RenderGPU
Segona pràctica de GiVD 2020-21

Abstract
Se ha iniciado la adaptación del paso 1 (aunque no se ha añadido lo que ya faltaba en la práctica 1). Completamos el paso 2,3, y 4, junto con algunos opcionales de dichos pasos, como el Toon Shading y el spotlight. El paso 5 está completo.

Un detalle importante a destacar del paso 3 de Lights, es que Ruben ayudó a corregir errores en Object para pasar las normales a la GPU, ya que antes solo pasabamos points y color, y con los cambios de Ruben podemos pasar points, colors, y normals. Por otro lado, Ruben también ayudó a pasar variables a la GPU en el caso de Lights. (Ayudando a Sebastian a corregir errores pasando el número de luces (numl), y la luz ambiental). También, Rubén se ha encargado de terminar el Plane del paso1 y corregir fallos, y de crear el cálculo para obtener normales(es la misma para todos los vértices) y el envío de la información a los shaders, junto con la textura del paso5.

Features

- Fase 1
    - Adaptació a la lectura de fitxers de dades
    - [x] Objectes -> Josep, Álvaro
    - [x] Escenes virtuals -> Josep, Álvaro
    - [ ] Escenes de dades Reals
    - [x] Material -> Sebastian
    - Light
    - [x] Puntual -> Sebastian
    - [x] Direccional -> Sebastian
    - [x] Spotlight ->Sebastian
    - [x] Ambient Global ->Sebastian
    - Shading
    - [x] Phong ->Ruben
    - [x] Gouraud ->Sebastian
    - Textures
    - [x] Textura com material en un objecte->Ruben
    - [x] Textura al pla base->Ruben

- Fase 2 (OPT)
    - [x] Toon-shading i èmfasi de siluetes ->Sebastian, Ruben
    - [ ] Mapping indirecte de textures
    - [x] Animacions amb dades virtuals -> Josep
    - [ ] Normal mapping
    - [ ] Entorn amb textures

- [ ] Reflexions
- [x] Transparencias via objectes -> Álvaro (en la versión del github está comentado pero se ha añadido una prueba/imagen de como se puede ver la transparencia).
- [ ] Transparencias via environmental mapping

Extensions

(NOTA: Les extensions de la pràctica que heu fet i que no surten a la llista anterior)

Memòria

Los constructores para el paso 2 y 3 están hechos tal y como lo pide el pdf.
En el caso de la implementación de las luces en los shaders, implementamos un if else que cambiaba la variable L según el tipo de luz. Para la luz puntual se usaba el cálculo de siempre de L = normalize(conjunto[i].origen - vPosition); En el caso de la luz direccional es simplemente invertir el valor de L = normalize(conjunto[i].direction * (-1)) en lugar de usar .origen. Para la luz spotlight utilizamos una intensidad de la luz similar a la que calculamos para ToonShading, pero invertida puesto a que es direccional (spotAngle). Usando la variable spotAngle podemos determinar si el objeto está dentro del spotLight, ya que si el ángulo coseno dado por spotAngle es mayor al que le pasamos al shader, quiere decir que efectivamente el objeto está dentro de la dirección en la que pasa la luz, y por tanto, podemos usar spotAngle para calcular el factor por el cual multiplicaremos a los componentes de ambient, specular, y diffuse. Utilizó un exponente fijo de 2 para calcular este factor, pero lo ideal sería que fuera un parámetro en el constructor de la luz.

En el caso del cálculo de los Shaders, es hacer un simple Blinn-Phong pero usando las normas pasadas por el objeto, tanto el caso de Gouraud como de Phong shading. La mayor diferencia entre estos 2 shaders es que en Gouraud hacemos el cálculo en el vertex Shader mientras que con Phong lo hacemos en el Fragment Shader.

Respecto al paso 5, se ha implementado el paso de texturas a la GPU en la clase Object.cpp a partir de los métodos initTextura() y toGPUTexture(). El método drawTexture() no se ha utilizado. Ha habido que añadir un updateGL en el método browse para que una vez cargada la textura, actualice automáticamente para ver cómo queda con la nueva textura cargada. En initTextura() se pasa por defecto una textura para evitar problemas de cara a la implementación, y luego ya con la MainWindow se puede alternar entre las que se quiera. También se ha usado el método mirrored() para la carga de una textura para evitar que se texturice únicamente la mitad del .obj. Posteriormente, se ha añadido al Plane el cálculo del vector normal a partir de 3 de sus vértices, y las coordenadas de textura en cada uno de sus vértices, para poder enviar a los shaders los vértices, colores, normales y coordenadas de textura en el orden pertinente.

