Équipe 5

Stéphane Boulanger - 909 333 220

William Boudreault – 536 787 536

Infographie

IFT – 3100

Travail pratique 2

Travail présenté à

Philippe Voyer

Table des matières

1 - Sommaire

Dans le cadre de ce projet, nous devions développer une application qui permet de construire, d'éditer et de rendre des scènes visuelles en trois dimensions. Nous avons donc choisi d'utiliser le logiciel OpenFrameworks qui nous a été présenté dans le cours afin de concevoir notre programme d'édition. Nous avons conçu une interface simple d'utilisation afin de permettre à l'utilisateur de bien pouvoir visualiser ces scènes. Pour la plupart des fonctionnalités de l'application, l'utilisateur a accès à une caméra qui est manipulable soit avec la souris ou avec les touches du clavier dans certains cas, ce qui lui permet de pouvoir bien observer l'effet des différents rendus sur les modèles utilisés.

Dans ce document nous allons uniquement focusser sur la deuxième partie de la session et traiter des fonctionnalités de cette application qui font appel à la texture, l'illumination classique, le lancer de rayon, la topologie et l'illumination moderne.

Étant encore des débutants dans le domaine de l'infographie, nous avons utilisé plusieurs ressources qui nous ont permis d'implémenter les nombreuses fonctionnalités qui se retrouvent dans ce projet. Par exemple, la majorité des shaders que nous utilisons ont été récupéré soit dans le répertoire GitHub qui contient les exemples du cours ou ont été trouvés après des recherches sur le web. Nous avons modifié ceux-ci afin d'avoir les résultats auxquels nous nous attendions, mais nous ne prétendons pas avoir réinventer la roue et écrit ces shaders à partir de zéro. En revanche, il nous a fallu comprendre le fonctionnement de ceux-ci afin de les intégrer à notre projet, ce qui en a fait un processus assez formateur et éducatif.

Vous trouverez ci-dessous un lien vers la vidéo qui présente toutes les fonctionnalités qui ont été implémentées dans ce projet. https://www.youtube.com/watch?v=cWTVQz7wAbI

2 - Interactivité

L'utilisateur a accès à plusieurs menus qui sont regroupé selon les différentes fonctionnalités qui sont implémentées dans le projet. Un menu principal à gauche de l'écran permet de choisir sur lequel des thèmes il veut travailler. Un menu secondaire est parfois disponible lorsqu'une fonctionnalité à des besoin bien particuliers et il se trouvera à droite de l'écran. Différent boutons et gradateurs permettent de régler les paramètres des modèles en cours d'édition. Lorsqu'il y a des touches de clavier disponible pour l'utilisateur, des instructions sont affichées dans le haut de l'écran afin de faciliter la prise en main de l'application.

La fonctionnalité qui permet d'effectuer les lancer de rayons génère une fenêtre pop-up qui permet à l'utilisateur de visualiser des scènes. Celle-ci est créée avec FreeGLUT. Le programme lit le fichier de scène passé en paramètre, calcule la taille des objets et récupère les objets. Ensuite, elle configure OpenCL en récupérant un ou des appareils compatibles sur la machine, alloue l'espace nécessaire sur les appareils compatibles pour générer la scène et compile le fichier kernel « *rendering_kernel.cl* » qui contient la logique d'intersection et de lancers de rayons. Les touches qui permettent de manipuler la scène et leur fonctionnement sont indiquées à l'ouverture de la fenêtre pop-up. La touche « h » est utilisée pour cacher ou afficher le menu. Pour fermer cette fenêtre il suffit de cliquer sur le " *x* " qui se situe dans le coin supérieur droit de la fenêtre.

3 - Technologie

Pour les choix technologiques du projet, nous avons opté pour l'utilisation d'OpenFrameworks avec Visual Studio 2022. Nous avons jumelé OpenFrameworks à Visual Studio 2022 parce que l'intégration se faisait aisément. En effet, le générateur de projet OpenFrameworks permet de créer un projet directement dans Visual Studio 2022 sans complications. Visual Studio 2022 est une plateforme très répandue et le soutien est facile à obtenir en cas de problème. OpenFrameworks est construit avec la technologie d'OpenGL et s'utilise avec le langage de programmation C++. Nous avons utilisé la version 4.3 d'OpenGL ce qui nous permet d'utiliser les shaders qui sont supporté par cette version.

Au niveau de l'implémentation du lancer de rayons, nous avons décidé d'utiliser smallptGPU qui utilise le GPU pour les calculs et nous avons eu besoin de récupérer des librairies externes pour modifier le code source. Nous avons dû récupérer les librairies Boost C++ suivantes :

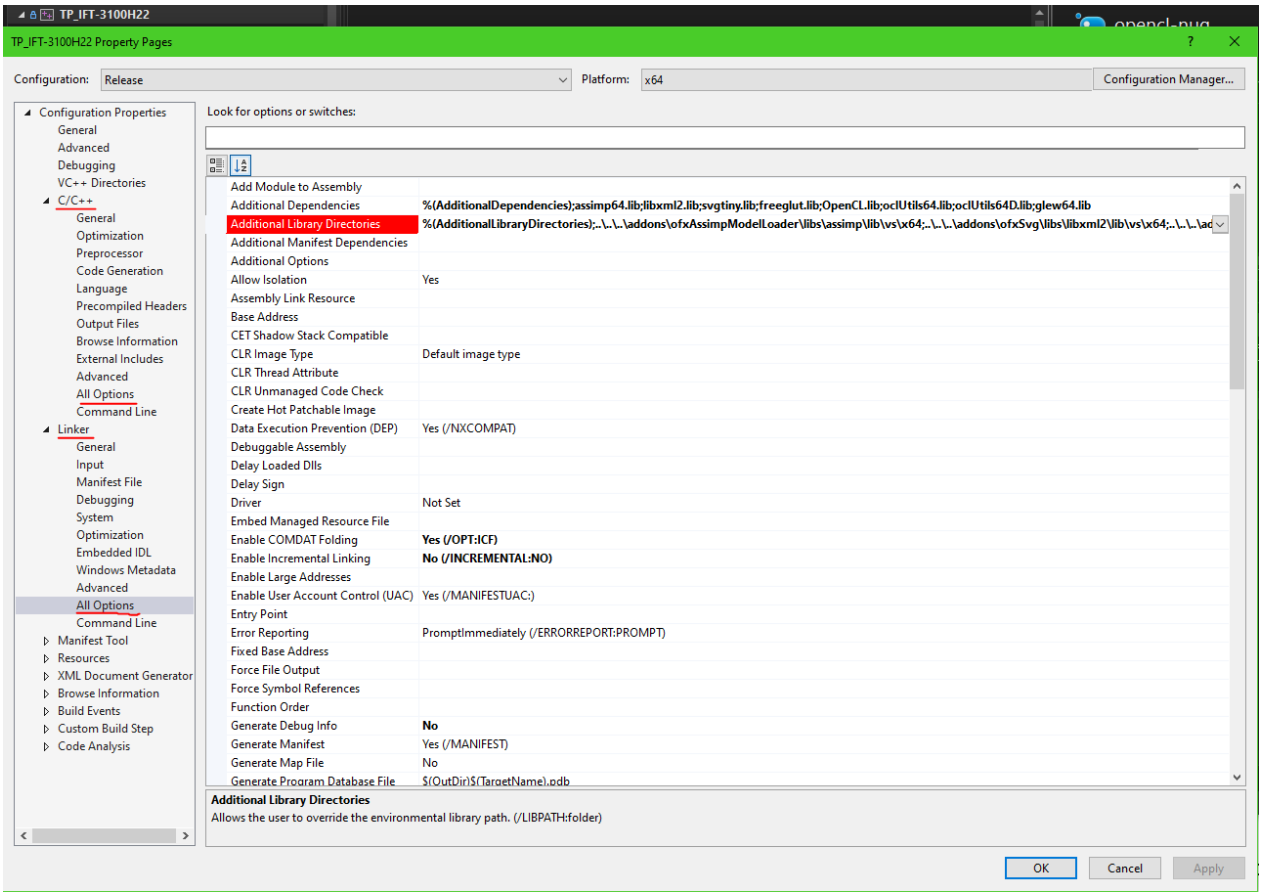
- libboost_chrono-vc141
- libboost_date_time-vc141
- libboost_thread-vc141

Ainsi que les dépendances prises d'un exemple d'utilisation de OpenCL du site de dev NVIDIA https://developer.nvidia.com/opencl pour faire fonctionner smallptGPU qui utilise une ancienne version de OpenCL (la version que nous avons prise est « OpenCL Multi Threads »). Le code source des dépendances OpenCL a été modifié pour utiliser une interface hpp prise d'un NuGet package de Visual Studio (opencl-nug version 0.777.77) qui communique avec les fichiers OpenCL de NVIDIA. Les librairies de l'exemple NVIDIA qui ont été récupérés sont :

- freeglut (version 64 bit de glut32 et qui permet d'arrêter l'exécution d'une fenêtre sans arrêter le programme au complet)
- glew64
- oclUtils64 et oclUtils64D
- OpenCL.

Nous avons également récupéré tous les fichiers C++ dans les dossiers « inc » au même niveau des dossiers « lib » de l'exemple NVIDIA téléchargé.

Toutes ces librairies externes ont été incluses dans le « Linker/All Options » du projet Visual Studio, soit dans « Additional Library Dependencies » ou « Additional Dependencies » et tous les fichiers C++ qui étaient dans les dossiers « inc » ont été inclus dans « C/C++ / All Options », au même niveau que le « Linker », dans « Additional Include Directories ». (Image à la page suivante)



4 - Compilation

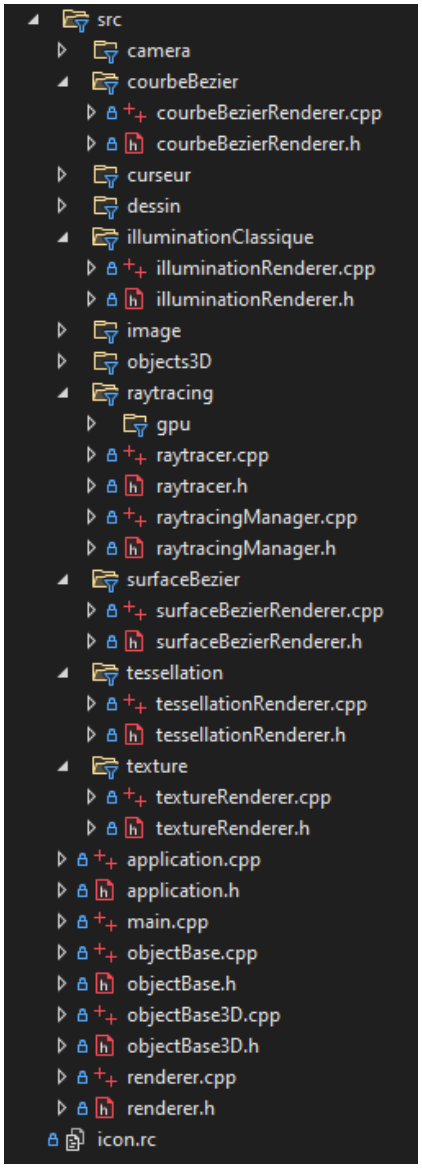
Notre projet a été testé et conçu pour être compilable et exécutable sous Windows 10 et Windows 11. Vous devez avoir déjà installé la version 0.11.0 d'OpenFrameworks disponible sur ce site https://openframeworks.cc/download/ et aussi téléchargé la version 2022 de Microsoft Visual Studio disponible à l'adresse suivante https://visualstudio.microsoft.com/fr/downloads/ .

