Kiril Shiyan

FN273222008  
  
Computer Graphics

Course Work  
  
Topic:  
Ray tracer

Problem definition:

Computer graphics is a field dedicated to the creation, manipulation, and representation of visual content using computational techniques. At its core, it involves transforming objects from a conceptual or mathematical space—often referred to as the "world space"—into a visual form, typically a two-dimensional image that can be displayed on a physical device such as a monitor or screen. These objects may exist in two-dimensional, three-dimensional, or even higher-dimensional (ND) mathematical representations.

Computer graphics relies on a variety of rendering techniques. One of the most widely used and powerful among these is **ray tracing**.

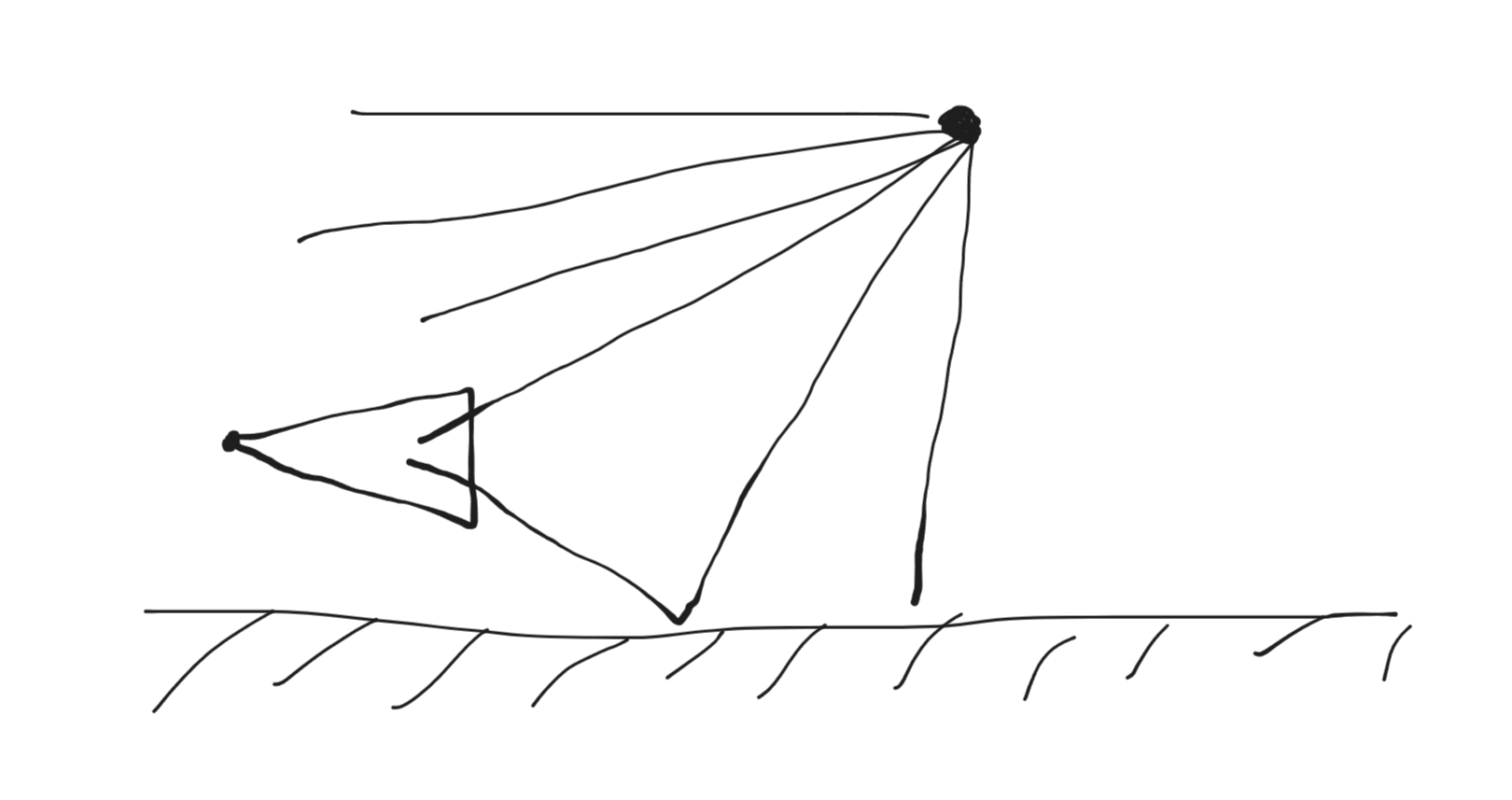
Ray tracing simulates the way light interacts with objects in a virtual environment to generate highly realistic images. It works by tracing the path of light rays as they travel through a scene, accounting for phenomena such as reflection, refraction, and shadows. This method can produce visually stunning results with photorealistic lighting and shading, making it a popular choice in fields such as visual effects, video games, architectural visualization, and scientific simulations.

