IFT-3100 Travail Pratique 3

Visualiseur interactif de scènes 3D
Document de design

Travail présenté à M. Philippe Voyer
le 23 avril 2017

Nathalie Chang 111 159 879

Table of Contents

1. Sommaire

L'application développée pour ce projet permet à un usager de créer et d'éditer une scène 3D. Une interface simple et facile d'utilisation combine des menus avec boutons de type « sliders » et cases à cocher, ainsi que l'utilisation de touches du clavier permettent à l'usager de développer une scène et de contrôler les paramètres des entités de la scène. L'application permet de créer une grande variété de primitives, d'appliquer des matériaux à ces primitives, d'ajouter des sources lumineuses et d'utiliser divers modèles d'illumination.

Dès le lancement de l'application, l'usager est appelé à sélectionné la version OpenGL/GLSL qu'il souhaite utiliser. Par la suite, un menu de création de diverses primitives est présenté à l'usager, qui offre entre autres, des primitives 2D et 3D, diverses courbes paramétriques, une surface paramétrique, diverses types de sources lumineuses, et aussi l'option de créer un nuage de points avec une texture et un effet visuel de particule. L'usager peut aussi importer des modèles 3D de format .obj. L'importation d'images en tant qu'arrière-plan, ou l'importation de texture pour le nuage de points est aussi possible.

L'application permet de sélectionner des objets grâce à un simple clique de la souris, et d'éditer de nombreux paramètres pour ces objets. L'espace de couleur peut-être en HSB ou en RGB. Les paramètres de dessin ainsi que de transformation peuvent être modifiés lorsqu'un objet est sélectionné. La modification de la position d'un objet peut se faire à l'aide de coordonnées cartésiennes ou de coordonnées polaires. Un lien de parenté peut être établi entre des entités, ce qui signifie qu'elles sont regroupées sous une forme et peuvent être déplacées ensemble. Les paramètres du matériau de la primitive peuvent être modifiés et divers types de shaders sont offerts pour calculer l'illumination de la primitive. Les paramètres des sources lumineuses sont aussi modifiables. Parmi les shaders, il y aussi un shader de géométrie qui est offert. Ce dernier crée une deuxième primitive à partir d'une première, la déplace et applique un facteur d'agrandissement de deux.

L'application offre une caméra qui peut tourner et se déplacer autour d'une entité. Il est possible d'afficher le menu des paramètres de la caméra en tout temps et de centrer la caméra sur l'entité choisie de la scène. Les modes de projection orthogonale et perspective sont disponibles.

Un mode de lancer de rayon est offert à l'usager pour démontrer un mode d'illumination très réaliste pour une scène comprenant des cubes et des sphères.

L'application permet aussi d'appliquer une texture diffuse sur des sphères et des cubes, et d'appliquer une normal map pour des cubes.

Plusieurs effets plein écran sont aussi disponibles, tel que des filtres de Motion Blur, de Sharpen, et deux filtres avec différents niveaux d'Emboss.

2. Interactivité

L'application qui a été développée pour ce projet est hautement interactive et fournit à l'usager une interface conviviale pour la création d'une scène 3D.

Des instructions fournissant une description des touches à utiliser pour effectuer diverses fonctions est présentée en tout temps dans le haut de l'écran. Les contrôles sont les suivants :

Menu des fonctions

L: Importer un fichier pour l'arriere-plan
La touche « L » ouvre une fenêtre Windows qui permet la sélection d'une image qui sera affichée en tant qu'arrière-plan de la scène.

X: Exporter l'image en cours
La touche « X » permet de d'exporter une image de la scène. Le fichier est sauvegardé dans le répertoire courant.

P: Ajouter un parent (Tenir P + selectionner parent)
Afin de créé un lien de parenté entre deux entités, il suffit de sélectionner une entité dans la scène en cliquant avec le bouton de souris de droite. Ensuite le bouton de souris doit être relâché et l'on doit tenir la touche « P », et par la suite il faut sélectionner le parent avec le bouton de souris droite. Une flèche apparaît démontrant le lien et la direction vers le parent.

C: Modifier parametres de la camera
La touche « C » permet d'afficher le menu de contrôle de la caméra, qui permet de modifier les paramètres suivants : angles des champs de vision (horizontal et vertical), le ratio d'aspect et les distances du plan de clipping avant et arrière.

F: Centrer la camera sur l'objet selectionne ("focus")
La touche « F » permet de déplacer la caméra et de la centrer sur une entité qui est sélectionnée. Cette entité devient la cible de la caméra.

W/A/S/D: Tourner la camera autour de l'entite seletionnee ("orbit")
Les touches « W », « A », « S », et « D » permettent de tourner la caméra autour d'une cible, vers le haut et le bas, vers la gauche et la droite.

Q/E: Rapprocher ou eloigner la camera ("dolly")
Les touches « Q » et « E » permettent de rapprocher ou d'éloigner la caméra d'une cible.

R: Activer/Désactiver le mode lancer de rayon
La touche « R » permet d'entrer le mode lancer de rayon. Afin d'en sortir, il s'agit simplement de faire la touche de nouveau.

F1: Afficher ou cacher l'interface graphique
La touche « F1 » permet de cacher les menus et l'interface graphique.

F10: Recharger tous les shaders

La touche « F10 » permet de recharger tous les shaders, qui a été très utilisé lors du développement des shaders.

À noter : Dans le cas des primitives requérant de nombreuses touches de contrôles, telles que les courbes cubiques, les courbes paramétriques Catmull-Rom et les surfaces bézier, les consignes sont ajoutées au menu des fonctions en jaune, afin de guider l'usager.

De plus, divers menus sont mis à la disposition de l'usager à la gauche et à la droite de l'écran. Les menus offerts de façon permanente sont le menu des options globales, et le menu de création d'entités. Le menu intitulé « Global Options » permet la modification de la couleur d'arrière-plan, l'activation de la lumière accompagnant la caméra, la modification de la lumière ambiante, et la sélection d'un filtre plein écran. De même, le menu de création de primitive présente des cases à cocher permettant de sélectionner le type de primitive à créer. Après avoir fait un choix, l'usager doit cliquer avec **le bouton de souris gauche** dans l'espace de la scène et l'entité apparaît à l'endroit désiré.

