Essent aquesta la reintroducció a conceptes que van ser comentats al llarg de l’assignatura de *Computer Graphics*, en aquet primer *lab* s’ha treballat envers els conceptes de nodes, materials, llums i *shader;* duent-se a terme el renderitzat d’una senzilla escena tenint en compte tant conceptes com la il·luminació mitjançant l’equació de Phong com únicament texturitzant així com treballar en l’aplicació d’un *skybox*. L’estructura escollida ha sigut disposar com a nodes de l’escena les pròpies *meshes*, les llums i el *skybox*; essent cada tipus emmagatzemat per separat, és a dir, *meshes* i llums als seus respectius vectors i el *skybox* com una variable, doncs hem aplicat un *render multi-pass* on s’ha considerat que és més àgil emmagatzemar tot per separat que anar dividint un vector amb tots els nodes de l’escena incloent *meshes*, llums i *skybox*.

Una vegada definit això definim l’escena per defecte el llum ambient i un *skybox* així com un llum i un node, on tots podran ser canviats una vegada dins de l’aplicació, on al propi render de l’aplicació s’ha afegit el render del *skybox* abans de renderitzar els diferents nodes sense dur a terme *depth test* i que no es pinti davant de cap objecte.

Dit això, el següent que hem dut a terme ha sigut la senzilla tasca d’aplicar una textura a una *mesh* concreta, sense complicar-ho amb il·luminació o reflexos.Per tal de dur-ho a terme hem creat una subclasse *TextureMaterial* que hereta de *StandardMaterial* on no es sobreescriu cap mètode d’aquesta, canviant únicament el *fragment shader* utilitzat, passant del *flat.fs* al creat per nosaltres *texture.fs*, donant-se això degut a què no eren necessàries noves *uniforms* ni s’havien de dur a terme canvis al *render*.

Respecte el *shader* *texture.fs*, és l’encarregat de simplement aplicar la textura corresponent a la *mesh*, on passem com a uniforms el color del material (per si s’ha modificat) així com la pròpia textura i fem servir els valors de les coordenades de textura *v\_uv* provinents del vèrtex shader (basic.vs) per llegir el texel concret amb la funció *texture()* per definir el color final com el color de la textura multiplicat pel color decidit per l’usuari, on blanc correspon a la textura sense canvis.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

*Figura 1: Dos meshes de tipus helmet on la de l’esquerra ha sigut texturitzada aplicant a la vegada un color (35, 116, 166), mentre que la de la dreta ha sigut texturitzada amb un color (255, 255, 255), per tant sense patir cap canvi*

A continuació, vam centrar els nostres esforços a definir un *skybox* per l’escena, on vam crear una subclasse *Skybox* de *SceneNode* que sobreescrivia els mètodes *render()* i *renderInMenu()*. Per tal d’aplicar un *skybox*, hem decidit fer servir *cubemaps* pintats a una *mesh* consistent en un cub, donant suport a fer servir servir directoris que contenen les imatges corresponents a cada cara i construint la textura *cubemap* mitjançant *cubemapFromImages()*. Per altra banda, per tal d’evitar que al moure’s la càmera es pugui sortir del propi cub, al render el que fem és que sigui molt similar al de *StandardMaterial* però situant la *model matrix* del cub a la posició de la càmera amb la funció de *Matrix44* *setTranslation()*. Per últim, cal destacar que es pot canviar el *skybox* utilitzat amb el procés ja comentat tenint disponibles tots els que es troben al directori *environments*.

De cara a pintar el *skybox*, no podíem fer servir el *shader* que havíem fet servir anteriorment doncs ara estàvem treballant amb *cubemaps*, que tenen una lectura diferent, per tant hem definit un segon *shader*, no gaire més complex però sí més interessant. En aquest cas per tal de llegir el cubemap fem servir *textureCube()*, on en aquest cas no ens serveixen els UVs provinents del *vertex shader* (*basic.vs* de nou) doncs per tal de saber quina posició de la textura s’ha de llegir es fa servir un vector. Per tant, partint del vector V el qual es defineix iniciant-se de la posició en coordenades de món que volem pintar i apuntant cap a l’ull (la càmera), en aquest cas busquem el vector des de l’ull fins a la posició a pintar, definint per tant un vector per tal d’obtenir el valor de la textura al punt que volem pintar.

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza media

*Figura 2: Mesh de tipus helmet texturitzada amb rerefons format per un skybox amb el cubemap corresponent a San Giuseppe Bridge aplicat*

Una vegada disposàvem de s*kybox* per l’escena, el següent que duríem a terme seria renderitzar la *mesh* com un mirall que reflectís el *skybox* de l’escena. Per dur-ho a terme, hem afegit un altre material creant la subclasse *ReflectiveMaterial* que hereta de *StandardMaterial* i on no cal sobreescriure cap mètode doncs pot aprofitar perfectament tots els definits a la classe pare.