While ray tracing is computationally intensive, advancements in hardware and optimization algorithms have made it increasingly accessible for real-time applications. In non-real-time applications, ray tracing can be made even more powerful and detailed.

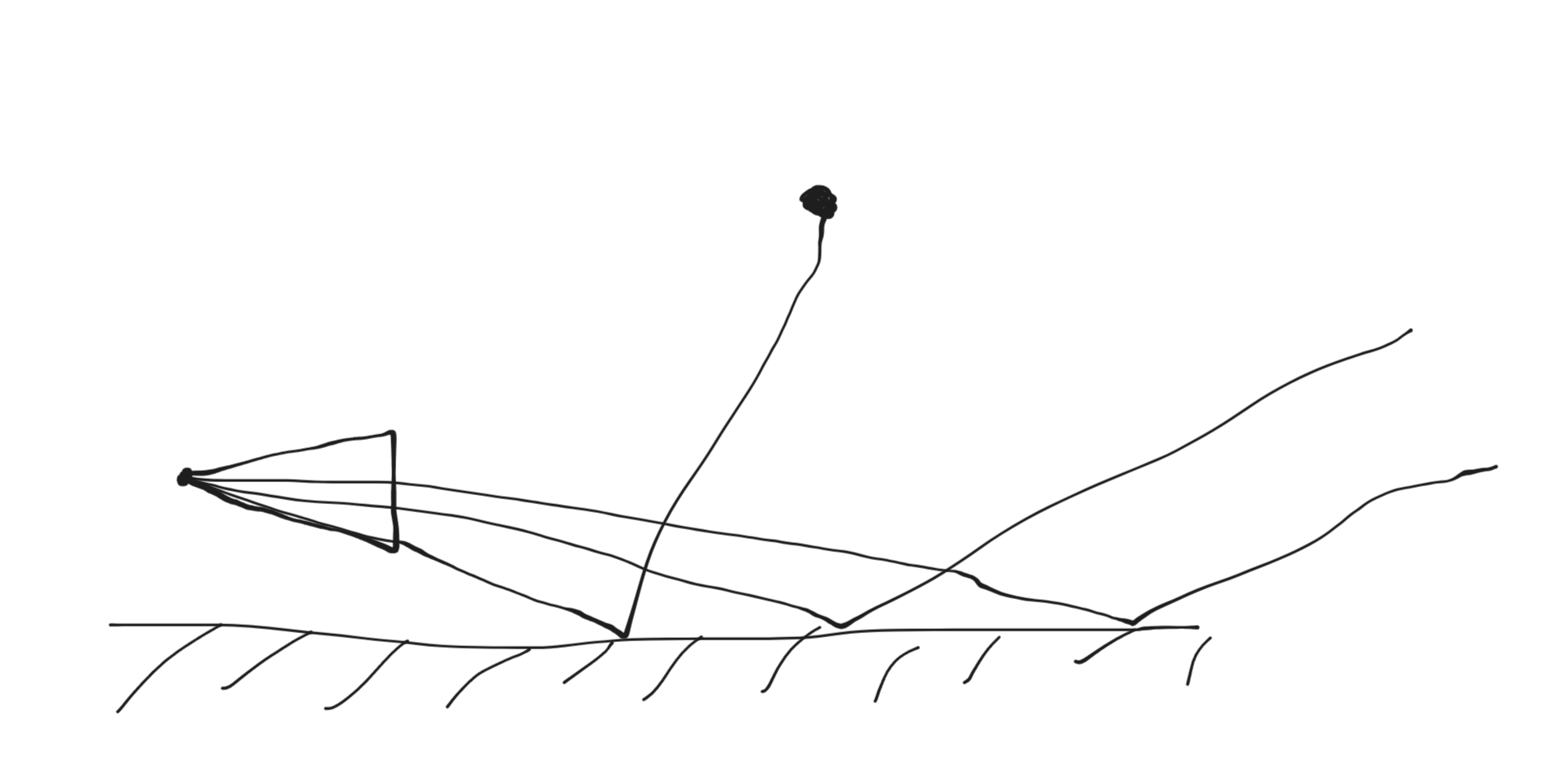
Problem Solution

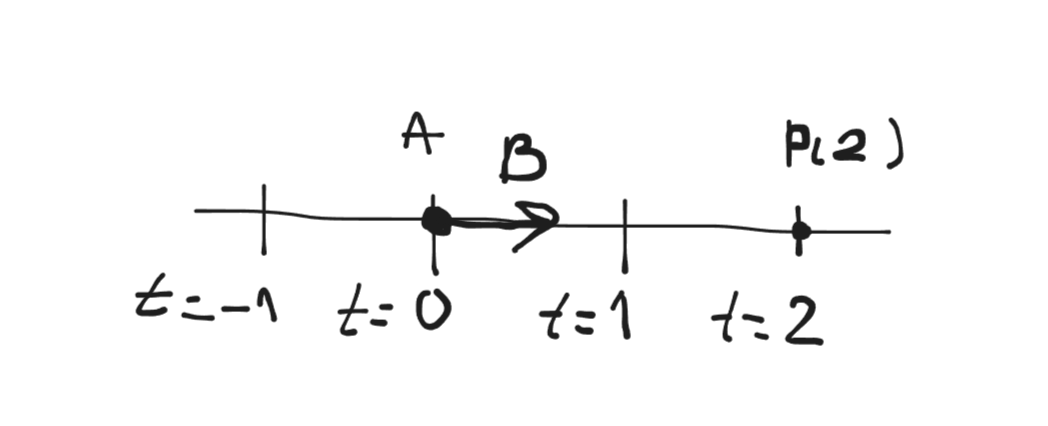
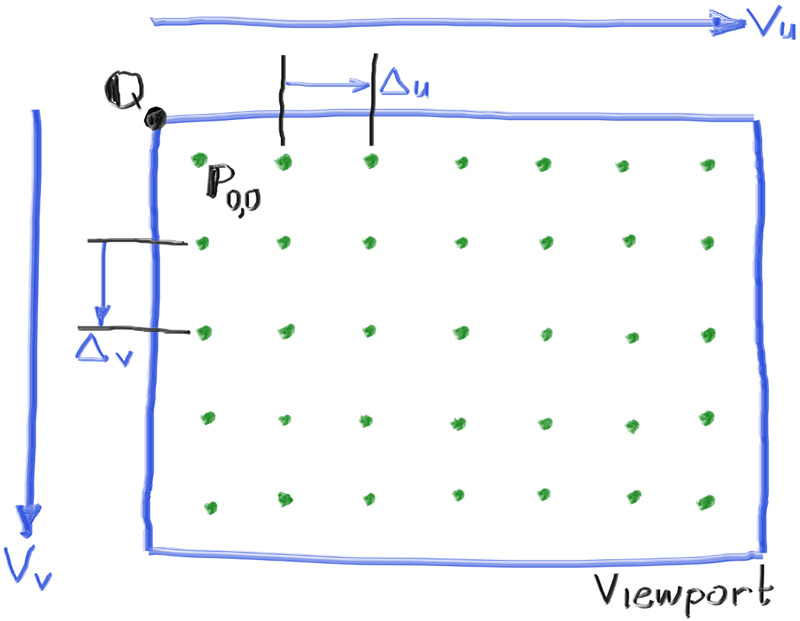
In core of all ray tracing techniques, lies a segment that has start but has no end, also known as ray. As you may know, in real life, human eye does not emit any light, only absorbs it. However, implementing it in an algorithm can be cumbersome and inefficient, since we cannot know ahead of time what rays will reach our eye. That’s why we revert process, so that our eye sends rays and then processes all collisions of those rays with objects in world space.