Afin de compiler le projet il faut suivre les étapes suivantes.

1. Depuis le répertoire d'installation d'OpenFrameworks 0.11.0., extraire le dossier *TP_IFT-3100H22.zip* dans le répertoire *\apps\myApps*.
2. Ensuite exécuter la solution *TP_IFT-3100H22.sln* qui se trouve dans le répertoire *\apps\myApps\TP_IFT-3100H22*.
3. La solution devrait s'ouvrir dans votre IDE Visual Studio,
4. Vous n'avez plus qu'à compiler et exécuter le projet.

5 - Architecture

Notre architecture est assez similaire à l'architecture de base qui nous est fourni lorsque l'on crée un nouveau projet dans Openframeworks. Mais elle est divisée afin de faire en sorte que chaque dossier est un élément indépendant dont nous devions faire le rendu. Nous avons une interface publique pour la classe renderer. Ce qui nous permet d'appeler la fonction draw() dans chacune de nos classes avec des paramètres similaires préétablis.

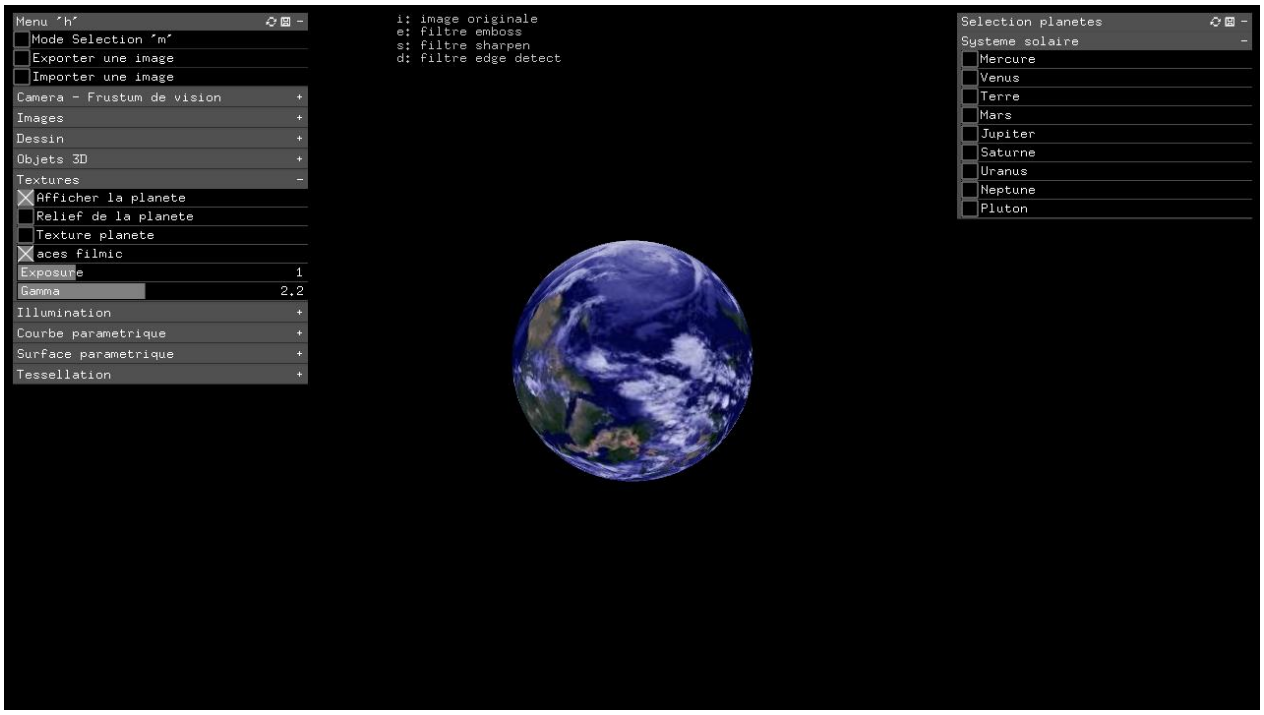


6 - Fonctionnalités

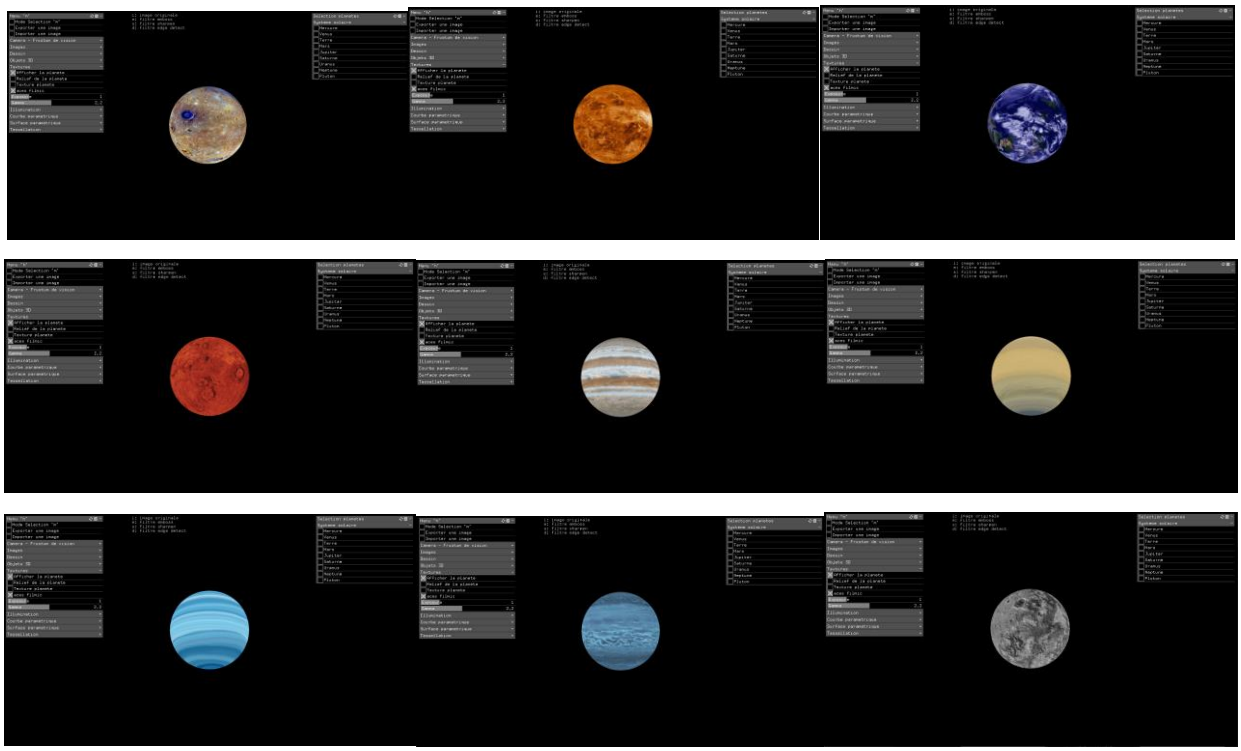
6.1 – Texture (6)

6.1.1 – Coordonnées de texture (6.1)

Nous avons un modèle texturé avec des coordonnées de mapping adéquatement distribuées sur la surface du maillage géométrique. Vous pouvez retrouver cette fonctionnalité dans notre application sous l'onglet *Textures* qui se trouve dans le panel de gauche. Il faut appuyer sur *Afficher la planète* afin de faire le rendu de celle-ci. Il s'agit d'une sphère construite à partir de mesh en utilisant la méthode *ofMesh*. On récupère les dimensions d'une image afin de faire un mapping adéquat sur la sphère.

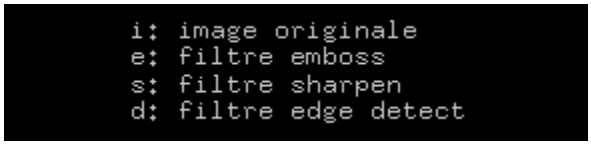


Cette fonctionnalité permet de représenter les neuf planètes qui composent le système solaire. Il y a donc une image différente pour chaque planète qui vient se mapper correctement sur la même sphère de mesh.



6.1.2 – Filtrage (6.2)

L'application permet d'appliquer 3 types d'algorithmes de traitement d'image qui affectent l'apparence des textures utilisées. Cette fonctionnalité est utilisée sur l'élément visuel qu'est la sphère de mesh sur laquelle nous appliquons une texture représentant une planète du système solaire. Nous utilisons un algorithme de convolution d'image telle qu'elle a été présentée dans le cours en la modifiant pour l'appliquer à notre modèle. Les filtres appliqués sont : *emboss, sharpen, edge_detect*. Nous avons un peu modifié les paramètres du filtre *edge detect* afin qu'elle offre plus de luminosité et de contraste. Les commandes afin d'appliquer ces filtres se trouve dans le haut de l'écran de l'application et s'utilisent avec les touches du clavier.



Il est possible d'appliquer ces filtres sur n'importe quelle planète sélectionnée. Voici un exemple avec la planète terre.