El *shader* que s’utilitza per dur-ho a terme és *reflective.fs*, on l’estructura és similar a la de *skybox.fs* però disposem d’un altre vector a tenir en compte: el vector reflectit R. El raonament darrere això és que si abans teníem en compte el vector de l’ull al punt del *skybox* concret ara necessitem que el material reflecteixi el *skybox* basat en l’angle de visió que tenim on, per tant, necessitem el vector reflectit a la superfície de la *mesh* respecte la normal per tal d’obtenir el vector que ens indicarà el vector director per indexar el *cubemap* i obtenir el color corresponent.

Pantalla de un video juego

Descripción generada automáticamente

*Figura 3: Meshes de tipus sphere i helmet amb un material reflector aplicat reflectint el skybox Panorama*

Continuem ara amb el primer cas d’il·luminació seguint l’equació de Phong on hem creat una subclasse *PhongMaterial* la qual hereta de *StandardMaterial*, afegint quatre atributs que descriuen com es comporta el material respecte diversos factors d’il·luminació: *ambient* en relació al llum ambient, *diffuse* amb el component difús del llum, *specular* pel corresponent component especular i, finalment, *shininess* per graduar el *power* de com afecta el llum al factor especular. A més, s’han sobreescrit els mètodes *render()*, *setUniforms()* i *renderInMenu()*.

Respecte les sobreescriptures és de gran importància comentar el *render*, on com s’ha comentat a l’inici hem dut a terme un *render multi-pass*, on renderitzem cada node una vegada per cada llum aplicant un *blending* additiu GL\_ONE perquè sempre es sumi, no tenint en compte l’*alpha* pel moment. A més, de cara a que el llum ambient no es tingui en compte més que per la primera passada, s’indica com a a partir del segon llum a tenir en compte.

Per tal d’aplicar l’equació, disposem del *shader phong.fs*, descrivint l’equació a nivell de fragment i no vèrtex per obtenir una millor qualitat final. El procediment és bastant senzill i segueix la pròpia equació que ja de per si mateixa és força clara, on l’estructura que segueix parteix de llegir inicialment la textura (que pot ser canviada de color amb l’indicat per l’usuari com anteriorment) on el color resultant es multiplicarà per les diferents propietats dels materials per ja tenir calculats els corresponents *,* i . El que resta són les conseqüents multiplicacions i productes escalars entre els diferents components de l’equació, on es defineixen per una banda el vector , partint de la posició de món que s’està renderitzant cap a la posició de món del llum en qüestió així com el vector ja mencionat, essent el vector en aquest cas el que reflecteix el vector incident del llum (-) i no el de , establint la *shininess* del material com la potència a la qual s’eleva el producte escalar entre els vectors i al càlcul del component especular.

Una captura de pantalla de un videojuego

Descripción generada automáticamente con confianza media

*Figura 4: Mesh de tipus lantern aplicant un material que aplica l’equació de Phong on la llum s’ha desplaçat a (-2.6, 94.2, 72.6) i s’ha canviat la shininess a 50*

Finalment, dedicarem unes línies a comentar les bondats de *ImGUI* i les possibilitats que permetem així com les seves representacions. Centrant-nos primer en les possibilitats a nivell d’escena, on hem afegit els canvis al fitxer *main.cpp*¸ on hem permés canviar el llum ambient representat com un color per facilitar la seva visualització, afegint també botons per afegir llums i *meshes* augmentant així la complexitat de l’escena al gust de l’usuari, així com el canvi del *skybox* ja comentat.

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

*Figura 5: Mesh de tipus helmet amb dues llums i llum ambient a (0, 0, 0). El primer dels llums amb component difús (227, 244, 59) i component especular a (130, 200, 33) així com a la posició (-20.4, 23.5- -21.4). El segon llum*

Si baixem a nivell de node podem observar les possibilitats respecte els canvis que s’han mostrat a les imatges que han aparegut amb anterioritat, podent canviar textures, materials i la pròpia geometria, on en tots aquests casos les opcions es defineixen com a combos degut a què són opcions tancades. Ara bé, encara queda a nivell de material que també conté bona quantitat d’opcions de personalització, on tenint en compte que són el que hem afegit ens centrarem a les opcions a nivell de *PhongMaterial*, doncs els altre tipus no permeten canvis al no intervenir les propietats del material. En aquest cas trobem que podem personalitzat les propietats del material descrites com a colors (excepte la *shininess*) doncs en la mateixa idea de les propietats de la llum, creiem que és una representació molt més còmode de cara a l’usuari al veure’s empre delimitat al rang (0,1) que s’ha de tenir en compte al *shader*.

Pantalla de un video juego

Descripción generada automáticamente