Screenshots

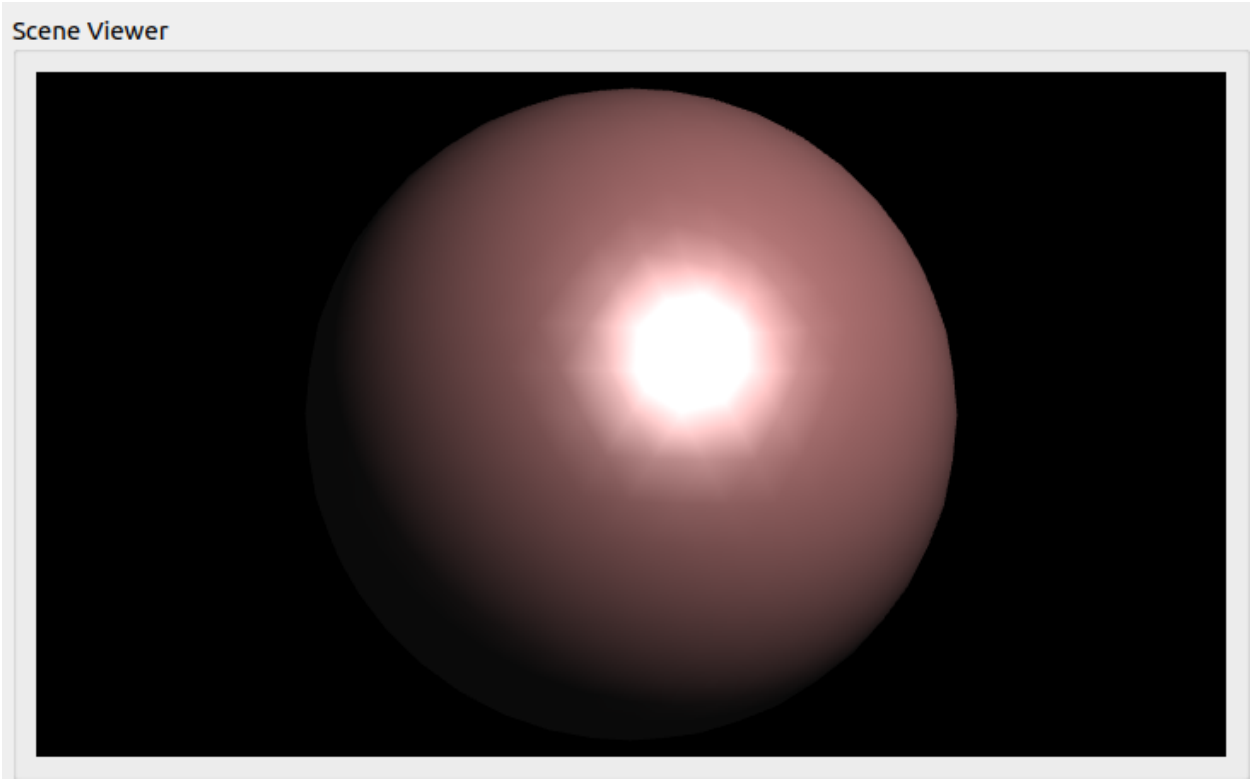
Paso 4:
Usando la luz puntual, llamamos al constructor en scene.cpp:

```
light = make_shared<Light>(Puntual, vec3(10,10,20), vec3(0.2, 0.2, 0.2), vec3(0.8, 0.8, 0.8),vec3(1.0, 1.0, 1.0), vec3(1.0, 1.0, 1.0));
addLight(light);
```