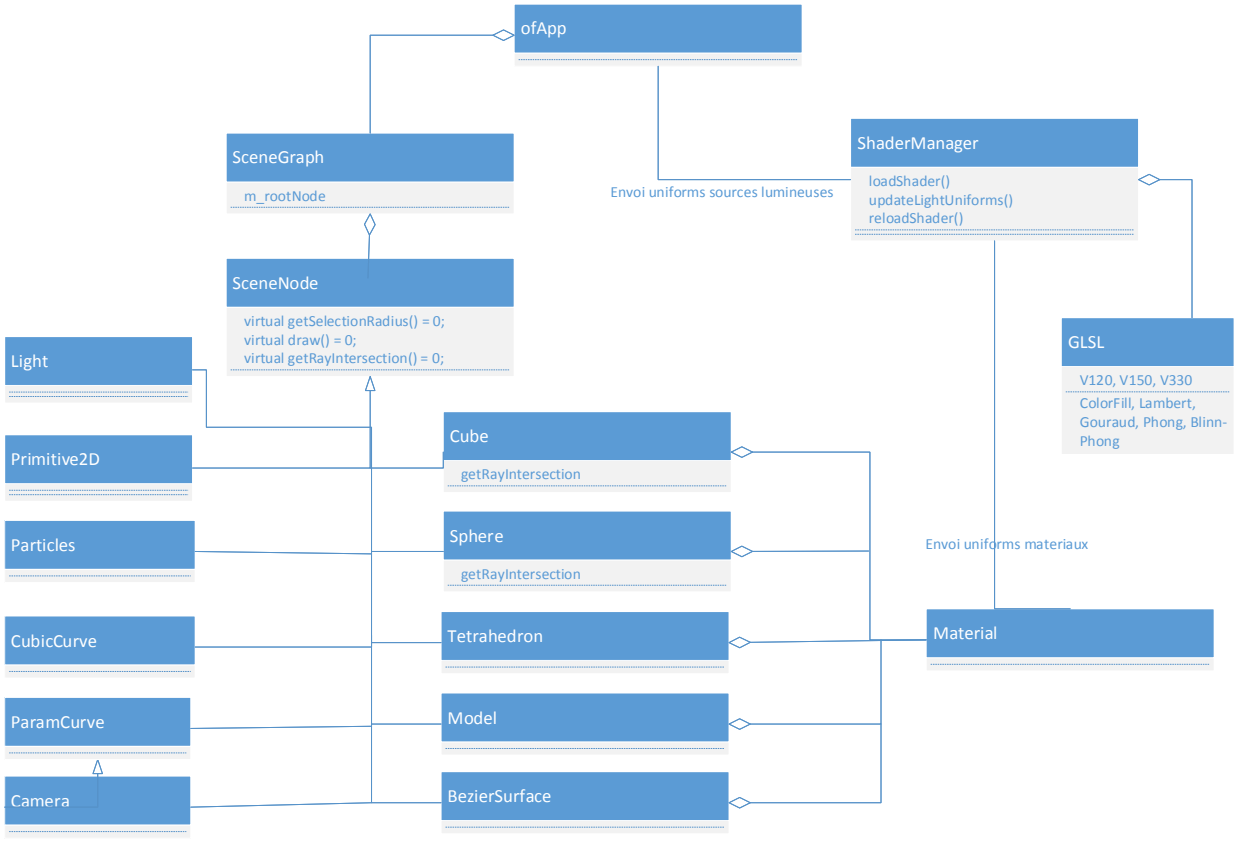
Pour sélectionner un objet, il suffit de cliquer sur l'objet avec **le bouton de souris droite**. Lorsqu'un objet est sélectionné son menu d'édition apparaît à la droite et expose les paramètres modifiables pour cet objet, tels que la couleur de remplissage, etc. De même, les contrôles de la transformation de l'objet sont disponibles dans le menu pour permettre à l'usager d'appliquer une translation (en x, y ou z), une rotation (autour des axes x, y ou z) ou un changement de proportion (« scale ») à l'objet. Le choix du type de shader, les paramètres d'éclairage, la sélection parmi des matériaux prédéfinis, et la sélection d'une texture sont présentés dans le sous-menu Material de la primitive. Afin d'accéder à tous les menus offerts, il est important rouler/dérouler les menus et sous-menus grâce au symbol « - » à droite du titre du menu.

3. Technologie

Afin de développer l'application pour ce projet, Visual Studio 2015 fut utilisé comment environnement de développement, ainsi que OpenFrameworks qui offre un niveau d'abstraction des fonctions OpenGL, ainsi que des outils pour la création de GUI et la gestion d'évènements de la plateforme. L'application a été programmée en C++.

4. Architecture

L'architecture de l'application est présentée dans le diagramme UML suivant :



La gestion de l'interface avec l'usager est gérée par la classe ofApp. Cette dernière traite des évènements venant du clavier ou de la souris, détermine l'objet sélectionné et affiche un cercle de sélection, présente les menus contextuels, s'occupe de dessiner la scène, effectue le mode de lancer de rayon, et transmet les uniformes au ShaderManager pour l'utilisation de shaders.

Une instance ofApp possède un objet SceneGraph, qui représente la structure arborescente des entités de la scène. Chaque entité créée dans la scène est une SceneNode. La classe SceneNode est virtuelle pure et toutes les entités de la scène dérivent de SceneNode. Les entités disponibles sont définies dans les classes Primitive2D, Cube, Tetrahedron, Model, Particles, ParamCurve, CubicCurve, BezierSurface, Light et Caméra. Chacune définit ses paramètres éditables afin d'alimenter les menus d'édition dans ofApp. Ces « ofParameters » sont définis à l'intérieur des constructeurs de chacune des classes. Les classes définissent aussi entre autres leurs fonctions draw(), qui précisent comment dessiner les entités en question.