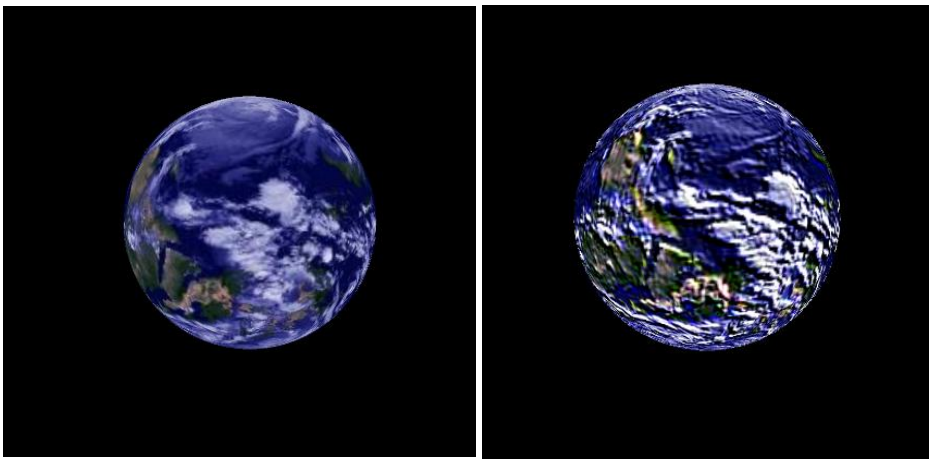
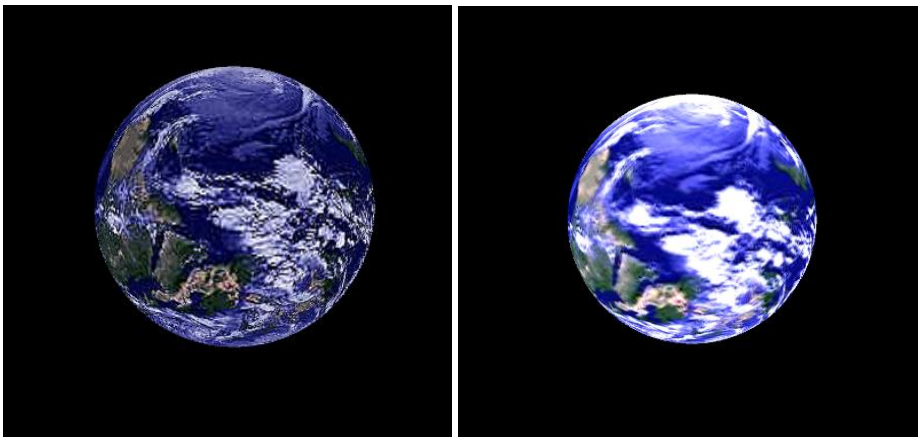


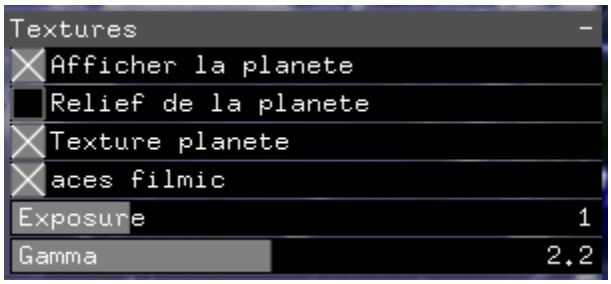
Image original ***Emboss***



Sharpen ***Edge detect***

6.1.3 – Mappage tonal (6.3)

L'application permet d'appliquer un algorithme de mappage tonal sur une image qui est aussi utilisé comme texture pour la planète. Cette fonctionnalité est fortement inspirée des exemples vus dans le cours. Elle se retrouve dans le menu *Textures* de l'application. Pour l'utiliser il faut qu'*Afficher la planete* soit coché et ensuite il faut sélectionner *Texture planete*. L'application affichera l'image qui sert de texture à la planète en deux dimensions et la rendra sur l'entièreté de la zone d'affichage de l'application.

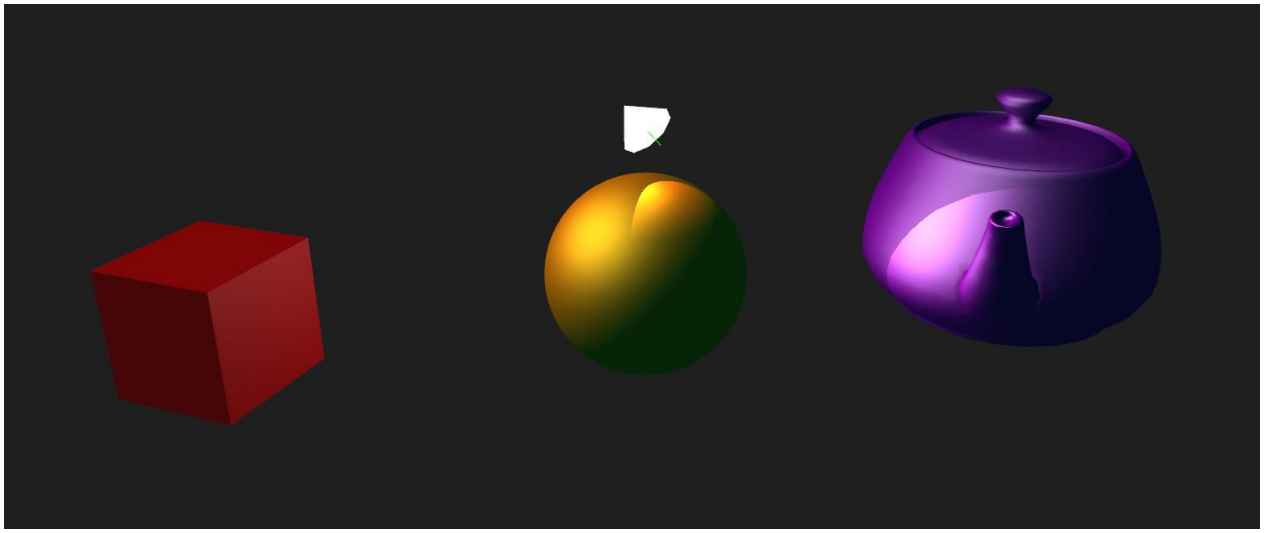


Pour exécuter cette fonctionnalité, nous avons récupéré les shaders de mappage tonal qui ont été présenté dans le cours. L'utilisateur peut donc sélectionner *aces filmic* qui offre un rendu plus lumineux ou *reinhard* qui lui est plus sombre. Il a aussi la possibilité d'ajuster le facteur d'exposition et la correction gamma via les gradateurs correspondants. Cette fonctionnalité peut s'appliquer sur l'ensemble des planètes sélectionné. Il suffit de cocher la planète du système solaire que l'on désire afin de voir un rendu en mappage tonal.

6.2 – Illumination classique (7)

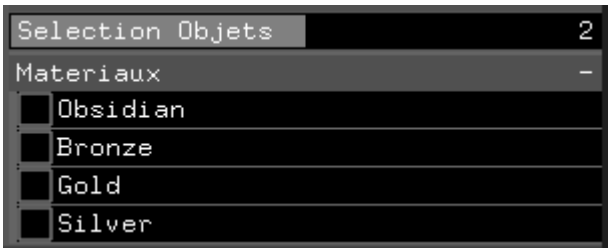
6.2.1 – Modèles d'illumination (7.1)

L'application permet de faire le rendu de 3 éléments visuels fait à partir des modèles d'illumination de Lambert, Gouraud, Phong, Blinn-Phong et le shader par défaut utilisé par OpenFramework. Les 3 éléments visuels sont un cube, une sphère et le teapot 3D partagé dans les exercices du cours.



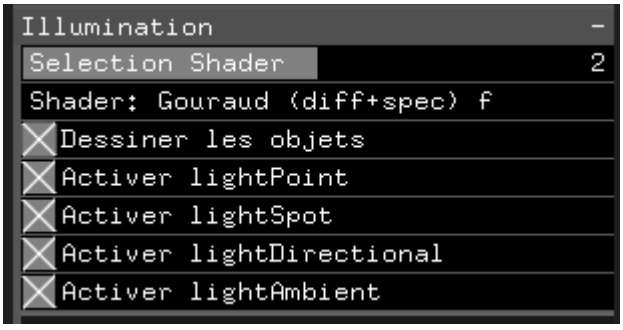
6.2.2 – Matériaux (7.2)

Tous les éléments de la scène peuvent avoir 4 matériaux : Obsidienne, Bronze, Or et Argent. Chaque matériau possède une couleur ambiante, diffuse, spéculaire et émissive et un facteur de brillance spéculaire. On sélectionne l'objet avec la barre de sélection et clique sur le bouton qui correspond au matériau voulu.



6.2.3 – Types de lumière (7.3)

L'application supporte 4 types de lumières avec le modèle d'illumination par défaut et 3 avec tous les autres modèles d'illumination. Les lumières supportées par tous les modèles d'illumination sont la lumière ponctuelle, la lumière directionnelle et la lumière ambiante. La lumière qui est seulement supporté par le modèle d'illumination par défaut est la lumière projecteur. On peut activer et désactiver les lumières désirées dans le menu pour le modèle d'illumination sélectionné.

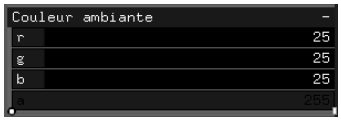


6.2.4 – Lumières multiples (7.4)

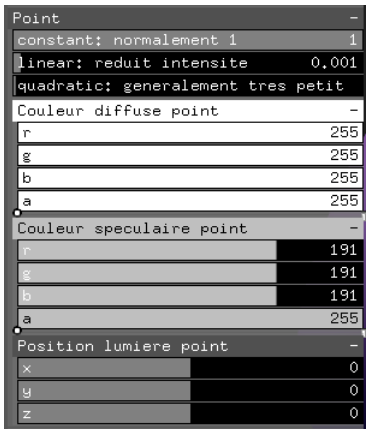
L'application supporte tous les types de lumière qu'il supporte en même temps pour une même scène. Par exemple, le modèle d'illumination de Lambert supporte une lumière directionnelle, ambiante et ponctuelle en même temps et le modèle d'illumination par défaut supporte les 4 types de lumières différentes en même temps. Pour la lumière ambiante, on peut seulement changer la couleur, mais pour tous les autres types de lumière, on peut modifier la position de la

lumière, la couleur diffuse et spéculaire et, pour la lumière ponctuelle et projecteur, on peut contrôler le facteur d'atténuation

