





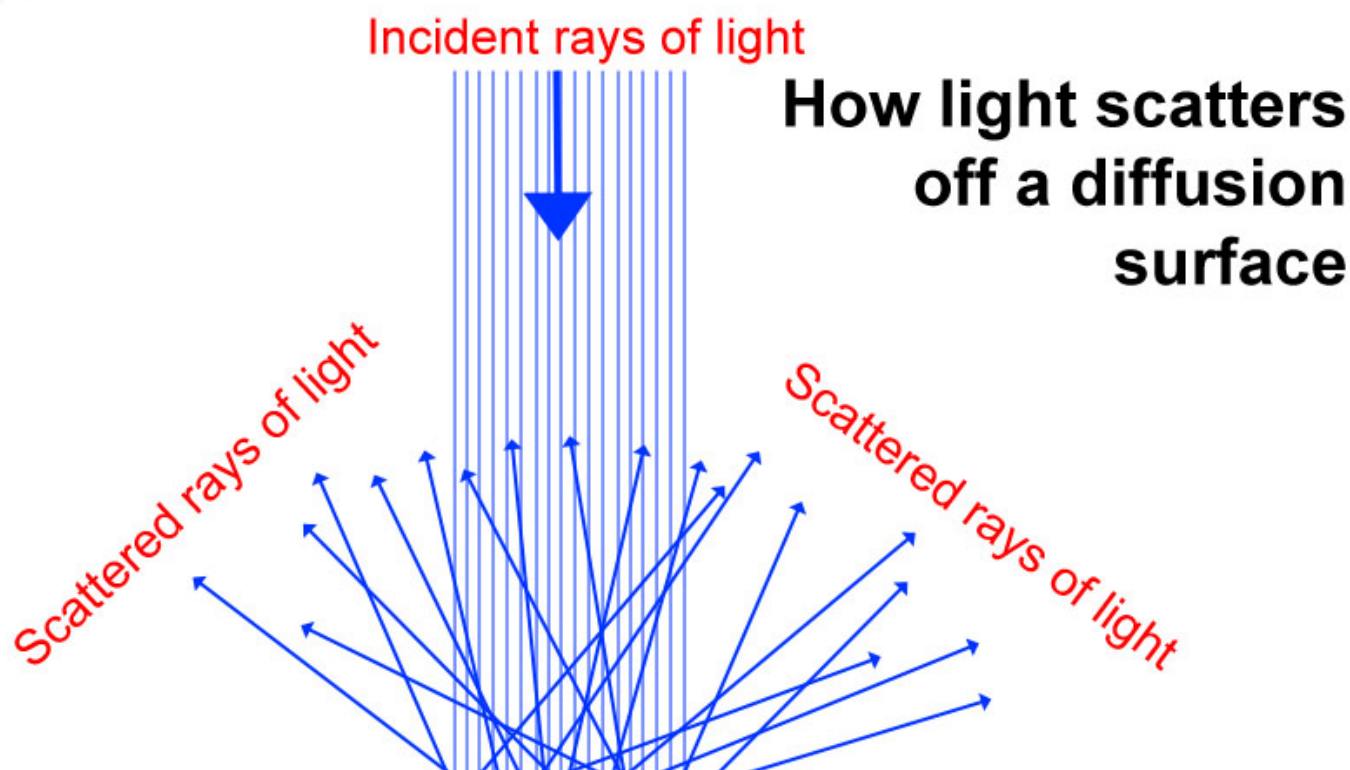
Materiales

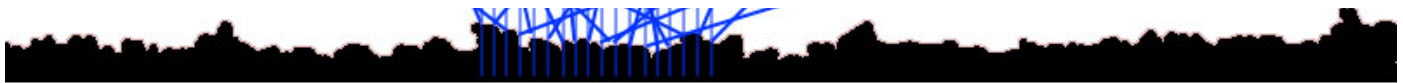
En el mundo real, tenemos esferas que pueden ser de muchos distintos colores y materiales. Esto quiere decir que reflejan la luz de una manera distinta.

Basketball	Globe	Marble	Orange
			

Cada uno de estos materiales refleja la luz de múltiples maneras a la vez, pero nos vamos a enfocar primero en el canal de luz que influencia más el color de la esfera primero. A esta le llamamos la luz difusa y es la consecuencia de las irregularidades de la superficie.

Light





As light hits a diffusion surface the light scatters unevenly in a random fashion reflecting off the chaotic structure of the micro-surface.

Creen una estructura para almacenar el material y modifiquen su función `ray_intersect` de su esfera para que retorne un `Intersect` en lugar de solo un booleano. Por ahora solo tenemos un canal de luz (difuso) pero muy pronto vamos a tener muchos más. Pueden utilizar este código como referencia para su `intersect`:

```
// ray_intersect.rs

use nalgebra_glm::Vec3;
use crate::color::Color;
use crate::material::Material;

#[derive(Debug, Clone, Copy)]
#[allow(dead_code)]
pub struct Intersect {
    pub distance: f32,
    pub is_intersecting: bool,
    pub material: Material,
}

impl Intersect {
    pub fn new(point: Vec3, normal: Vec3, distance: f32, material: Material) -> Self {
        Intersect {
            distance,
            is_intersecting: true,
            material,
        }
    }

    pub fn empty() -> Self {
        Intersect {
            distance: 0.0,
            is_intersecting: false,
            material: Material {
                diffuse: Color::new(0, 0, 0),
            },
        }
    }
}

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

y modifiquen su esfera para que pueda guardar su material:

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

Modifiquen su función de `ray_intersect` para que retorne un objeto `Intersect`. Asegurense de agregar el cálculo de la distancia hasta el impacto dentro de su esfera, lo vamos a utilizar en la siguiente lección.

Pueden utilizar un esqueleto similar a este para su `material.rs`

```
use crate::color::Color;  
  
#[derive(Debug, Clone, Copy)]  
pub struct Material {  
    pub diffuse: Color,  
}  
  
impl Material {  
    pub fn new(  
        diffuse: Color,  
    ) -> Self {  
        Material {  
            diffuse,  
        }  
    }  
  
    pub fn black() -> Self {  
        Material {  
            diffuse: Color::new(0, 0, 0),  
        }  
    }  
}
```