Les primitives trois dimensionnelles possèdent une instance de Material afin de permettre l'illumination de ces objets. La classe Material est responsable de transmettre les uniformes des matériaux au

ShaderManager qui charge le shader selon le mode d'illumination sélectionné et la version GLSL choisie par l'usager au lancement de l'application.

La classe SceneNode définit ce qui est commun à toutes les entités et donc gère les transformations. Les matrices de transformations sont calculées et obtenues via des fonctions définies à l'intérieur de SceneNode. SceneNode permet aussi la création de lien de parenté entre entités et la matrice locale d'une entité reflète ce lien. SceneNode calcule la matrice monde d'une entité en calculant le produit matriciel entre sa matrice locale et la matrice monde de son parent, et ceci même s'il y a plusieurs niveaux hiérarchiques.

Une instance ofApp possède aussi un pointeur vers une instance caméra de la classe Caméra. La classe Camera expose tous les paramètres d'ajustement de la caméra et alimente le menu correspondant. La classe Camera gère aussi le déplacement de la caméra et le passage entre la projection orthogonale ou la projection perspective. Il est aussi possible de centrer la caméra sur une entité sélectionnée en modifiant la position cible de la caméra.

L'architecture du projet est simple et flexible, ce qui a permis l'ajout de fonctionnalités au cours de l'évolution du projet.

5. Fonctionnalités

La section suivante présente les quinzes (15) critères fonctionnels qui ont été implémentés des cinq catégories suivantes : Image, Dessin vectoriel, Transformation, Géométrie et Caméra.

1. Image

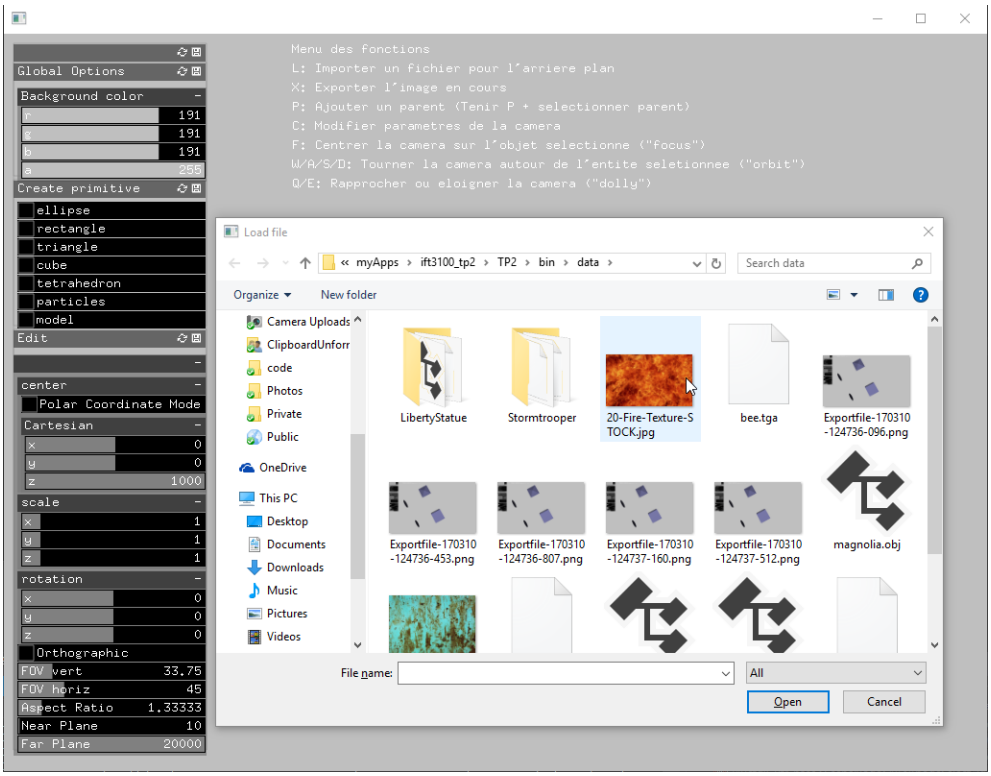
1.1 Importation

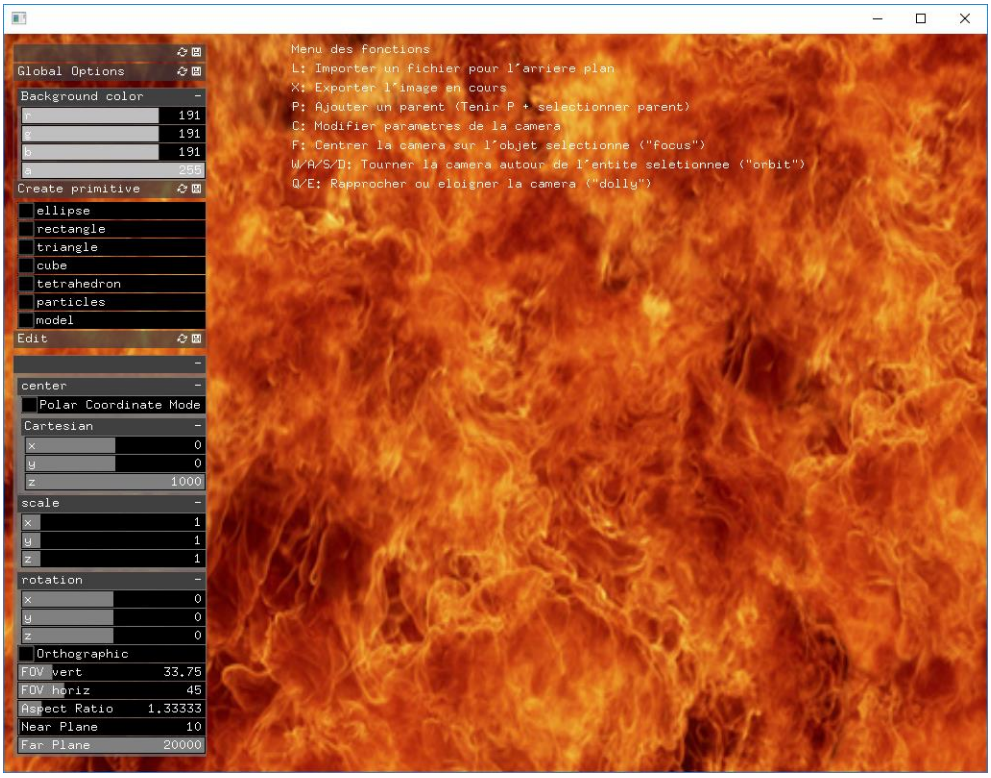
L'application permet à l'utilisateur d'importer des images pour l'arrière-plan ou la texture du nuage de particules, ou des fichiers de modèles 3D de format .obj en accédant à une fenêtre explorateur Windows pour permettre la sélection du fichier désiré.