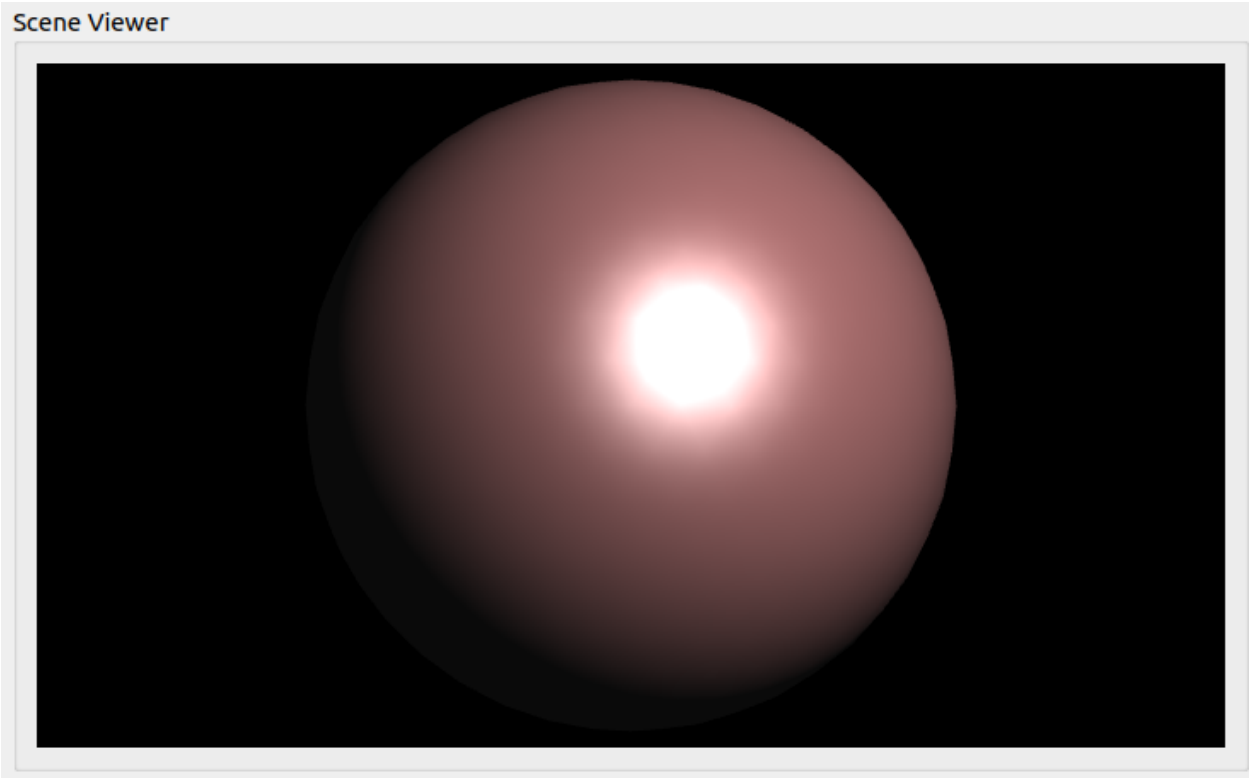
Usando el siguiente constructor de material, el cual llamamos desde Object.cpp:

```
material = make_shared<Material>(vec3(0.2, 0.2, 0.2),vec3(0.8, 0.5, 0.5),vec3(1.0, 1.0, 1.0),vec3(1.0,1.0,1.0),20);
```

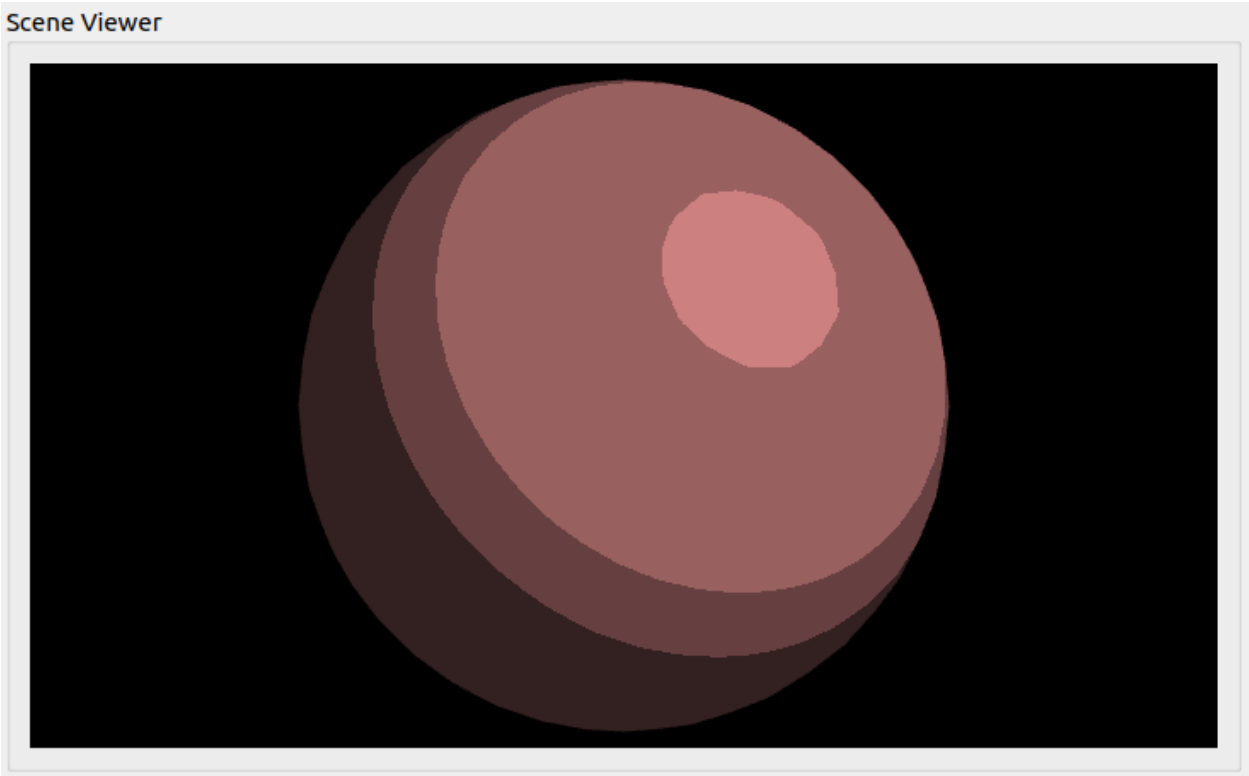
Gouraud Shader:



Phong Shader:



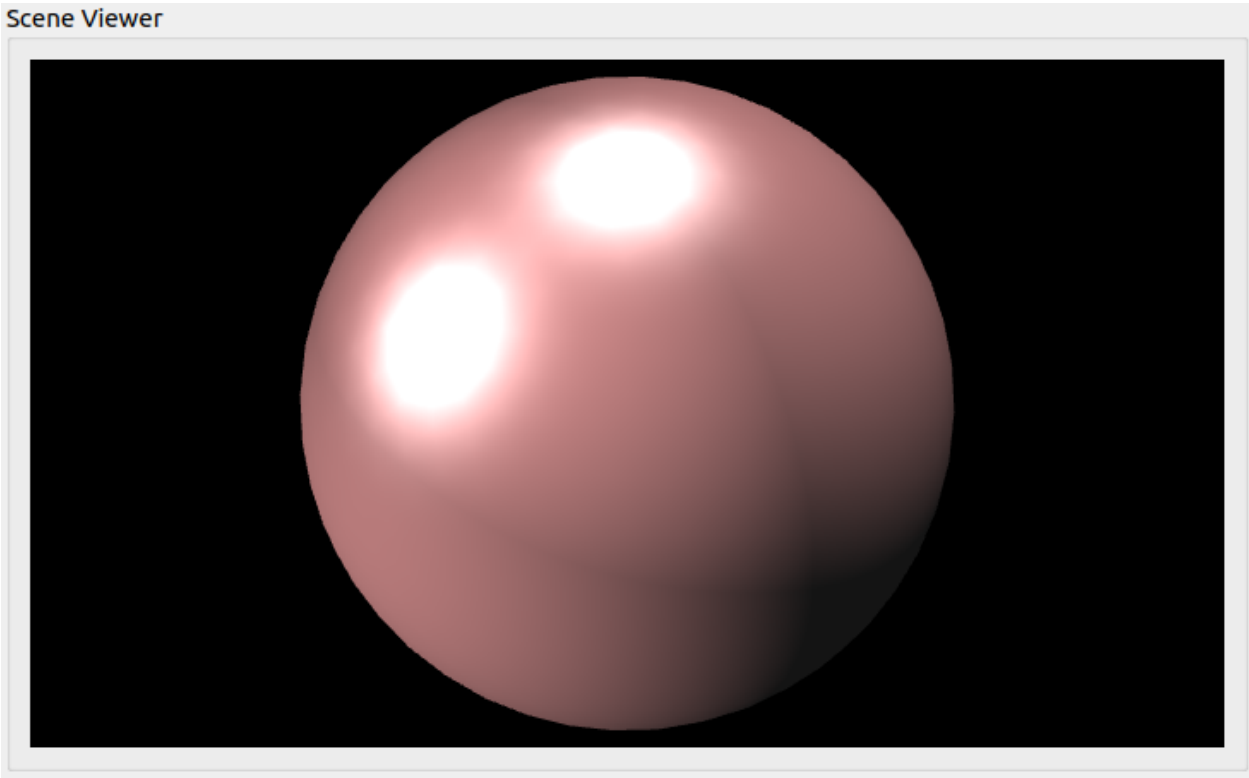
Toon Shader:



NOTA: toon shader está hardcodeado. No entendemos porque al adaptar las fórmulas anteriores en el fragment shader, nos sale un error.

