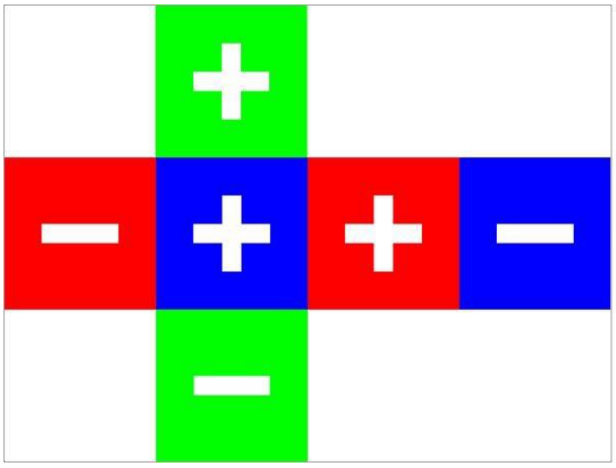
TP – Cube mapping

Malek.Bengougam@gmail.com

Partie 1 – Cube Map

L'utilisation principale de la technique du Cube Mapping est la simulation d'environnements distants. On utilise six (6) textures représentant un point de vue lointain capté depuis l'origine de la scène.



Remarquez que le repère d'une Cube Map est un repère main gauche (Z positif vers le fond). Chacune des images a été capturée avec un FOV de 90° dans les six directions cardinales.

Du point de vue d'OpenGL, une CubeMap est un Texture Object composé d'un tableau de six textures. Il faut d'abord créer un *Texture Object* comme d'habitude mais cette fois-ci avec une cible (target) spécifique qui est **GL_TEXTURE_CUBE_MAP**. Ensuite il faut charger chacune des faces de la Cube Map. Chaque face de la CubeMap est identifiée individuellement par une constante spécifique de la forme :

GL_TEXTURE_CUBE_MAP_{POSITIVE|NEGATIVE}_{X|Y|Z}.

Les constantes ont une valeur croissante à partir de **GL_TEXTURE_CUBE_MAP_POSITIVE_X**, c'est-à-dire +x,-x,+y,-y,+z,-z ce qui permet de faire ceci :