Dans le cas d'une image en arrière-plan, l'importation est accessible à l'aide de la touche « L » du clavier.

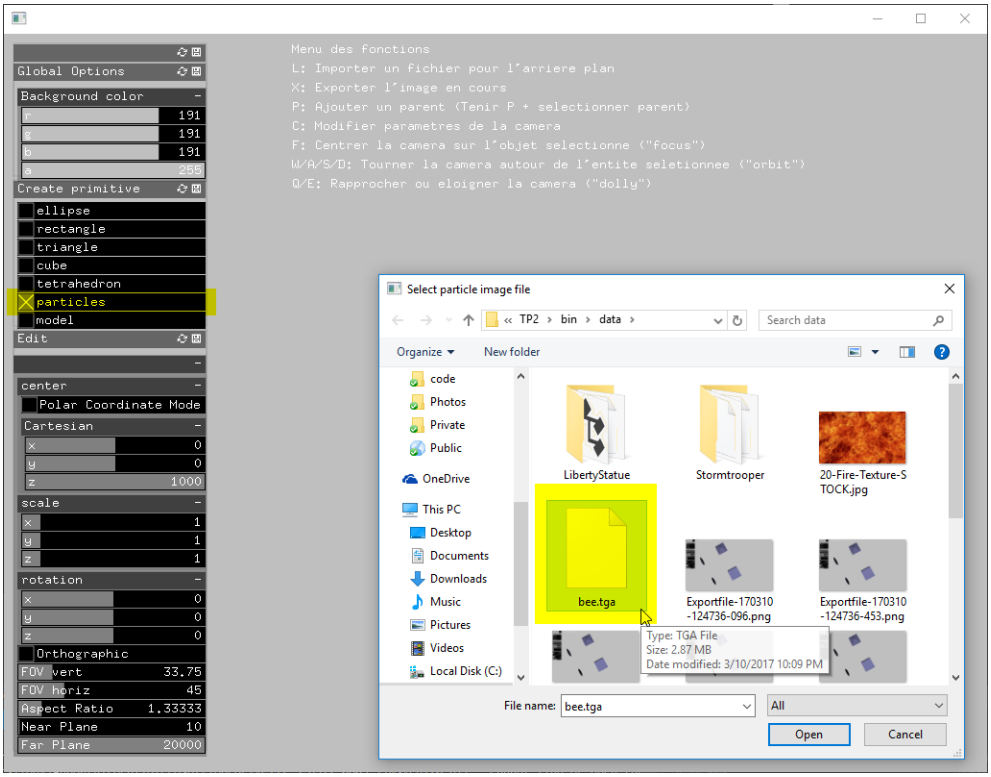
Dans le cas des modèles 3D ou de la texture du nuage de particules, la fenêtre Windows apparaît lorsque l'usager clique dans l'espace écran afin d'indiquer l'endroit de création voulu pour l'objet.

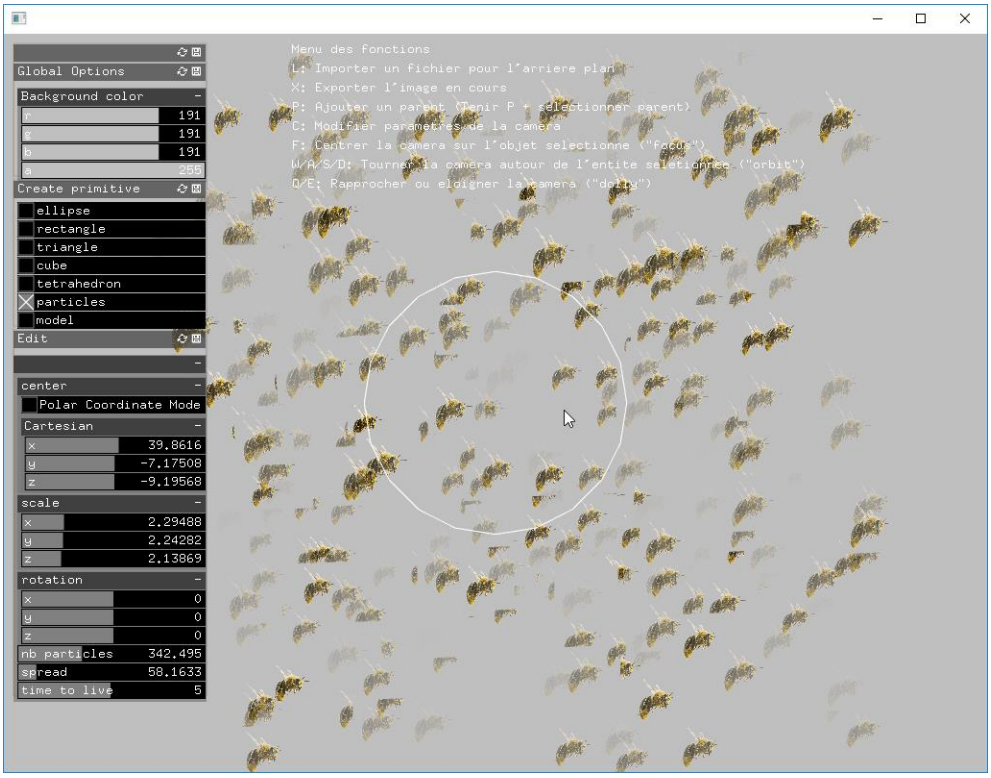
Importation d'une image pour l'arrière-plan :