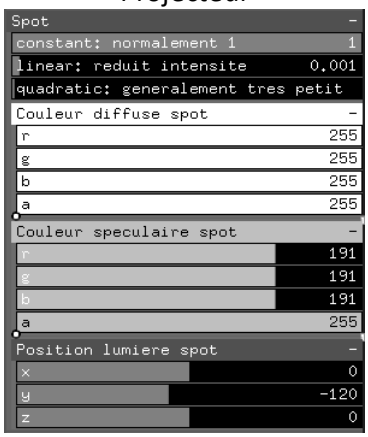
Ambiante



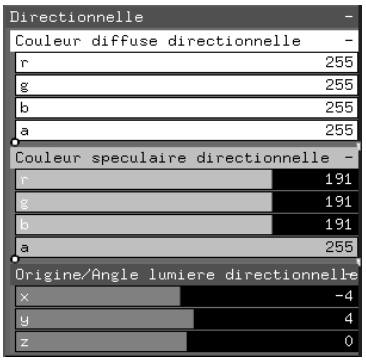
Ponctuelle



Projecteur



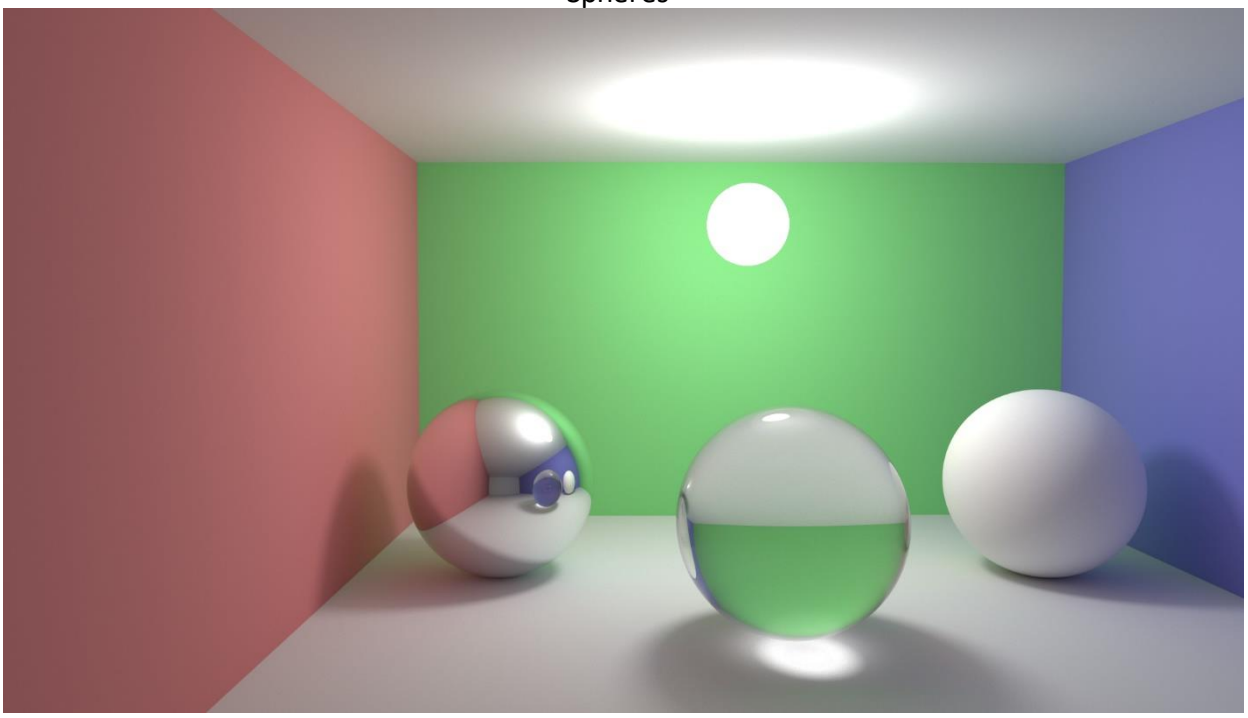
Directionnelle



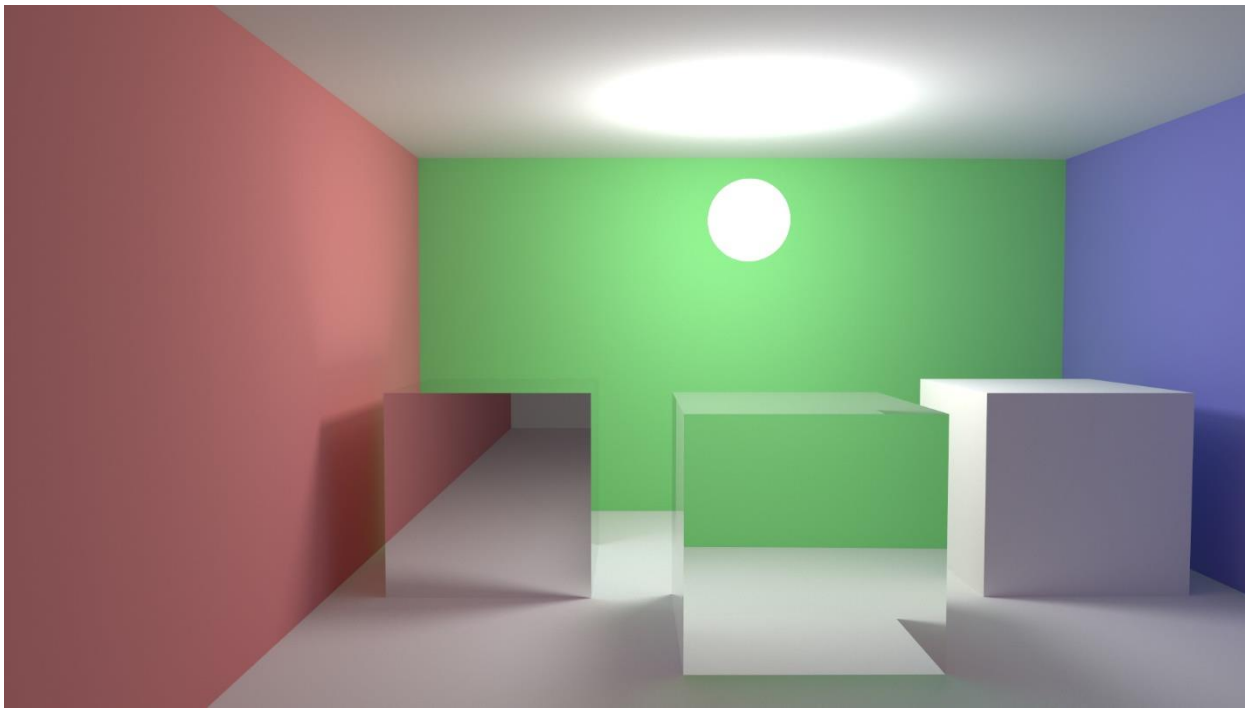
6.3 – Lancer de rayon (8)

La technique de rendu de lancer de rayon utilisé est celle utilisé par l'application originale smallptGPU qui a l'air d'être un Raytracing récursif. Les images suivantes montrent les objets supportés. À gauche est un objet avec réflexion spéculaire, au centre un objet de verre (réfraction) et à droite un objet avec une réflexion diffuse.

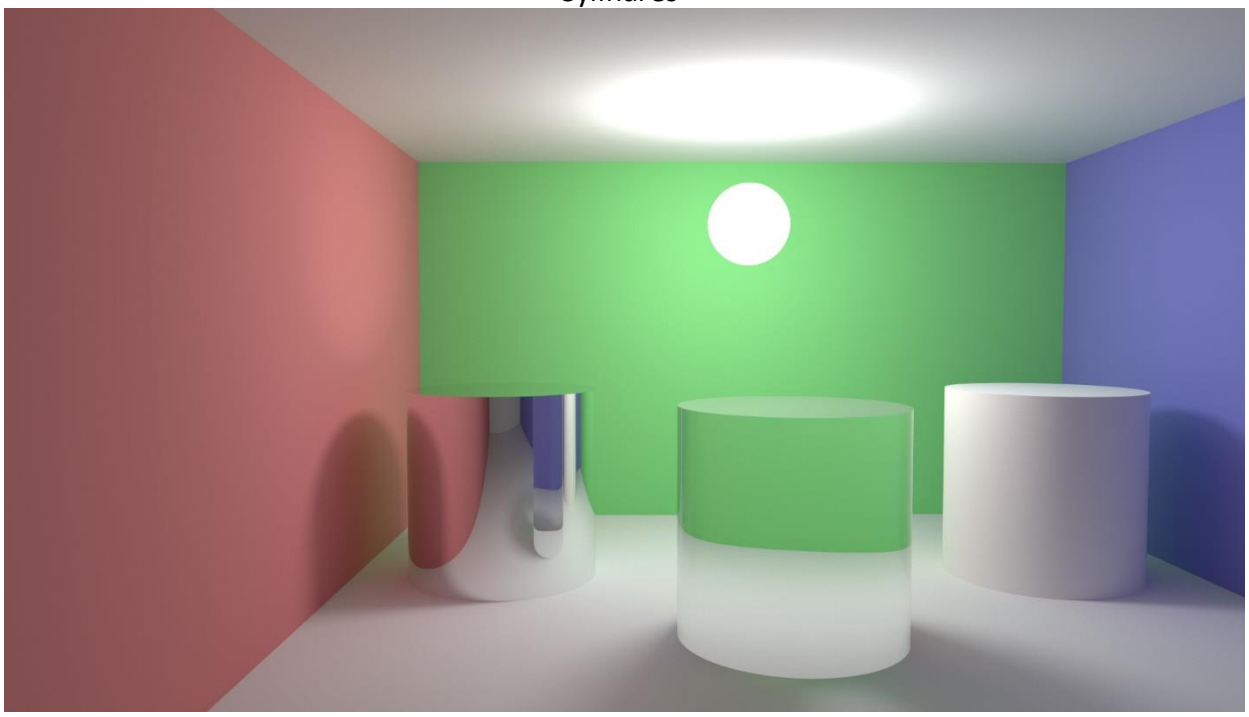
Sphères



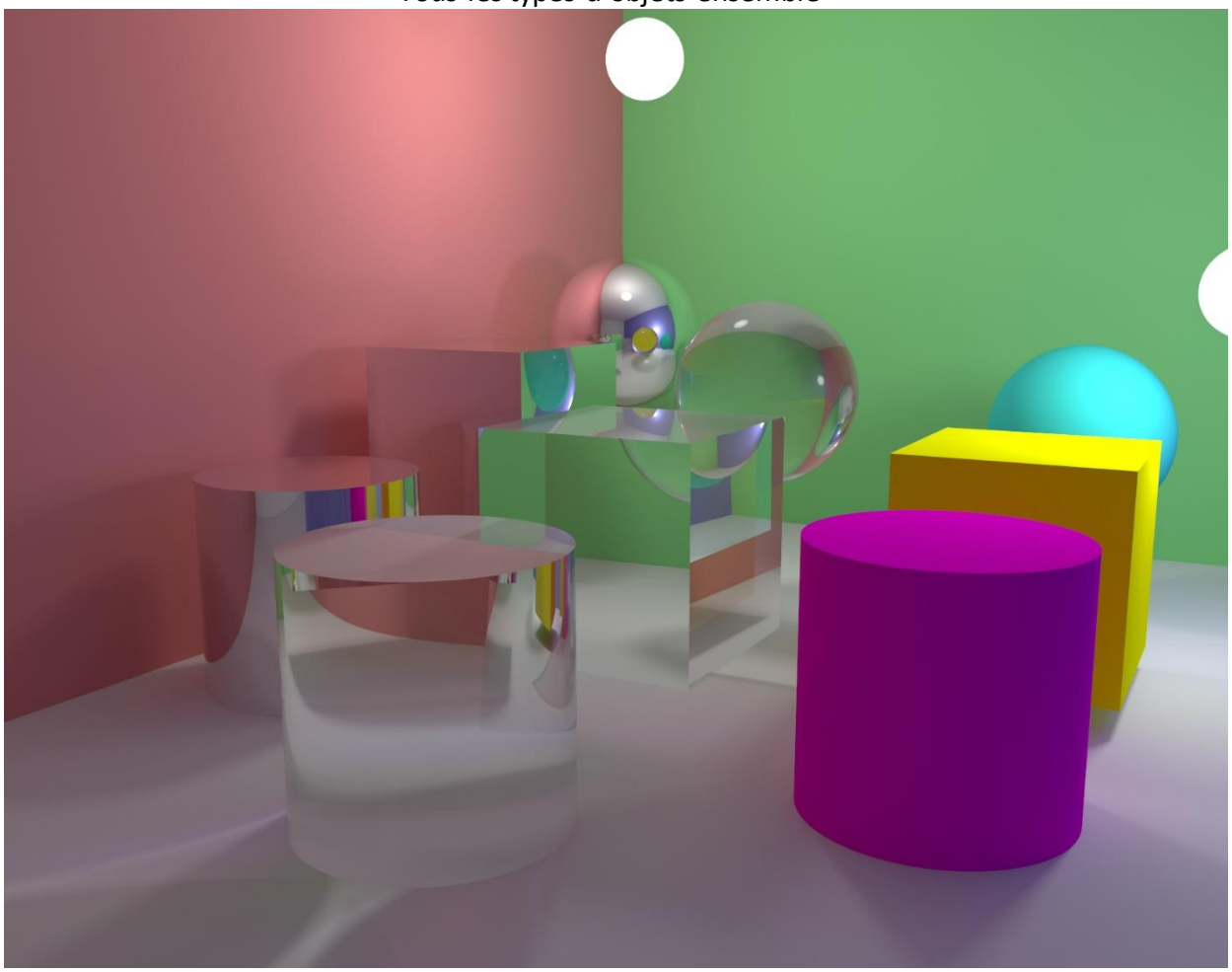
Cubes



Cylindres



Tous les types d'objets ensemble



Pour générer une scène, il suffit de sélectionner une scène et cliquer sur le bouton « générer la scene ». On peut également spécifier le nombre de passes requis avant de générer une image (si le nombre de passe est plus petit ou égal à 10, l'image ne sera pas automatiquement générée). Le nombre de passe est le nombre de fois qu'un rayon qui doit faire le nombre de bonds demandé à été lancé. On peut spécifier le nombre de bonds de chaque rayon et la taille de la fenêtre générée (la taille de l'image qui sera générée est légèrement plus petite à cause de la bordure de contour de la fenêtre Windows et de la bordure du dessus qui contient le titre).

