handout

- RayTracer
  - addSceneObject(SceneObject*)
    - Add a object to our scene
    - Our ray-tracer only support spheres and the background cube map
  - render()
    - Shoot rays into the scene
    - Save the result of each ray to the frame buffer
  - save(string)
    - Saves the latest render operation to a tga image file

See `ex16_raytracer.cpp` for examples on how the handout is used. You are encouraged to play around with it (try to add more objects).

Task 1. Shooting rays.

- Implement shooting one ray for each pixel
- Start with the `render()` method in `RayTracer.cpp`
- Assume a pinhole camera where the "screen" is "1" away from camera positio. We can then write the position of a pixel relative to the camera as:

```
//[i, j] = pixel index
x = (i+0.5)*(screen.right-screen.left)/width + screen.left;
y = (j+0.5)*(screen.top-screen.bottom)/height + screen.bottom;
z = -1.0f;
```

- Automatically becomes the direction of our ray.

Exercises - Programming

Simple ray-tracer

Task 2. Intersection.

- Implement ray-sphere intersection test
- Start with `intersect(Ray)` in `Sphere.hpp` and compute t for intersection point $q(t)$ (see description under)

Ray-sphere intersection

- A sphere is characterized by a radius r, and a center c
- All points $[x, y, z]$ are on the sphere if they satisfy:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r$$

- Simply find the point $[x, y, z]$ that satisfies the above equation closest to the camera.
- Lets formulate a function for a point along the direction of our ray

$$q(t) = q_o + tq_d$$
$$q_o : \text{Origin of the ray}$$
$$q_d : \text{Direction of the ray}$$

- Insert our function into the sphere formula.

$$(q(t)_x - c_x)^2 + (q(t)_y - c_y)^2 + (q(t)_z - c_z)^2 = r$$

- Quadratic formula where t is the only unknown.

- Solve using the quadratic formula to find intersections (roots of the polynomial)

$$
\begin{aligned}
t &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \\
a &= q_d \cdot q_d \\
b &= 2(q_d \cdot (q_o - c)) \\
c &= (q_o - c) \cdot (q_o - c) - r^2
\end{aligned}
$$

- If $b^2 - 4ac$ is negative, we get an imaginary root; no intersection
- Otherwise, we hit the sphere, and the closest intersection is the smallest positive of the two roots
- q(t) is then the intersection point
- Three cases (root t_0 and t_1):
  - One positive, one negative ($t_0 t_1 < 0$): Return $max(t_0, t_1)$
  - Two positive roots: Return $min(t_0, t_1)$
  - Two negative roots: Return -1 (No intersection)

Task 3. Reflection

- Implement the reflection effect
- Start with the `ReflectiveEffect` class in `SceneObjectEffect.hpp` and implement the `rayTrace(Ray, float, vec3, RayTracerState)` method
- Spawn a new reflection ray from the intersection point $q(t)$
- You can use `glm::reflect(I,N)` to create the reflection vector

Exercises - Programming

Simple ray-tracer

Exercises - Programming

Simple ray-tracer

Extras/Possible extensions

- Multisampling
  - Spawn more rays for each pixel and combine the results
- Shadows
  - Every time a ray intersects with something, shoot a new ray from that point towards each light source, if the ray intersects with something, the original intersection point is in shade.
- More objects
  - Plane
  - Box
  - Triangle
  - Triangle mesh
- More effects
  - Phong
  - Fresnel
    - Spawn two new rays; reflection and refraction
    - Combine them using Schlicks approximation from the shader exercise