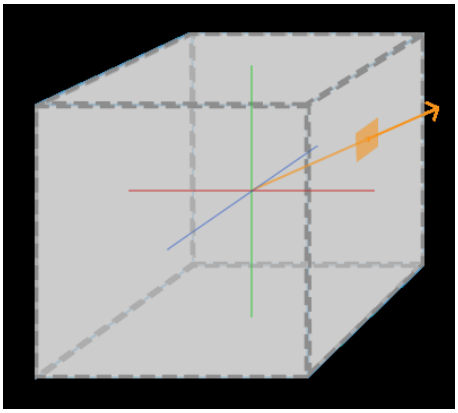
```
glGenTextures(1, &cubemap_texture);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemap_texture);
for (int i = 0; i < 6; i++)
      glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGBA8, w, h, 0,
GL_RGBA, GL_UNSIGNED_BYTE, cubemap_image[i]);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Le tableau cubemap_images[] est un tableau de six pointeurs vers données brutes des images de chacune des faces que l'on aura préalablement chargé avec la bibliothèque STB par exemple.
Les cubemaps sont échantillonnées via des coordonnées de textures 3D. En pratique, Il n'est pas nécessaire de spécifier de coordonnées de texture car on va mettre les interpolateurs du GPU à notre profit (de façon un peu différente en fonction de ce que l'on souhaite faire).

En GLSL il faut un sampler particulier pour échantillonner une cubemap, c'est le type **samplerCube**.

En OpenGL (ES) 2.0 on utilise la fonction `textureCube(samplerCube sampler, vec3 coords)` tandis qu'en OpenGL 3.x on utilise la fonction `texture(samplerCube sampler, vec3 coords)`.

Pour mieux comprendre comment le sampling fonctionne il faut s'imaginer être au centre de la cubemap. Les coordonnées 3D sont interprétées comme des vecteurs directeurs qui vont permettre au GPU d'échantillonner la bonne texture aux coordonnées 2D qu'il aura pu calculer.



Par exemple, lorsque la composante x est positive et majeure (abs(x) > abs(y) et abs(x) > abs(z)) le GPU comprend qu'il doit lire la texture **TEXTURE_CUBE_MAP_POSITIVE_X** et utiliser (y,z) comme coordonnées de texture 2D.

Dans le cas où la composante z serait négative et majeure il lirait la texture **_NEGATIVE_Z** avec (x, y) pour coordonnées de texture 2D.

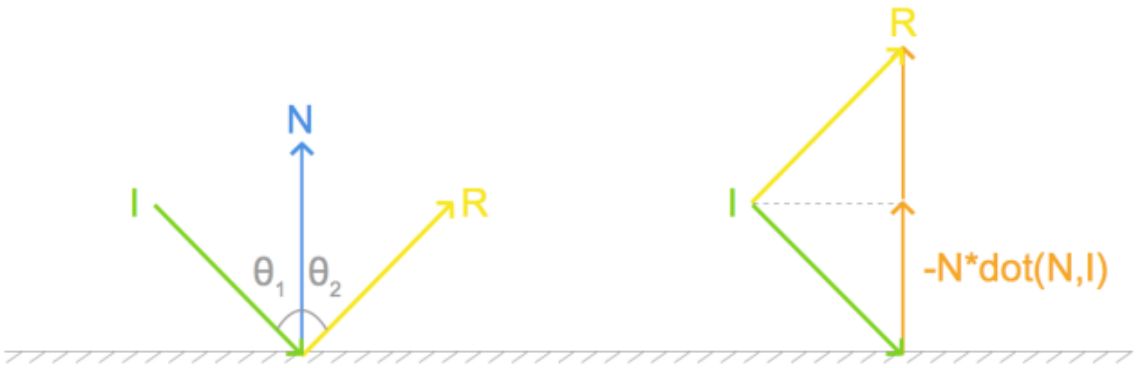
Partie 02 - Environment Mapping

L'idée exposée par cette technique est d'interpréter l'environnement distant comme source lumineuse.

Autrement dit, chaque texel d'une Cube Map devient potentiellement une lumière distante.

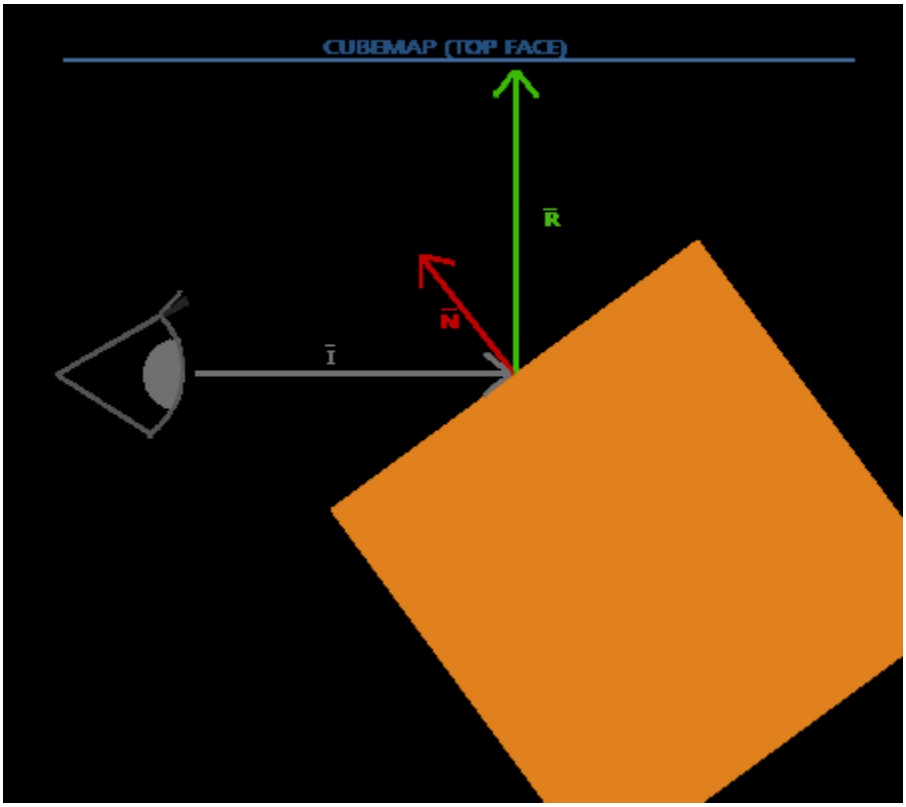
On peut utiliser des Cube Maps pour simuler une illumination ambiante de type diffuse ou spéculaire.

Il est très facile de produire le rendu d'un objet métallique très lisse (chromé par exemple) à l'aide de cette technique. En effet la couleur des objets métalliques provient exclusivement de leur réflexion spéculaire (ici avec une rugosité nulle). On sait que la réflexion spéculaire suit la loi de Descartes :



I correspond au vecteur incident de la lumière (vecteur arrivant à la surface), N étant la normale à la surface et R le vecteur réfléchi de I par rapport à N. Pour avoir un rendu chrome il suffit d'utiliser le vecteur R comme coordonnées de texture 3D au moment de la lecture de la CubeMap.

En GLSL, la fonction **`reflect(vec3 I, vec3 N)`** effectue ce calcul : attention le premier paramètre doit être un vecteur incident : le vecteur lumineux doit être orienté vers la surface.



Note 1: ceci n'est correct que pour les objets métalliques idéalement spéculaire (lisses, miroirs). Le cas des surfaces rugueuses est plus complexe et nécessite une implémentation beaucoup plus lourde en calculs.

*Note 2: cette technique peut s'appliquer à la composante diffuse en utilisant la normale **N** plutôt que **R**.*

2.1 exercices avancés (optionnels) :

a. Il existe une autre forme d'interaction lumineuse qui est la réfraction. Il s'agit du degré de déviation d'un rayon lumineux en passant d'un milieu (l'air par exemple) à autre (l'eau par exemple), qui est appelé indice de réfraction (lois de Snell-Descartes). En glsl, cela correspond à la fonction **refract**().

b. Augustin-Jean **Fresnel** a établi une équation permettant de mettre en relation la réflexion et la réfraction. En fonction de la nature du matériau une surface devient progressivement réflective au fur et à mesure que l'observateur forme un angle important avec la normale de la surface. Christophe **Schlick** a développé une approximation de l'équation de Fresnel moins coûteuse en calcul.

Références

http://web.cse.ohio-state.edu/~wang.3602/courses/cse5542-2013-spring/17-env.pdf

https://learnopengl.com/Advanced-OpenGL/Cubemaps

Autre exemple cette fois-ci avec des textures 2D (“hémisphériques”) à la place

https://www.clicktorelease.com/blog/creating-spherical-environment-mapping-shader/