Les fichiers scènes peuvent être édités pour générer une scène différente, les fichiers se doivent se trouver dans le dossier « {racine de l'application exe}/data/scenes/». Les fichiers peuvent seulement être édités, si un fichier est ajouté, il ne sera pas lu par le programme.

La première ligne du fichier spécifie la position de la caméra et dans quelle direction elle regarde. La deuxième ligne est le nombre de primitives dans le fichier et les lignes suivantes sont les primitives. Les primitives possibles sont sphere, cube, cylinder.

Les valeurs des primitives sont, en groupes de 3 (x,y,z) séparés par 2 espaces. Le premier groupe contient des valeurs qui changent selon la primitive : rayon/largeur, hauteur, longueur. Le deuxième groupe spécifie la position au centre de la primitive. Le troisième groupe spécifie l'émission. Le quatrième groupe spécifie la couleur entre 0 et 1 et la dernière valeur séparée par 2 espaces spécifie le type du matériau : 0 = diffuse, 1 = speculaire (métallique), 2 = réfraction (verre).

6.3.1 – Intersection (8.1)

L'application est capable de calculer le point d'intersection entre un rayon et un cube, une sphère et un cylindre. La méthode d'intersection a été modifiée pour calculer l'intersection selon le type d'objet et pour également calculer et retourner la normale de la surface frappée. Cette modification permet de ne pas avoir à modifier le reste du code puisqu'il n'a pas besoin de savoir quelle forme d'objet il est en train de traiter.

6.3.2 – Réflexion (8.2)

Les cubes, sphères et cylindres supportent les surfaces avec une réflexion. La technique de réflexion utilisée est une technique équivalente à la fonction *reflect* en GLSL qui était déjà présente dans le code original de l'application.

6.3.3 – Réfraction (8.3)

Les cubes, sphères et cylindres supportent les surfaces avec une réfraction. La technique de réfraction utilisée est une réfraction de surface diélectrique. Nous supportons la réflexion totale interne qui vérifie si le rayon est à l'intérieur de l'objet ou non et s'il possède assez d'énergie pour soit sortir de l'objet ou rentrer dans l'objet. S'il n'a pas assez d'énergie le rayon va seulement être

reflété. Nous supportons également l'effet de Fresnel. Toutes ces méthodes de réfraction étaient déjà présentes dans le code original de l'application.

6.3.4 – Ombrage (8.4)

L'ombrage est atteint grâce au fait que chaque rayon lancé qui frappe une surface a une partie de sa lumière absorbée et un nouveau rayon est lancé avec le restant de la lumière. La méthode pour calculer l'ombrage était déjà présente dans le code original de l'application.

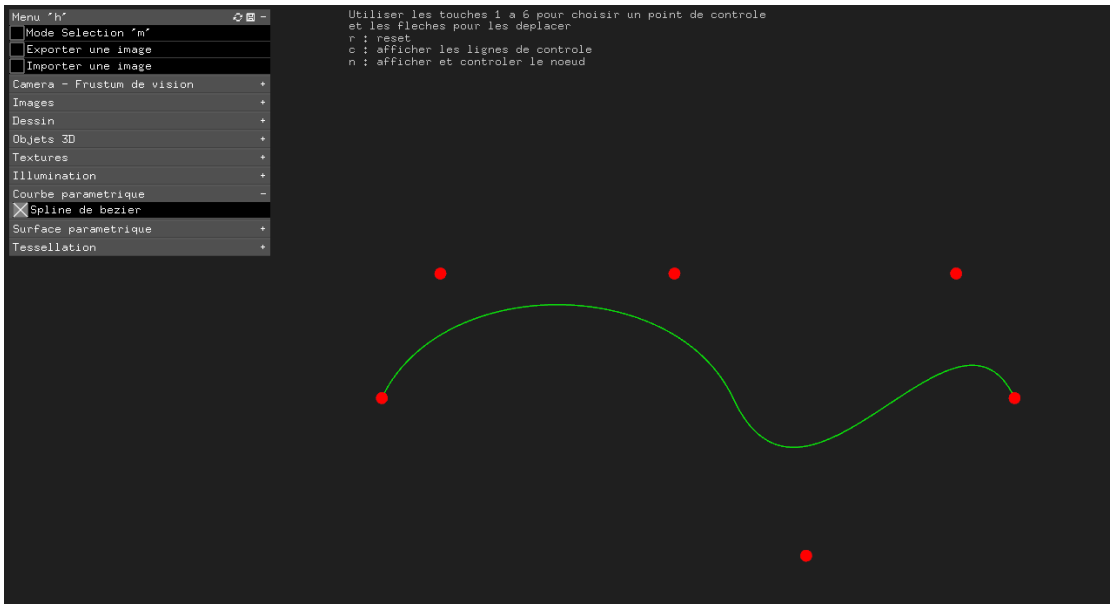
6.3.5 – Illumination globale (8.5)

L'illumination globale est atteinte grâce à un principe de radiosité, un rayon frappe une surface et une partie de sa lumière est absorbée par la surface et sa couleur change en fonction de la couleur de la surface frappée et un nouveau rayon est lancé avec la nouvelle couleur mixée dans une nouvelle direction calculée en fonction de l'angle du rayon. La méthode pour calculer l'illumination globale était déjà présente dans le code original de l'application.

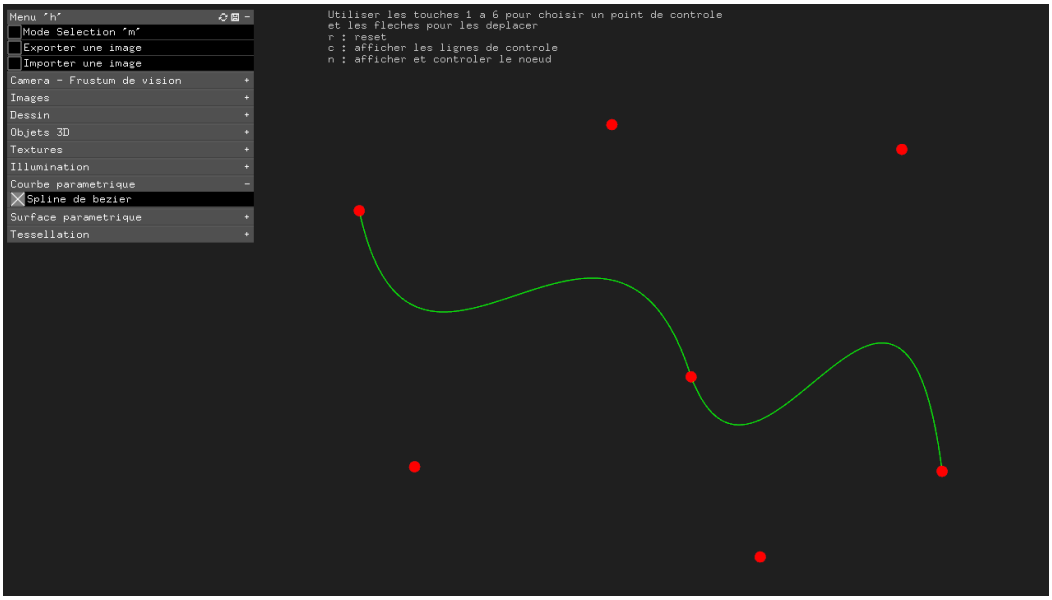
6.4 – Topologie (9)

6.4.1 – Courbe paramétrique (9.1)

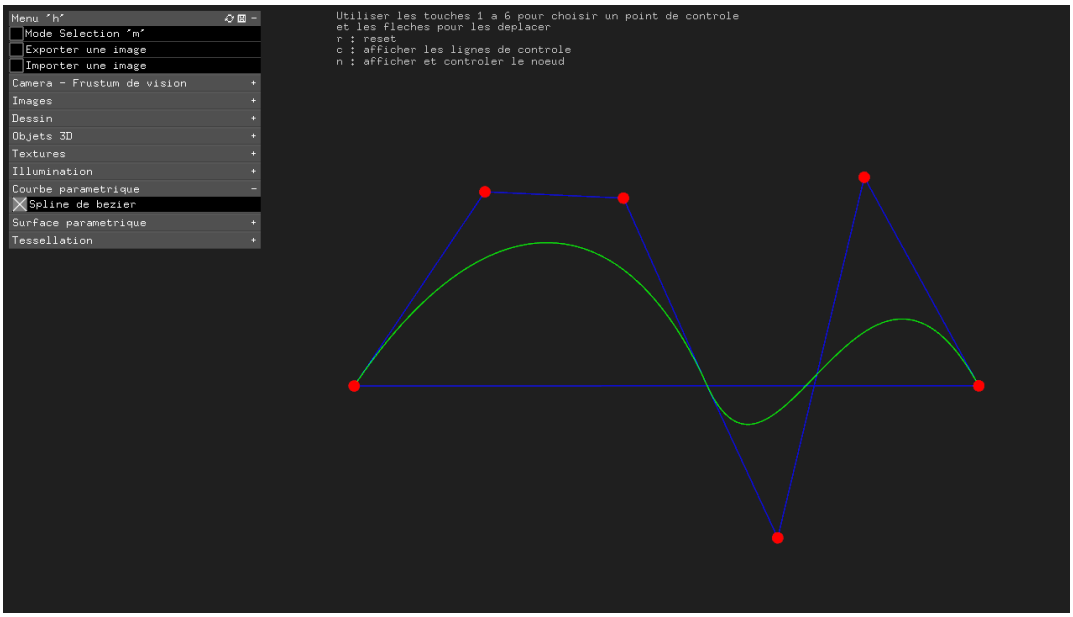
Il est possible de rendre un type de courbe paramétrique avec plus de 4 points de contrôle. Il s'agit d'une courbe de Bézier cubique, plus particulièrement un B-spline, qui est une combinaison linéaire de deux courbes de Bézier cubique. Afin d'activer cette fonctionnalité il faut aller sélectionner *Spline de bezier* sous l'onglet *Courbe parametrique* dans le menu de gauche de notre application. Des instructions sont offertes à l'utilisateur dans le haut de l'écran afin de lui permettre de manipuler la courbe.



