Objetos 3D

```
pub fn cast_ray(ray_origin: &Vec3, ray_direction: &Vec3, objects: &[Object]) -> u32 {
    for object in objects {
        if object.ray_intersect(ray_origin, ray_direction) {
            return 0xFFFFFF;
        }
    }
    0x000000
}
```

Nuestro primer objetivo es crear una función que pueda revisar si un rayo, que solo es una representación matemática de un vector, está interceptando con un objeto. En el ejemplo de arriba, si el rayo está interceptando con el objeto regresa el color blanco y si no intercepta con algo regresa el color negro. Es decir, que lo único que tenemos que hacer para poder colocar objetos en nuestro mundo es implementar esta función ray_intersect para cada tipo de objeto que queramos soportar.

```
use nalgebra_glm::Vec3;

pub trait RayIntersect {
    fn ray_intersect(&self, ray_origin: &Vec3, ray_direction: &Vec3) -> bool;
}
```

Como primer ejercicio, lo clásico es implementar un objeto esfera. ¿Cuáles son los valores que necesitamos para poder describir una esfera? La esfera más simple puede ser representada por un punto que marque el centro de la esfera y por un radio.

```
pub struct Sphere {
    pub center: Vec3,
    pub radius: f32,
}
```

Luego debemos simplemente implementar la funcion ray_intersect para nuestra esfera. Apoyense en sus conocimientos de algebra lineal para construir una función que determine si un rayo intercepta con una esfera. Este es un ejemplo que utiliza un discriminante para calcular el intercepto, pero existen otras maneras.

```
impl RayIntersect for Sphere {
  fn ray_intersect(&self, ray_origin: &Vec3, ray_direction: &Vec3) -> bool {
    // Vector from the ray origin to the center of the sphere
    let oc = ray_origin - self.center;
```

```
    // Coefficients for the quadratic equation
    // a = dot(ray_direction, ray_direction)
    // This is the dot product of the ray direction with itself, representing the squared length of the direction vector.
    let a = dot(ray_direction, ray_direction);

    // b = 2.0 * dot(oc, ray_direction)
    // This is twice the dot product of the vector oc and the ray direction.
    // It represents the projection of oc onto the ray direction, scaled by 2.
    let b = 2.0 * dot(&oc, ray_direction);

    // c = dot(oc, oc) - radius^2
    // This is the dot product of oc with itself minus the squared radius of the sphere.
    // It represents the squared distance from the ray origin to the sphere center minus the squared radius.
    let c = dot(&oc, &oc) - self.radius * self.radius;

    // Discriminant of the quadratic equation
    // discriminant = b^2 - 4ac
    // The discriminant determines the number of solutions to the quadratic equation.
    // If the discriminant is greater than zero, the ray intersects the sphere at two points.
    // If the discriminant is zero, the ray is tangent to the sphere and intersects at one point.
    // If the discriminant is less than zero, the ray does not intersect the sphere.
    let discriminant = b * b - 4.0 * a * c;

    // The ray intersects the sphere if the discriminant is greater than zero
    discriminant > 0.0
  }
}
```