Essent aquesta la introducció a conceptes que vam comentar a llarg de l’assignatura de *Computer Graphics*, s’ha treballat envers els conceptes de nodes, materials, llums i *shaders* on a continuació explicarem en detall les decisions de disseny i definició preses per dur a terme la feina.

**Materials**

Comencem des dels materials poder anar pujant en la jerarquia, on primer de tot cal comentar que hem fet un petit canvi a la classe *StandardMaterial* on hem indicat que el *shader* utilitzar per renderitzar aquest material sigui el corresponent a aplicar la textura únicament, sense cap tipus d’il·luminació, sense ser necessària dur a terme cap canvi més al codi.

[Imagen del casco texturizado]

Posteriorment, per tal d’aplicar l’equció dePhong per la il·luminació hem creat un material *PhongMaterial* el qual hereda de *StandardMaterial* per tal de tenir accés a funcions com la definida per pujar les *uniforms*, 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.

Dit això, sobreescrivim els tres mètodes de *StandardMaaterial* doncs requereixen forces canvis, on a *setUniuforms()* el que necessitem és afegir les propietats del material respecte les uniforms que ja es passaven a la classe pare, on hem reaprofitat el propi mètode per passar les uniforms corresponents. De cara al *render* és on es conté la major part del contingut nou, on hem dut a terme un *render multi-pass* amb no més motiu que la comoditat personal, on renderitzaem cada node una vegada per cada llum, aplicant un *blending* additiu GL\_ONE perquè sempre es sumi, no tenint en compte *alhpes* 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 0 a partir del segon llum a tenir en compte. Finalment, respecte les opcions a *ImGui* permetem canviar les propietats del material abans mencionades així com el propi color.

Per últim, hem afegit un últim material anomenat *ReflectiveMaterial* que hereda de *StandardMaterial* i al qual només es defineix el constructor i destructor perquè pot aprofitar perfectament tots els mètodes definits a la classe pare, canviant el constructor doncs necessitem que s’apliqui el *shader* que pinti el *skybox* reflectit així com indicar el cubemap del *skybox* com a la textura utilitzada.

***SceneNodes***

Envers els nodes que contenen l’escena, hem dut a terme una sèrie de canvis a diferents mètodes de la classe *SceneNode*, essent els corresponents als contructors per definir una *mesh* i *material* per defecte una vegada es crea un, on la major part dels canvis es donen a la funció renderInMenu() per tal de tenir més opcions a ImGui. Per tal de dur-ho a terme hem afegit una sèrie d’atributs (*mesh\_selected*, *material\_selected* i *texture\_selected*) per tal de poder indicar quin es selcciona de cadascun, on respecte a les opcions estan predefinides respecte els diferents models amb els que es té al framework.

Dit això, per tal d’aplicar l’equació de Phong necessitàvem una forma de definir llums i que continguin la informaci´´o necessària, creant-se doncs una subclasse de *SceneNode* anomenada lightamb els atributs *diffuse* i *specular* per indicar les propietats de la llum necessàries i únicament sobreescriu *renderInMenu()* doncs són diferents paràmetres a canviar i no necessita cap render. Per tant, mitjnçant ImGui es permet canviar els components *diffuse* i *specular* del llum així com la seva posició, donant-se que pel moment només comptem amb *point lights*.

Per últim, comptem amb una altra classe que hereda de *SceneNode* anomenada *Skybox* que, en efecte, és l’encarregada de gestionar el *skybox* de l’escena, la qual sobreescriu tant el render com el renderInMenu() de *SceneNode*. Ara bé, aquí cal destacar un parell de coses, essent la primera el fet de què el render només té un petit canvi respecte el de la classe pare i és el corresponent a centrar-ho a la posició de la càmera, doncs no volem sortir de l’espai d’aquest. El segon a detacar és que hem decidit que es puguin fer servir tant *skyboxes* en format HDRE com formats per diferents imatges, on en el cas dels priemera els convertim a textura mitjançant cubemapFromHDRE mentre que pels segons fem servir cubemapFromImages amb un directori concret.

***Shaders***

El primer de tots els *shaders* és el corresponent a *texture.fs* que és l’encarregat de simplement aplicar la textura corresponent a la mesh, sense aplicar cap tipus d’iluminació, on simplement 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 definir el color final com el valor de la textura multiplicat pel color per les variacions corresponents.

De cara a pintar el *skybox* 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 doncs funcionen amb vectors. El vector en qüestió és el vector V, el vector que apunta cap a l’ull a l’escena, essent en aquest cas la càmera amb la seva posició pasada com a uniform fins la world possition passada del vèrtex shader, normalitzant el vector i fent-lo servir directament.