L'utilisateur peut régler 7 points de contrôle de la courbe en incluant le nœud de la courbe de Bézier. Il a aussi la possibilité d'afficher ou de cacher ce nœud afin de rendre la courbe plus agréable à l'œil.



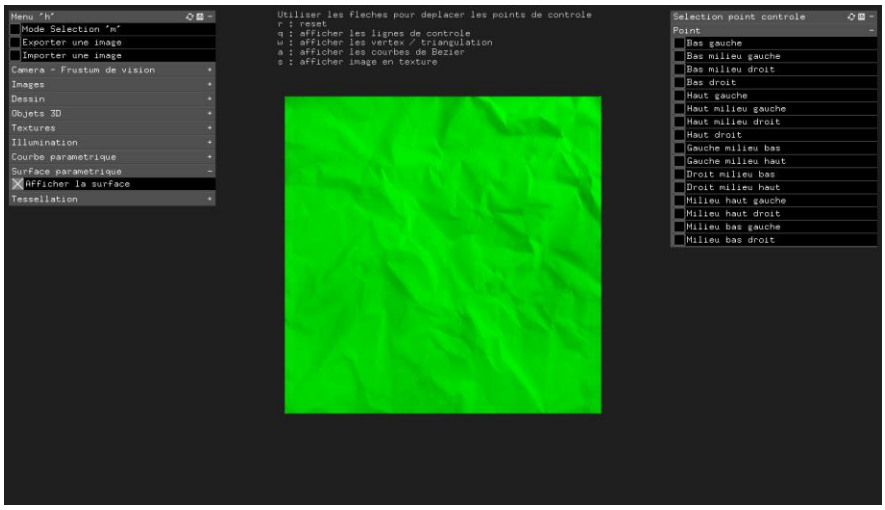
L'utilisateur a aussi la possibilité d'afficher les lignes de contrôle de la courbe paramétrique.



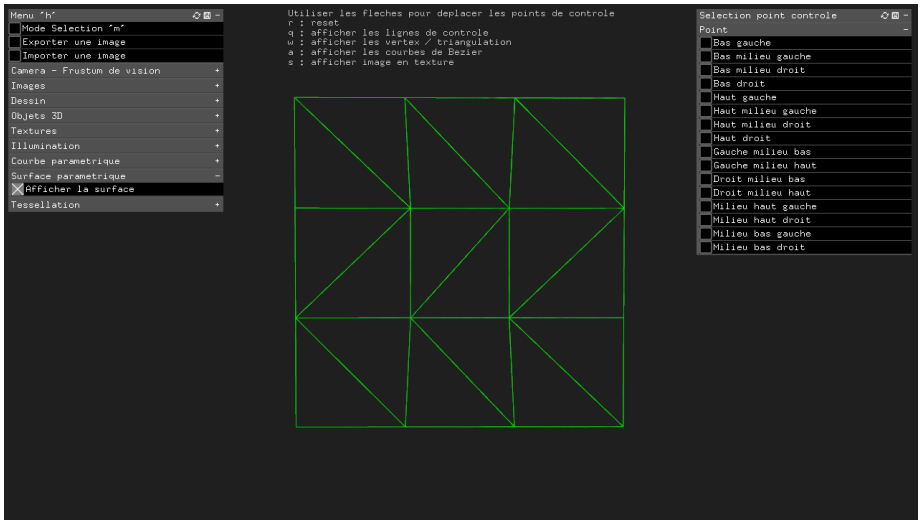
Afin d'implémenter cette fonctionnalité, nous avons utilisé la fonction d'évaluation d'une courbe de Bézier cubique qui se retrouve dans les exemples du cours. Nous l'avons ensuite extrapolé afin de pouvoir rendre un B-spline en utilisant deux courbes de Bézier cubique. La méthode *ofPolyline* nous permet d'affecter la position points qui forment la courbe.

6.4.2 – Surface paramétrique (9.2)

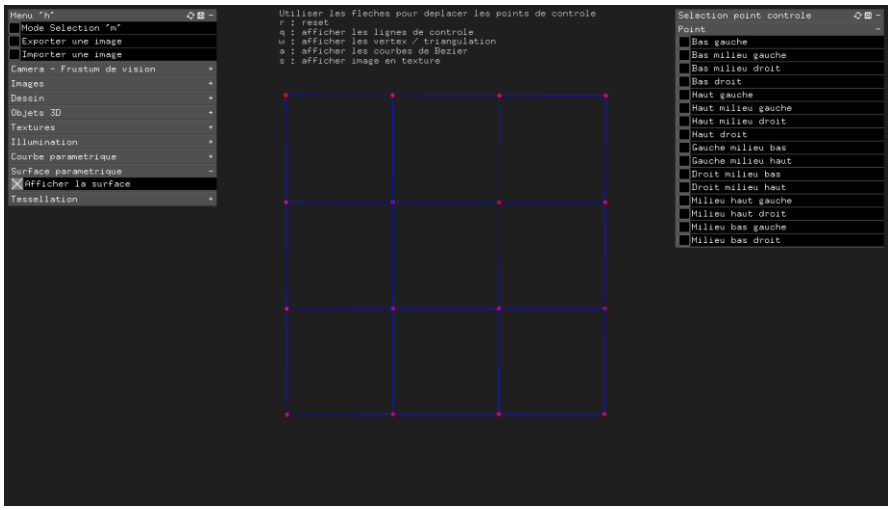
Il est possible de rendre une surface paramétrique. Il s'agit d'une surface de Bézier bicubique. Afin d'utiliser cette fonctionnalité il faut sélectionner *Surface de bezier* sous l'onglet *Surface parametrique* dans le menu de gauche de notre application. Des instructions sont offertes à l'utilisateur dans le haut de l'écran afin de lui permettre de manipuler la surface. Un menu est aussi offert à droite de l'écran afin de sélectionner le bon point de contrôle à manipuler.



Tout d'abord, pour construire la surface paramétrique nous avons créé une surface de type *plane* à l'aide de *ofMesh*. Cette méthode nous a permis d'effectuer une triangulation qui sert de base à notre surface. On peut voir le résultat dans l'image ci-dessous.

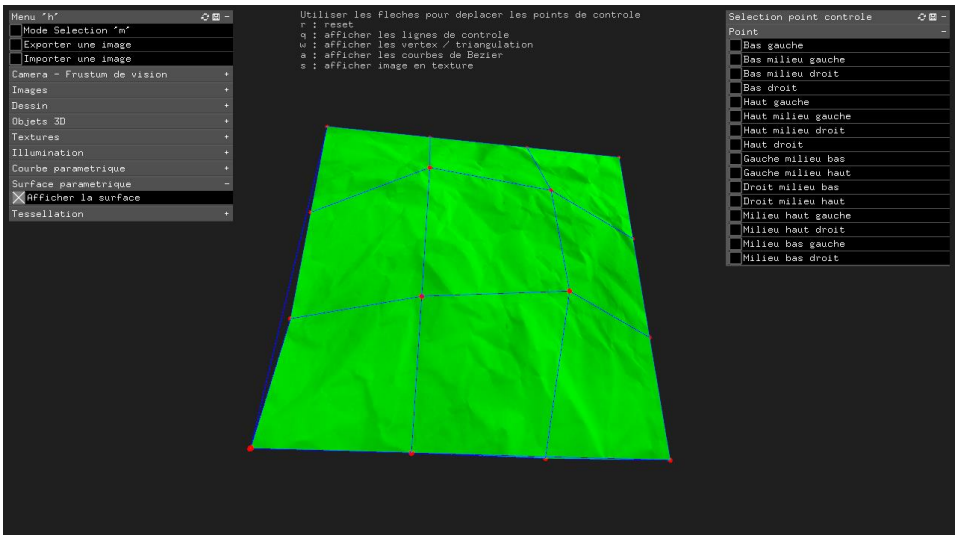


Ensuite, nous avons construit la structure de la surface paramétrique qui est constitué de huit courbes de Bézier cubique que nous avons superposé à la structure de mesh.



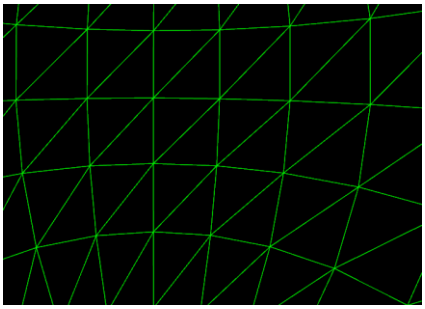
Chaque courbe de Bézier utilise la méthode *ofPolyline* qui nous permet de construire une ligne selon la position de certains points. Nous avons donc fixé les vertex de notre structure de mesh de départ au bon endroit sur les courbes de Bézier afin de pouvoir rendre cette surface paramétrique.

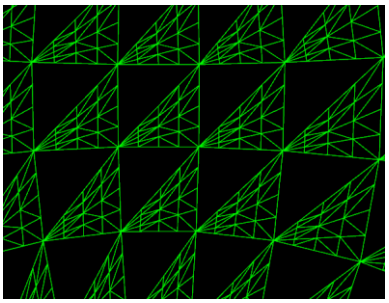
Finalement, nous avons ajouté une texture sur le plan de mesh afin d'ajouter à l'effet de surface. Il suffit donc de sélectionner un des 16 points de contrôle afin de paramétrer la surface.



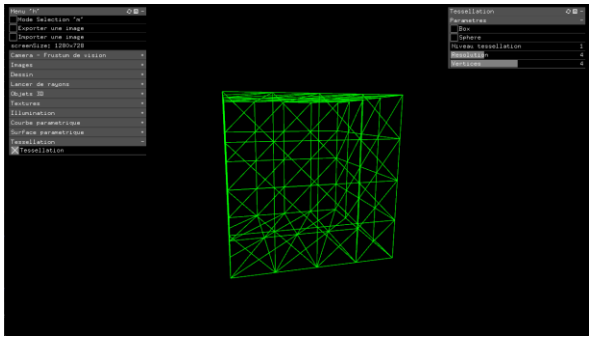
6.4.3 – Shader de tessellation (9.3)

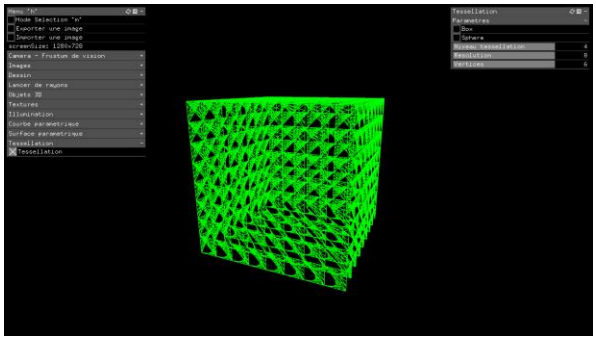
Un shader de tessellation permet de sous-diviser la géométrie de deux modèles. Cette fonctionnalité a été implémenté à l'aide de cinq shaders. La source de ces shaders qui nous a servie de base est indiqué dans la section ressources du document. Nous avons un shader de vertex et un shader de fragment qui s'occupent principalement de gérer les positions et les couleurs. Le shader de contrôle détermine le niveau de tessellation que l'on veut obtenir à l'aide des méthodes *gl_TessLeveInner[]* et *gl_TessLevelOuter[]*. Le shader d'évaluation s'occupe de calculer les coordonnées et les positions afin de déterminer la forme que la tessellation prendra. Le shader de géométrie s'occupe de fermer les triangles et de rendre la forme. C'est ainsi que nous pouvons obtenir un résultat comme celui- ci.