Rays emitted by source:



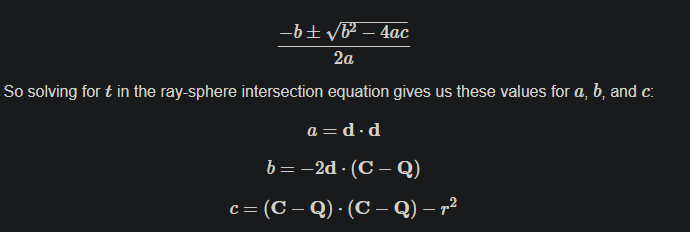
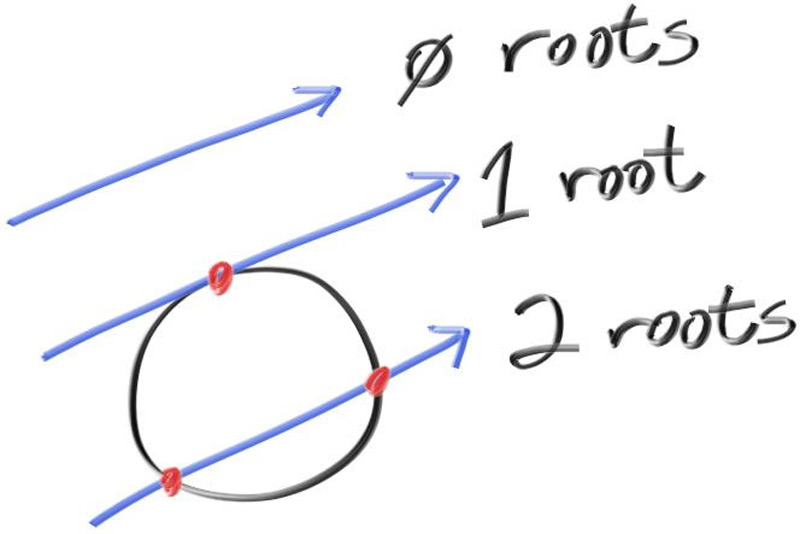
Rays emitted by camera:

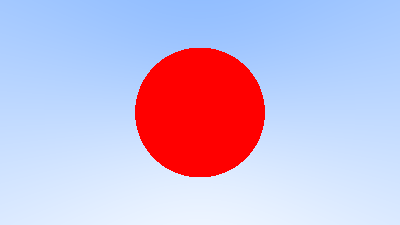


Mathematically, ray is just an equation of form **P**(t) = **A** + t**B**, where P is intersection point, A is source point and B is direction vector. By plugging different values of t, we can make our ray move to different points in space.  
  
Now that we know what ray is, we need to figure out exactly how to send them. For this, we need a new object. A mathematical representation of an eye. A camera. Camera is defined as follows. It is a 2D plane that starts at Q, has size of Vu by Vv and a step of Δu by Δv. Note that those are what’s called viewport, meaning they are using world space coordinate system. Now we can send through each square given amount of rays (the more rays are sent the better quality resulting image will be, but it will take more time) and see what color they add up to.  


Simple ray traced image



Now that we have simple camera and a ray, we can send, let’s add some object to world space. This project will concern itself only with sphere, as it is the simplest object for ray tracing.   
First, we need to find hit point. Sure, technically we can just iterate over all possible values of t, but it is computationally infeasable. We need better solution. By applying some math that can be found in sources, we can get next equation.  
  
Where C is center of sphere, Q is origin of ray, r is radius of sphere, and d is direction of ray. By solving this equation we can get following cases.  
  
Zero routes - no intersection for any t. One root - ray is tangent to sphere with one intersection point. Two root - ray passes through sphere. Knowing that we can define sphere as center, radius and color, and see what rays will impact it.

Simple sphere  


Sources

Implementation of theory described previously can be found on my GitHub <https://github.com/ThNeutral/cpp-ray-tracing>, which also has expanded functionality like multiple materials, shading, anti-aliasing and more. Implementation was heavily based on brilliant book called “Ray Tracing in One Weekend” (<https://raytracing.github.io/books/RayTracingInOneWeekend.html>) made by Peter Shirley, Trevor David Black, Steve Hollasch. I highly recommend to read book and try to implement ray tracer yourself. Book also has sequels called “Ray Tracing in Next Weekend” and “Ray Tracing The Rest Of Your Life” (<https://raytracing.github.io/>).