Importation d'un texture pour le nuage de particules :





1.2 Exportation

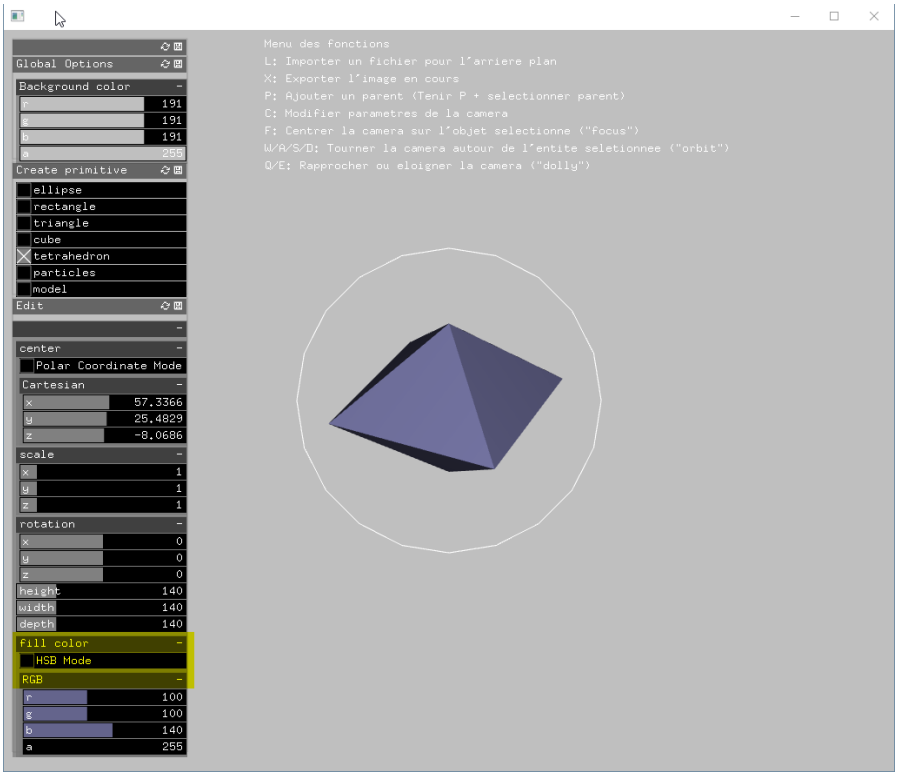
L'application permet à l'utilisateur d'exporter une image de la scène sous format « png » à l'aide de la touche « X » du clavier.

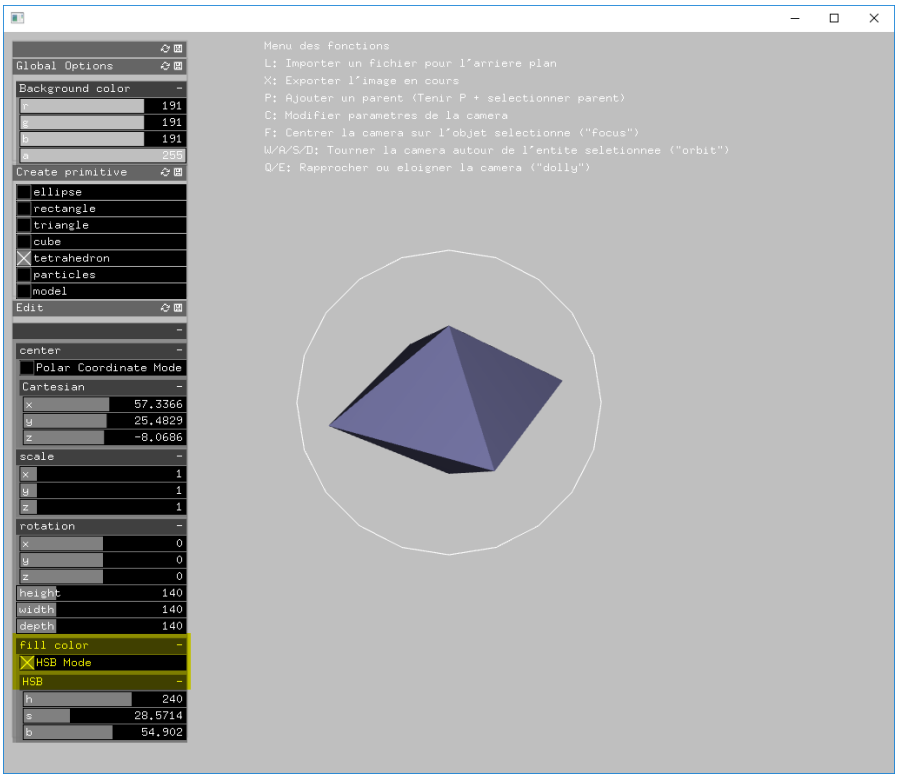
1.3 Espace de couleur

L'application permet à l'usager de sélectionner entre deux modes de couleur pour le remplissage des primitives 2D ou 3D et même pour les lignes de contour. Le mode par défaut est RGB, mais une case à cocher permet d'activer le mode de couleur HSB. Afin d'offrir à l'usager ces choix, j'ai créé une classe nommée ColorParameters qui regroupe les paramètres de couleur en un sous-groupe de ofParameters de la primitive créée. Par la suite, selon le mode de couleur voulu, ColorParameters s'assure que les paramètres correspondant soient affichés et que les valeurs soient affichées correctement lors du passage entre ces deux modes.

J'ai implémenté les algorithmes de conversion entre les modes de couleurs RGB et HSB, et les paramètres de couleur pour les primitives sont convertis à l'aide de ces fonctions.