Utilizando varias luces y el mismo material de antes:

```
    light = make_shared<Light>(Puntual, vec3(10,10,20), vec3(0.2, 0.2, 0.2), vec3(0.8, 0.8, 0.8),vec3(1.0, 1.0, 1.0), vec3(1.0, 1.0, 1.0));
  addLight(light);

    light = make_shared<Light>(Puntual, vec3(-20,-20,10), vec3(0.2, 0.2, 0.2), vec3(0.8, 0.8, 0.8),vec3(1.0, 1.0, 1.0), vec3(1.0, 1.0, 1.0));
  addLight(light);
```

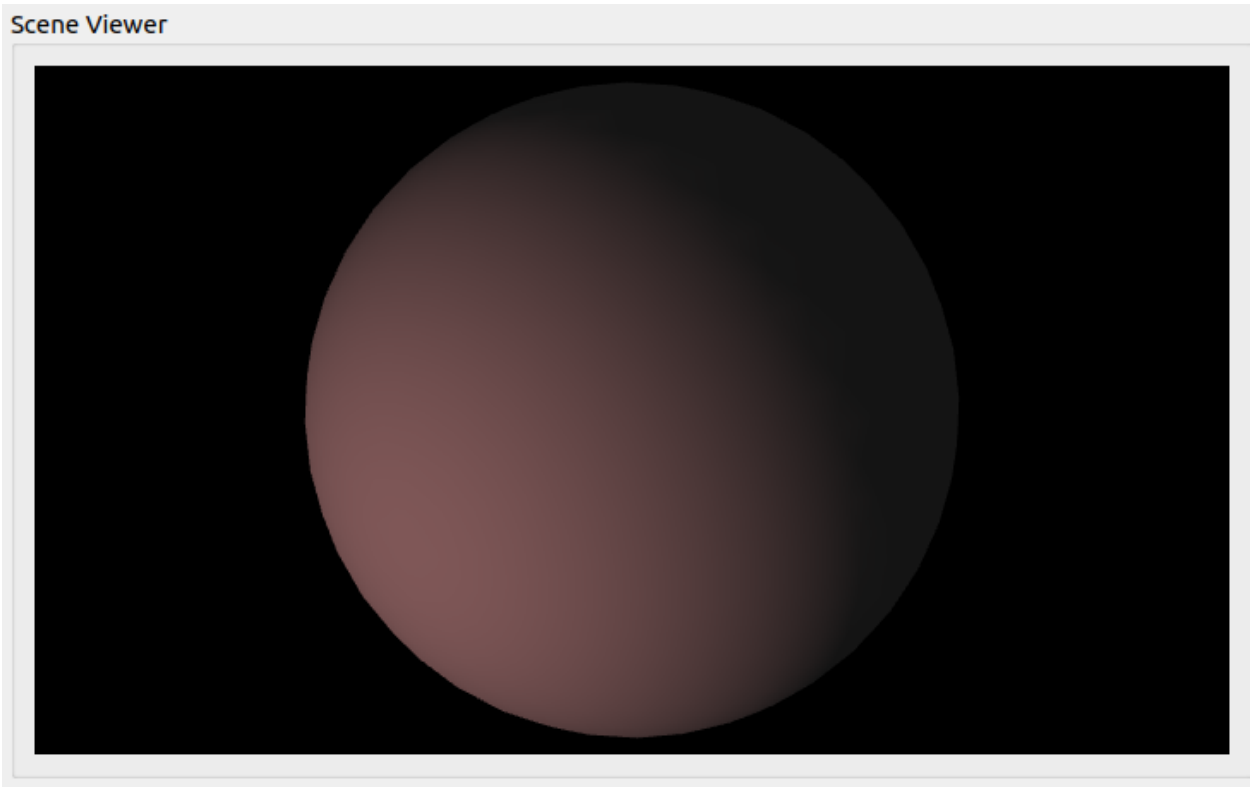


NOTA: la imagen está movida un poco para que se vea mejor la sombra.

Usando luz direccional:

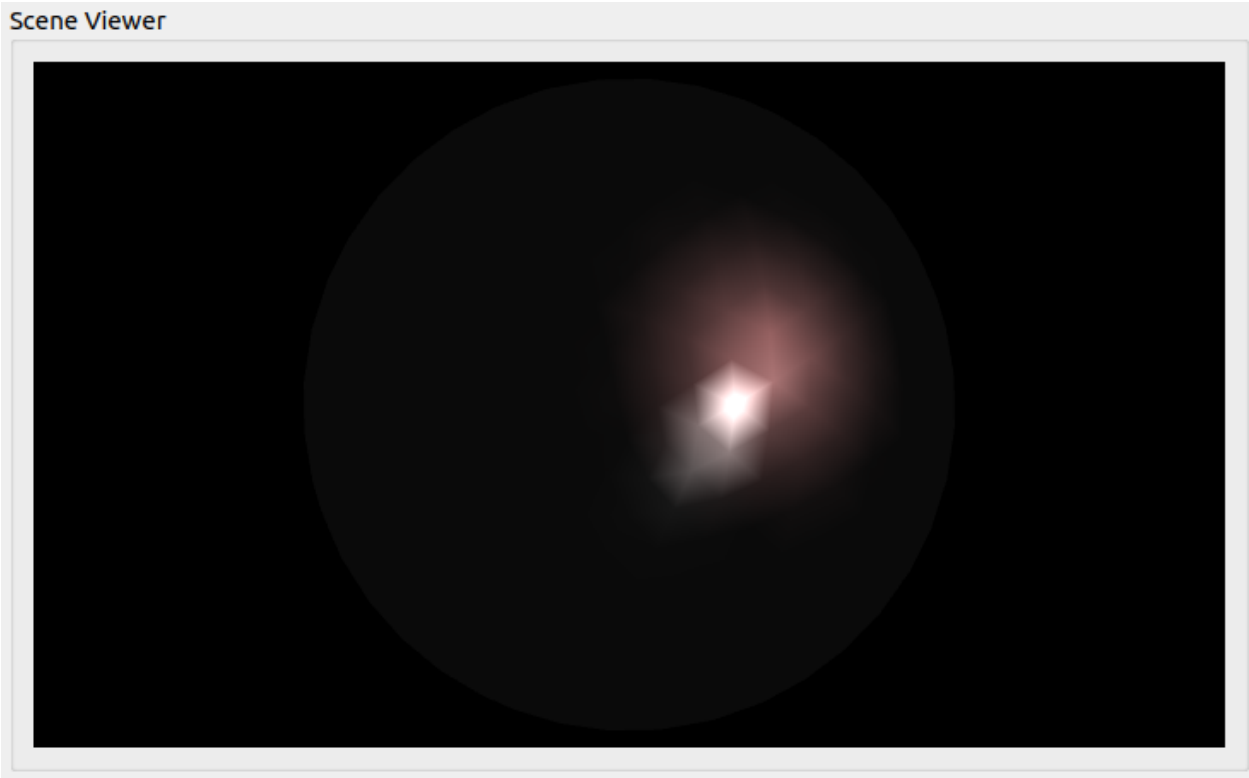
```
    light = make_shared<Light>(Direccional, vec3(10,10,20), vec3(0.2, 0.2, 0.2), vec3(0.2, 0.2, 0.2), vec3(0.8, 0.8, 0.8),vec3(1.0, 1.0, 1.0), vec3(1.0, 1.0, 1.0));
  addLight(light);

    light = make_shared<Light>(Direccional, vec3(-20,-20,10), vec3(0.2, 0.2, 0.2), vec3(0.2, 0.2, 0.2), vec3(0.8, 0.8, 0.8),vec3(1.0, 1.0, 1.0), vec3(1.0, 1.0, 1.0));
  addLight(light);
```

Usando .direction en la fórmula L:

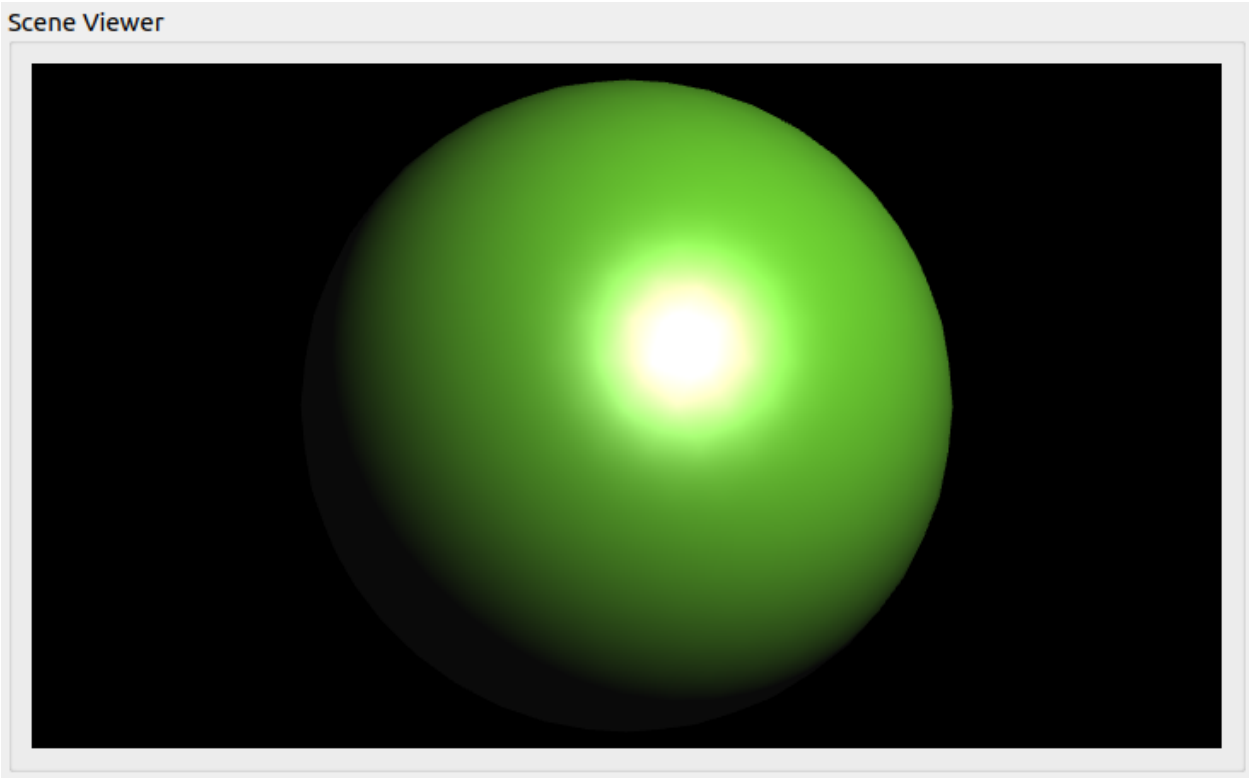


Usando spotlight:

```
  light = make_shared<Light>(Spot, vec3(10,10,20), vec3(0.2, 0.2, 0.2), 45.0,vec3(0.2, 0.2, 0.2), vec3(0.8, 0.8, 0.8),vec3(1.0, 1.0, 1.0), vec3(1.0, 1.0, 1.0));
  addLight(light);
```

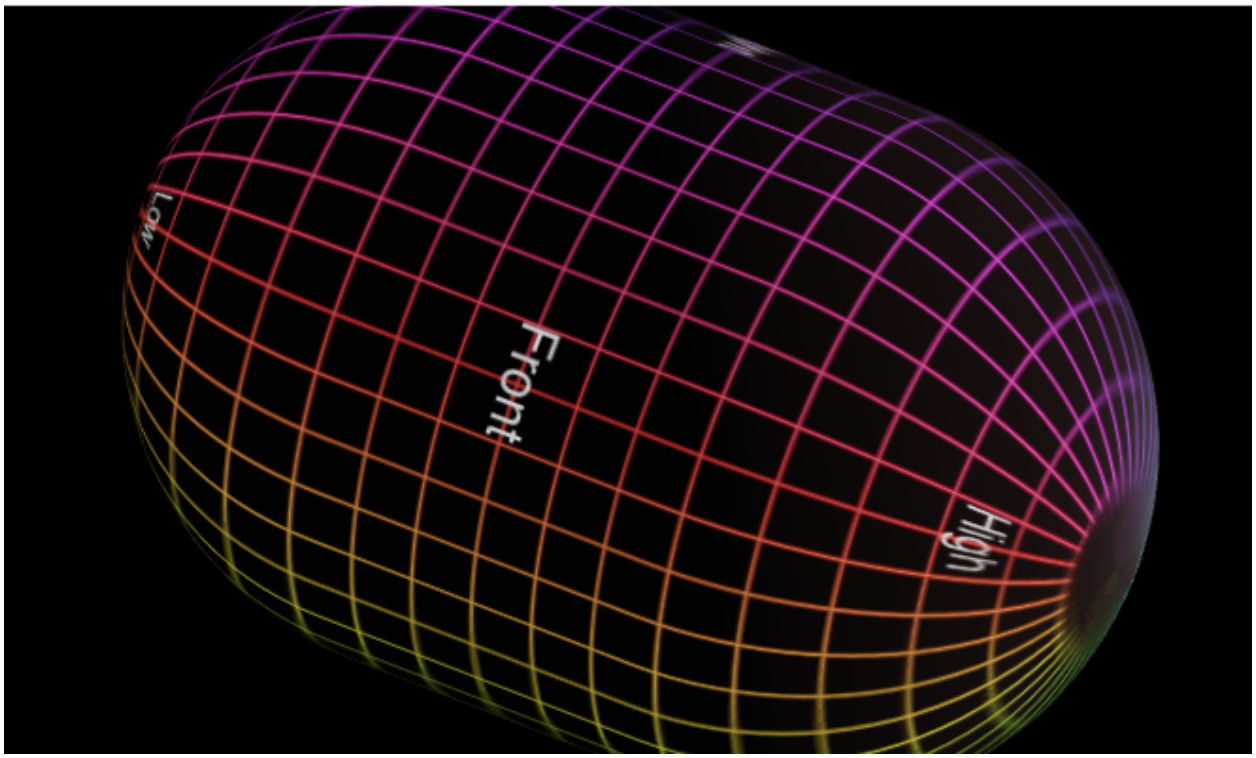


Podemos probar con otro material y funciona igual para todos los shaders.