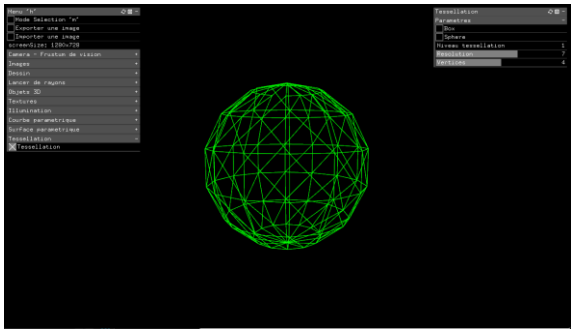


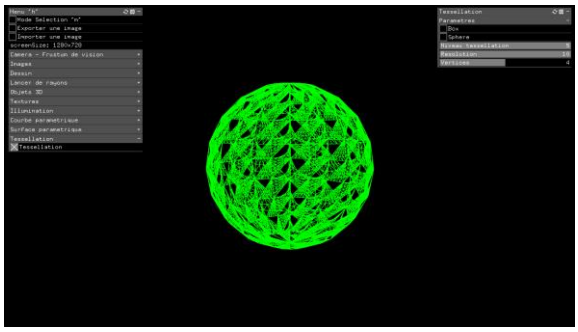


Nous avons implémenté la tessellation sur des objets de mesh à l'aide de *ofPrimitive*. Afin d'activer cette fonctionnalité, il suffit de cliquer sur tessellation dans le menu de gauche de l'application. Un second menu est disponible à droite afin de choisir l'objet sur lequel on veut appliquer la tessellation. L'utilisateur a l'option entre un Box ou une Sphère. Il peut paramétrer le niveau de tessellation, la résolution et le nombre de vertices de l'objet.





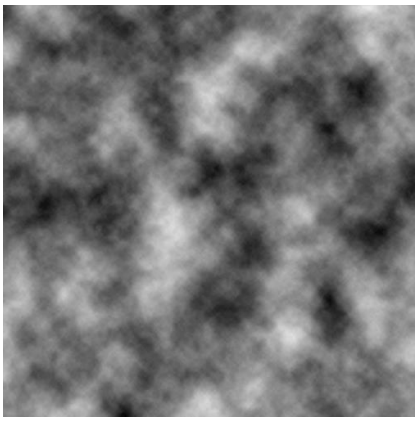


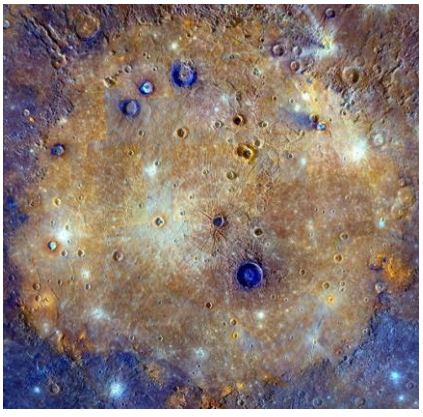


6.4.4 – Effet de relief (9.5)

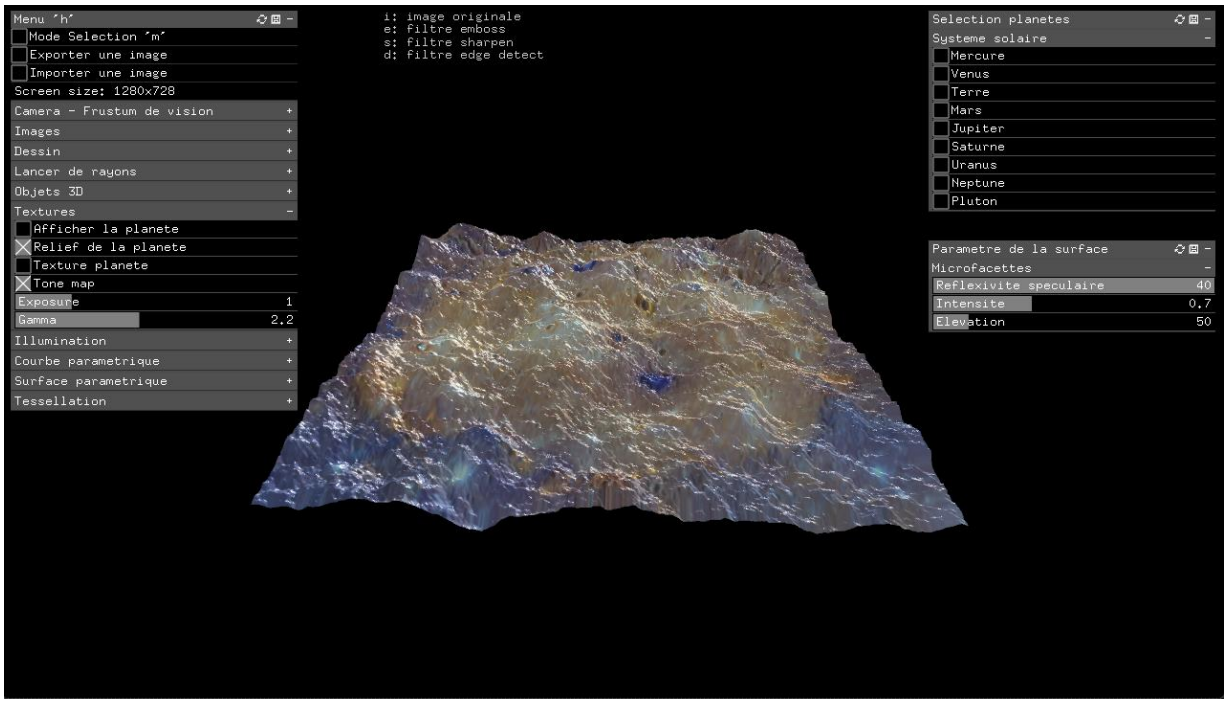
Il est possible de rendre un modèle ou des textures sont utilisées pour simuler un effet de relief sur la surface. Afin de réaliser cette fonctionnalité nous avons démarré à partir d'un exemple trouver sur le forum d'OpenFrameworks. La source des shaders qui rendent cette fonctionnalité possible se retrouve dans la bibliographie. Cette fonctionnalité s'ajoute donc à la catégorie *Textures* de notre application et permet de visualiser le relief des différentes planètes du système solaire. Pour l'activer il suffit de cocher la case *Relief de la planete* qui se situe dans le menu de gauche de l'application.

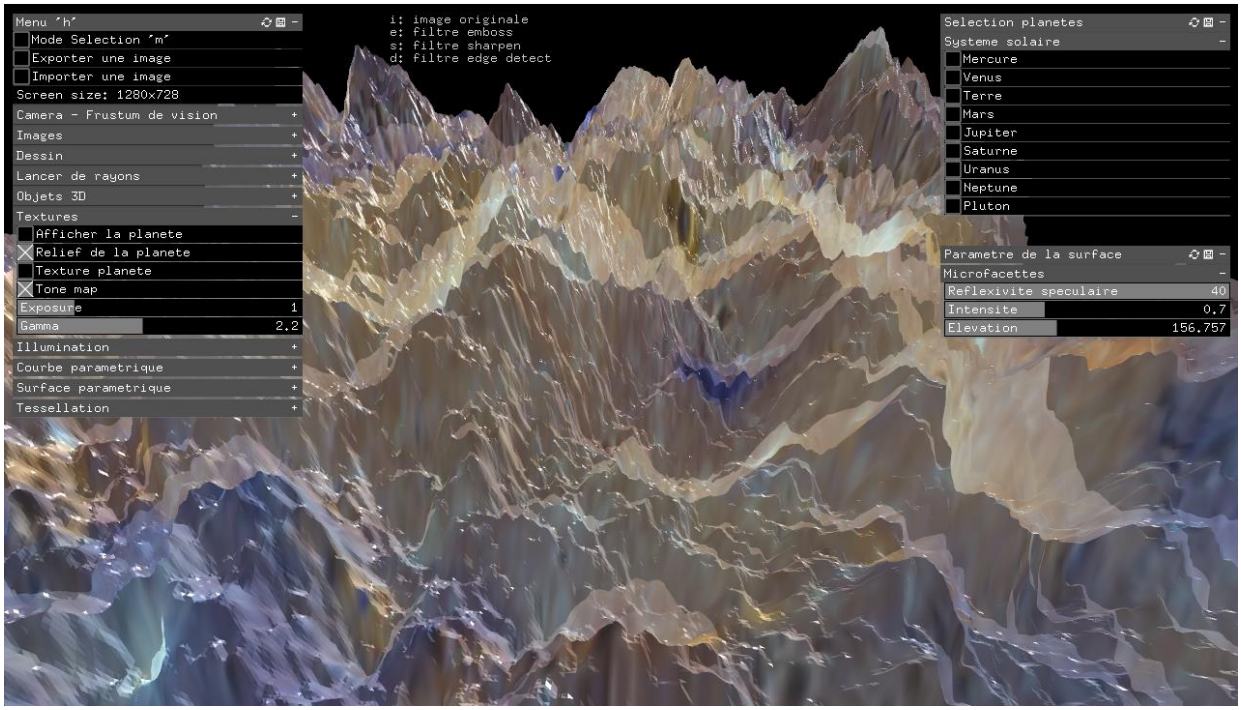
Pour rendre cet effet, il nous faut d'abord effectuer un normal mapping sur l'image de gauche ci-dessous qui contient plusieurs niveaux de gris et on l'applique sur l'image de droite.





Le normal mapping est appliqué sur un *plane* via la méthode *ofPlanePrimitive*. Les données sont d'abord initialisées dans le code de l'application et ensuite passé au shader de vertex et de fragment qui effectue le travail de normal map. On pourra par la suite apercevoir l'effet de relief qui est assez convainquant. Il est possible de paramétrer le gradateur *Intensite*, qui se retrouve dans le sou menu de droite, afin de faire varier la position en X des normales, ce qui permet d'avoir un relief plus abrupt.