Sélection du mode de couleur :





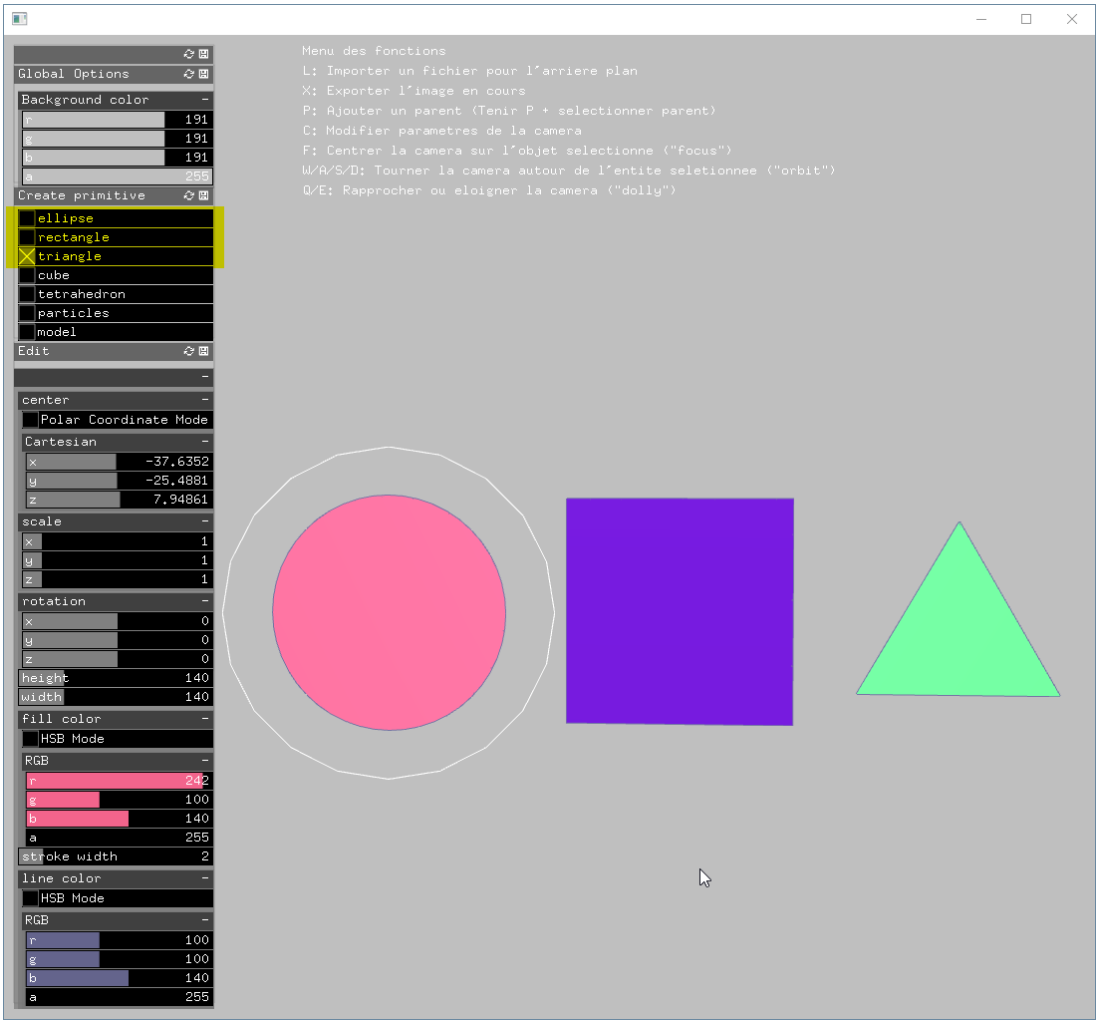
2. Dessin vectoriel

2.2 Primitives vectorielles

Grâce au menu de création de primitives, l'usager peut créer les trois primitives 2D présentés : triangle, rectangle et ellipse. Il s'agit simplement de sélectionner la primitive, et suite à un clique du bouton de souris gauche dans l'espace écran, la primitive est créée à cet endroit dans l'espace écran.

La classe Primitive2D s'occupe de la création des primitives, gère le dessin de la primitive correspondante à l'aide des fonctions offertes par OpenFrameworks, et fournit les paramètres modifiables au GUI.

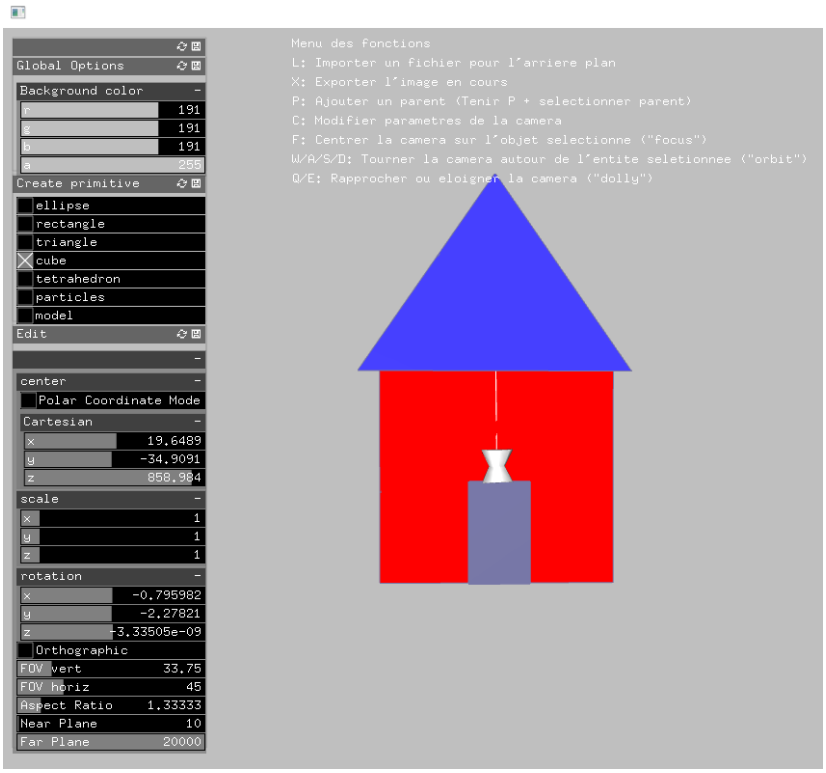
Création primitives 2D :