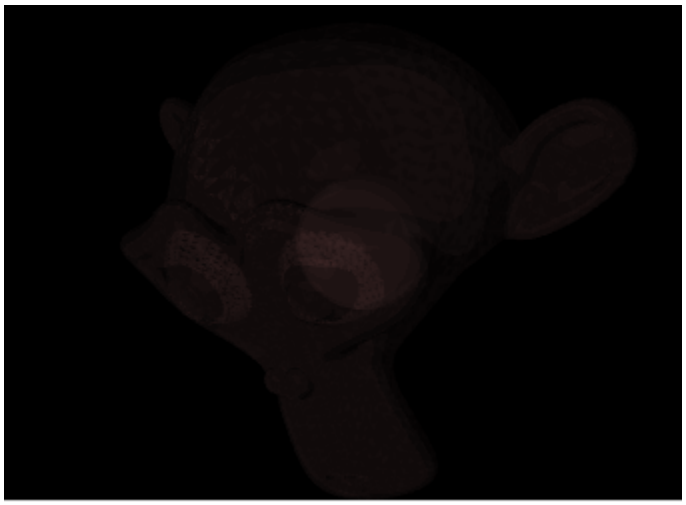


```
material = make_shared<Material>(vec3(0.2, 0.2, 0.2),vec3(0.5, 1.0, 0.2),vec3(1.0, 1.0, 1.0),vec3(1.0,1.0,1.0),20);
```

Texturas Paso 5:
Phong Tex:


Low
Front
High

Transparencia:
Como se puede apreciar podemos ver la esfera dentro del mono transparente.



Para hacerlo, se ha modificado la función initializeGL() además de modificar en el código (hardcodeado) el valor color.a (el cuarto del vec4) a un valor menor a 1.0 para los shaders.
Concretamente para este ejemplo, se ha usado el vToonshader con valor 0.1 de opacidad/transparencia.

```
void GLWidget::initializeGL() {
	glEnable(GL_DEPTH_TEST);
	//glEnable(GL_CULL_FACE);   //Optativa Trnasparencias (comentada)
	glEnable(GL_RGBA);
	glEnable(GL_DOUBLE);
	glDisable(GL_CULL_FACE); //Optativa transparencias (añadida)

	// Enable blending      //Optativa transparencias (añadida)
	glEnable(GL_BLEND);          //Optativa transparencias (añadida)
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);          //Optativa
transparencias (añadida)
```

Additional Information

Si copiamos el if else para spotlight que tenemos en Gouraud, y lo agregamos en Phong o Toon shader, que usan el fragment shader, nos sale este error.

```
11:58:09: Starting /home/sebas/build-RenderGPU-Desktop-Debug/RenderGPU ...
QGLShader::compile(Fragment): 0:59(5): error: operands to relational operators must be scalar and numeric
0:59(5): error: if-statement condition must be scalar boolean
0:63(5): error: operands to relational operators must be scalar and numeric
0:63(5): error: if-statement condition must be scalar boolean

QGLShader::link: "error: linking with uncompiled/unspecialized shader"
QGLShader::link: "error: linking with uncompiled/unspecialized shader"
```

Es un error que miramos y seguimos sin entender cómo solucionarlo, lo cual nos impide implementar todos los tipos de luces en Phong y Toon shading, el unico tipo de luz que no acepta es el Spotlight cuando usamos fragment shader. De hecho, una de las razones por la que Toon shading esta hardcodeada es por este error.

Para el paso 1, se han hecho todas las adaptaciones necesarias para que funcione lo que ya iba en la práctica anterior, pero no hemos añadido lo que faltaba por incluir (la parte ligada a datos reales). Además, en la lectura de escenas virtuales, si lee que es un brobject, este se pasa por widget, ya que no es capaz de abrirlo pasándolo por fichero, aun llegando el filename correcto, cosa que ya nos ocurría en la práctica 1.

El plano está implementado, pero al no ser hijo de Object no tenemos muy claro cómo agregarlo. Hemos intentado hacer que sea un hijo de object para poder usar el método addObject() y agregarlo directamente a la escena, pero aun así no hemos podido.

En ocasiones la aplicación crashea antes de lanzarse. La solución es simplemente intentarlo hasta que arranque. A veces con clickar 2 veces funciona.

```
23:19:19: Starting /home/sebas/build-RenderGPU-Desktop-Debug/RenderGPU ...
23:19:21: The program has unexpectedly finished.
23:19:21: The process was ended forcefully.
23:19:21: /home/sebas/build-RenderGPU-Desktop-Debug/RenderGPU crashed.
```