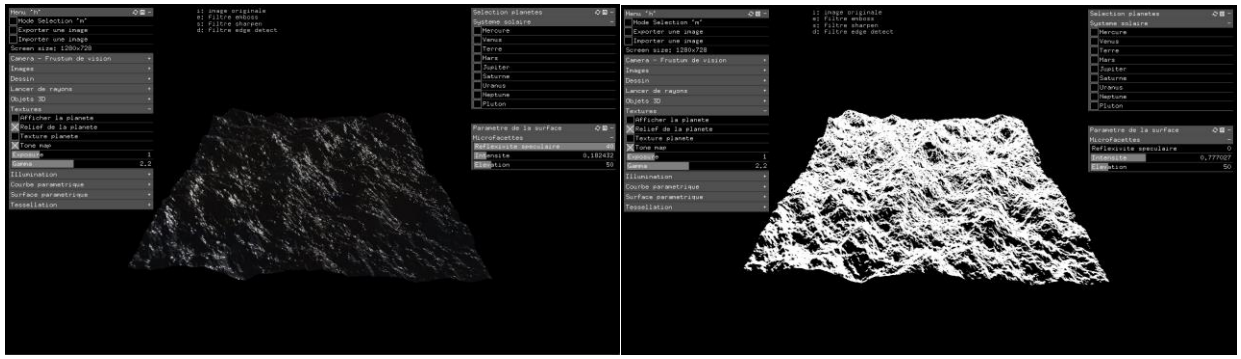
6.5 – Micro-facettes (10.4)

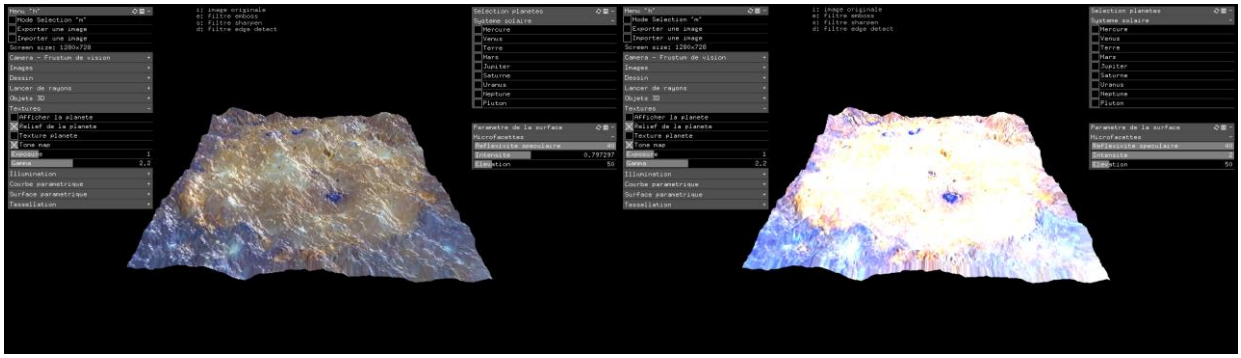
On peut observer une distribution de micro-facettes pour simuler la rugosité du matériau. Cette fonctionnalité fait suite à la précédente qui rend l'effet de relief. Le shader avec lequel nous avons démarré, trouvé sur le forum d'OpenFrameworks, contenait toutes les données nécessaires à la réalisation de cette fonctionnalité. Nous l'avons modifié pour correspondre à nos besoin. La source des shaders qui rendent cette fonctionnalité possible se retrouve dans la bibliographie.

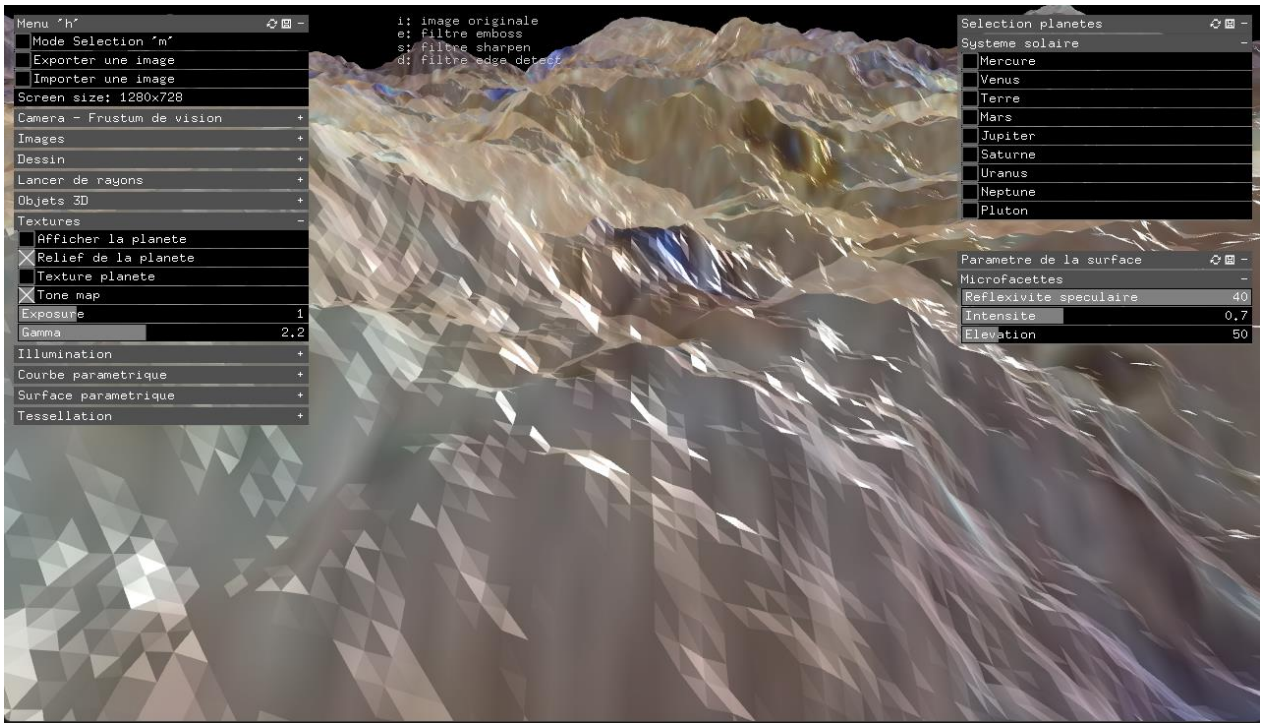
Les données sont d'abord initialisées dans le code. On récupère en autre la position de la souris qui servira de position de la lumière. Celle-ci sera ensuite passé au shaders de normal mapping afin d'effectuer une illumination spéculaire. On pourra apercevoir un phénomène d'occlusion simuler et un effet de micro ombrage sur la surface et sur les micro-facettes de la surface qui est un *plane* construit avec *ofPlanePrimitive*. Voici comment s'effectue les calculs dans le shaders de fragments :

```
uniform vec3 lightpos;
uniform float specularness;
uniform float diffuseness;
out vec4 outputColor;

void main( ) {
    vec3 N = normalize(cross(dFdy(vpos), dFdx(vpos)));
    vec3 V = normalize(vertexnormal);
    vec3 R = reflect(V, N);
    vec3 L = normalize(vec3(lightpos.x, lightpos.y, lightpos.y));

    vec4 texture = texture( colorTex, vertexTexCoord ) * diffuseness;
    vec4 ambient = vec4(0.2, 0.2, 0.2, 0.0);
    vec4 diffuse = vec4(0.2, 0.2, 0.2, 0.5) * max(dot(L, N), 0.0);
    vec4 specular = vec4(1.0, 1.0, 1.0, 0.8) * pow(max(dot(R, L), 0.0), specularness);
    outputColor = texture + ambient + diffuse + specular;
}
```

Par la suite on pourra modifier certains paramètres directement dans l'application via des gradateurs afin de moduler l'apparence de la surface.






Menu 'h'
Mode Selection 'm'
Exporter une image
Importer une image
Screen size: 1280x728
Camera - Frustum de vision
Images
Dessin
Lancer de rayons
Objets 3D
Textures
Afficher la planete
Relief de la planete
Texture planete
Tone map
Exposure 1
Gamma 2,2
Illumination
Courbe parametrique
Surface parametrique
Tessellation
i: image originale
e: filtre emboss
s: filtre sharpen
d: filtre edge detect
Selection planetes
Systeme solaire
Mercure
Venus
Terre
Mars
Jupiter
Saturne
Uranus
Neptune
Pluton
Parametre de la surface
Microfacettes
Reflexivite speculaire 40
Intensite 0,7
Elevation 50

7 - Ressources

GitHub:

https://github.com/philvoyer/IFT3100H22

https://github.com/chimanaco/ofGeometryShaderExamples

https://github.com/uwe-creative-technology/CT_toolkit_sessions

https://github.com/leozimmerman/ShadersLibrary

https://github.com/iceman201/RayTracing/blob/master/Ray%20tracing/Cylinder.cpp#L20

Sites web:

https://openframeworks.cc/documentation/

https://learnopengl.com/Getting-started/Shaders

https://learnopengl.com/Lighting/Light-casters

https://forum.openframeworks.cc/t/specular-mapping/22400

https://stackoverflow.com/questions/5281261/generating-a-normal-map-from-a-height-map

https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection

https://blog.johnnovak.net/2016/10/22/the-nim-ray-tracer-project-part-4-calculating-box-normals/

https://gamedev.stackexchange.com/questions/191328/how-to-perform-a-fast-ray-cylinder-intersection-test

https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html

http://www.kevinbeason.com/smallpt/

http://davibu.interfree.it/opencl/smallptgpu2/smallptGPU2.html

https://developer.nvidia.com/opencl

Images:

https://besthqwallpapers.com/textures/crumpled-paper-texture-white-crumpled-paper-background-paper-texture-white-paper-crumpled-paper-131988

https://web.cortland.edu/flteach/civ/davidweb/resources.htm

https://www.istockphoto.com/search/2/image?phrase=mars+texture

https://nasa3d.arc.nasa.gov/detail/ven0aaa2

https://www.planetary.org/articles/0514-juno-meets-cassini-a-new

https://www.deviantart.com/uxmal750ad/art/Saturn-Planetary-Texture-Stock-Image-509267806

https://nasa3d.arc.nasa.gov/detail/nep0fds1

8 - Présentation

8.1 - Stéphane Boulanger

Après avoir complété un Baccalauréat dans le domaine des sciences sociales, je suis maintenant étudiant au certificat en informatique. Je suis un passionné de code et des nouvelles technologies. Tout au long de mon parcours, j'ai choisi de me concentrer sur la programmation orientée objet, le développement logiciel et la programmation de jeux vidéo.

Dans ce projet j'ai fait les fonctionnalités de coordonnées de texture, filtrage, mappage tonal, courbe paramétrique, surface paramétrique, shader de tessellation, effet de relief et micro-facettes.

8.2 -William Boudreault

J'ai fait un DEC-BACC au Cégep Garneau en informatique et je suis ensuite allé à l'université Laval pour compléter mon Baccalauréat en informatique. J'ai beaucoup aimé les cours de conception et analyse d'algorithmes pour maximiser les performances et, maintenant, ce cours d'infographie fait également partie de mes favoris.

Dans ce projet j'ai fait le module 7 et le module 8 : Modèles d'illumination, matériaux, types de lumières, lumières multiples, intersection, réflexion, réfraction, ombrage et illumination globale.