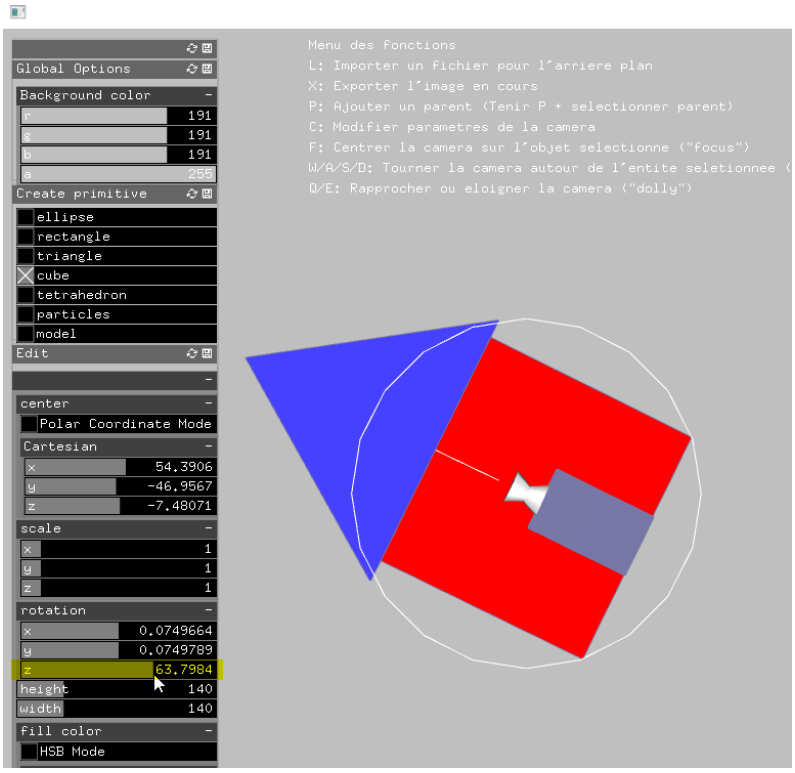
2.3 Formes vectorielles

L'application permet de créer des liens de parenté entre les entités à l'écran. Conséquemment, les objets avec un lien de parenté se déplacent ensemble et forment un tout. Par contre, chacun peut être sélectionnés individuellement et les paramètres de dessin (ex : couleur de remplissage) sont modifiés de façon individuelle. Le code permettant la création de ce lien est présenté dans la section 3.1 Transformation interactive.

Création d'une forme :



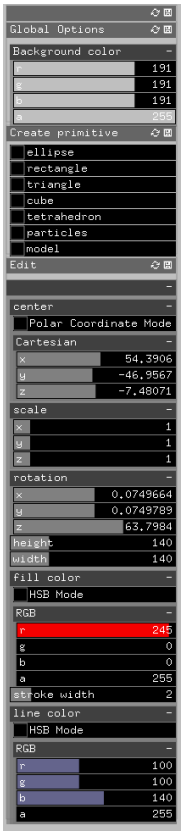
Rotation de la forme :



2.4 Outils de dessin

L'application permet à l'utilisateur de modifier interactivement la valeur des outils de dessin vectoriels, tels que l'épaisseur et la couleur de la ligne de contour, la couleur de remplissage de la primitive, ainsi que la largeur et la hauteur de la primitive. Ces paramètres sont des objets de type « ofParameter » et sont définis dans le constructeur de la primitive.

Menu de dessin :



2.5 Interface

Une interface graphique est présentée à l'utilisateur pour la modification des paramètres de l'image. Divers menus sont offerts selon le contexte de la tâche en cours. Un menu d'options globales inititulé « Global Options » est toujours disponible et permet à l'usager de modifier la couleur de l'arrière plan. De même, un menu de création de primitive permet de sélectionner la primitive voulue. Lorsque la primitive est créée, son menu d'édition correspondant est présenté avec les paramètres associés à son édition. Le menu d'édition apparaît aussi lorsque la primitive est sélectionnée. De plus, les paramètres permettant de contrôler les transformations (translation, rotation et proportions) sont présentés dans le menu d'édition. Finalement, un menu permettant de contrôler les paramètres de la caméra est disponible lorsque l'on pèse sur la touche « C ».

3. Transformation

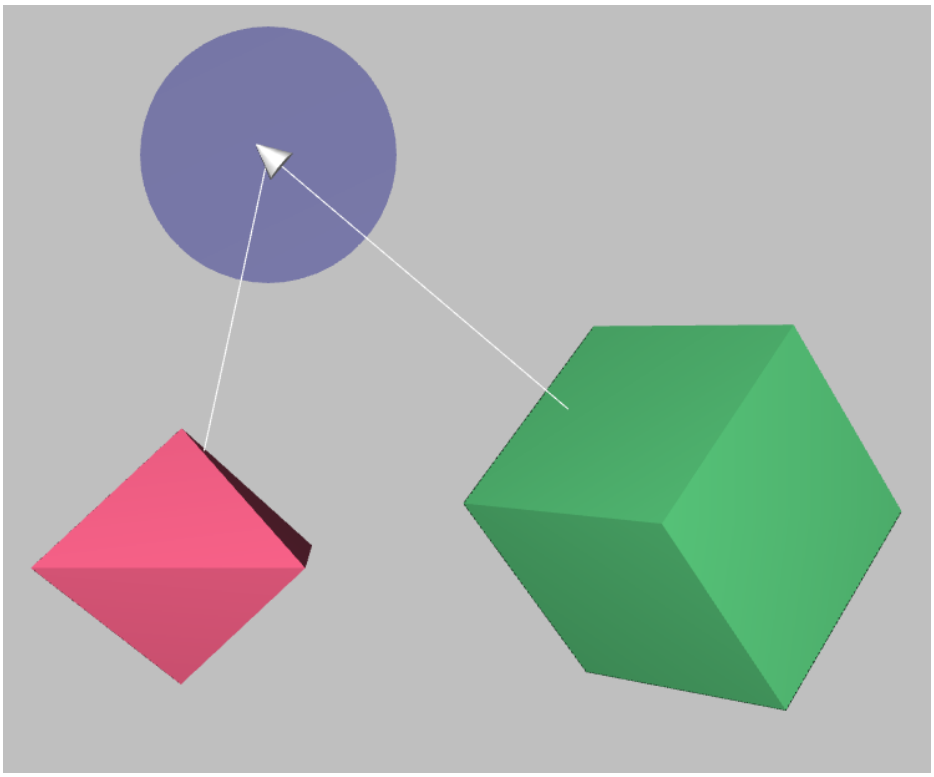
3.1 Transformation interactive

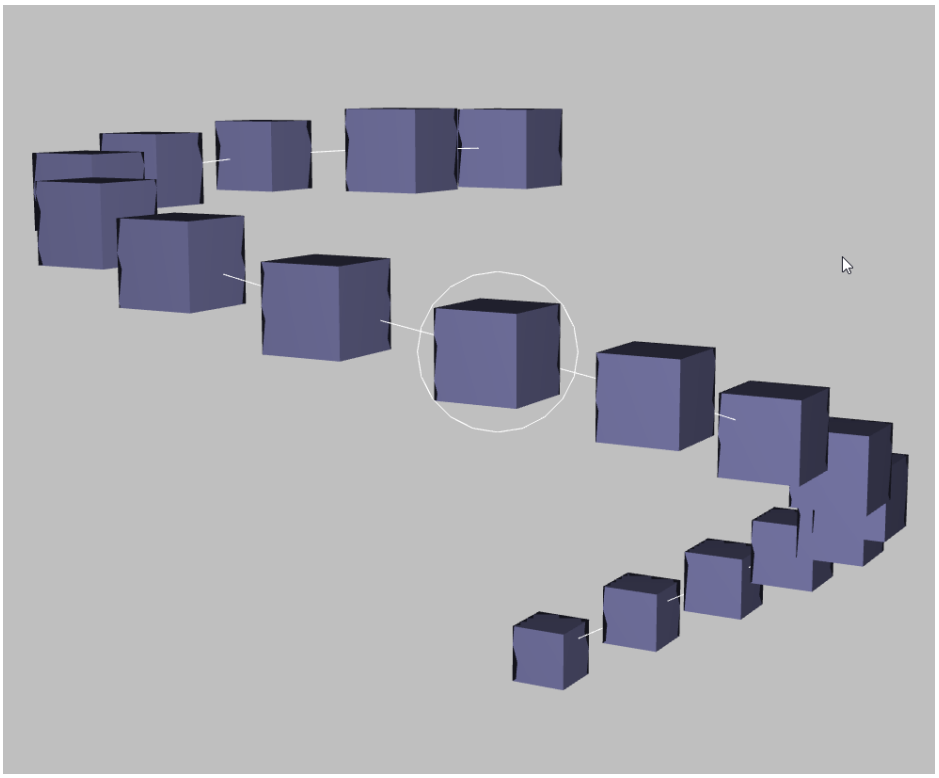
J'ai rédigé la classe SceneNode pour représenter une entité générale qui est créée dans une scène. Il s'agit de la classe de base, par la suite, chaque classe des différentes entités possibles dérivent de SceneNode. La classe de base SceneNode conserve et gère la matrice de transformation de l'entité. Les paramètres reliés aux transformations peuvent être contrôlés par l'utilisateur, tels que la position x, y z dans le cas de la translation, l'angle de rotation autour des axes pour la rotation, et finalement le facteur d'agrandissement pour ce qui est de la proportion. SceneNode permet aussi la création de lien de parenté entre deux entités, qui sera présenté davantage dans la

3.2 Structure de scène

La structure de scène, ou la classe SceneGraph du projet, correspond à la structure de données qui gère les objets SceneNode de la scène. Il s'agit de l'implémentation d'un arbre, qui utilise des « shared pointers » afin d'assurer une bonne gestion de la mémoire. La classe SceneGraph offre l'interface pour l'ajout de lien de parenté entre objets. Grâce à des fonctions utilisant la récursivité, telles que *drawchildren* et *getAllSceneNodes,* il est possible de faire un parcours de l'arbre en profondeur afin de dessiner la scène. Une flèche est dessinée allant d'objets enfants vers leur parent, afin d'offrir une aide visuelle.

Liens de parenté :

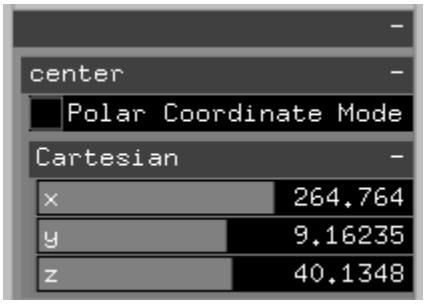




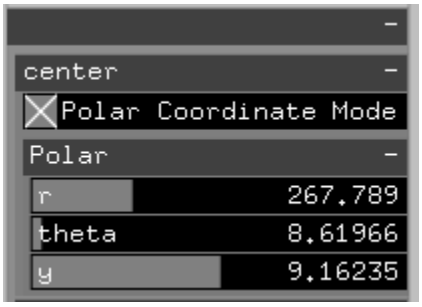
3.4 Coordonnées non-cartésiennes

L'application permet à l'usager de sélectionner entre deux représentations de coordonnées pour le déplacement des primitives 2D ou 3D. La représentation des coordonnées est cartésienne par défaut, mais une case à cocher permet d'activer les coordonnées polaires. Afin d'offrir à l'usager ces choix, j'ai créé une classe nommée PositionParameters qui regroupe les paramètres de position en un sous-groupe de « ofParameters » de la primitive créée. Par la suite, selon le mode de coordonnées voulu, PositionParameters s'assure que les paramètres correspondant soient affichés et que les valeurs soient affichées correctement lors du passage entre ces deux modes.

Coordonnées cartésiennes :



Coordonnées polaires :



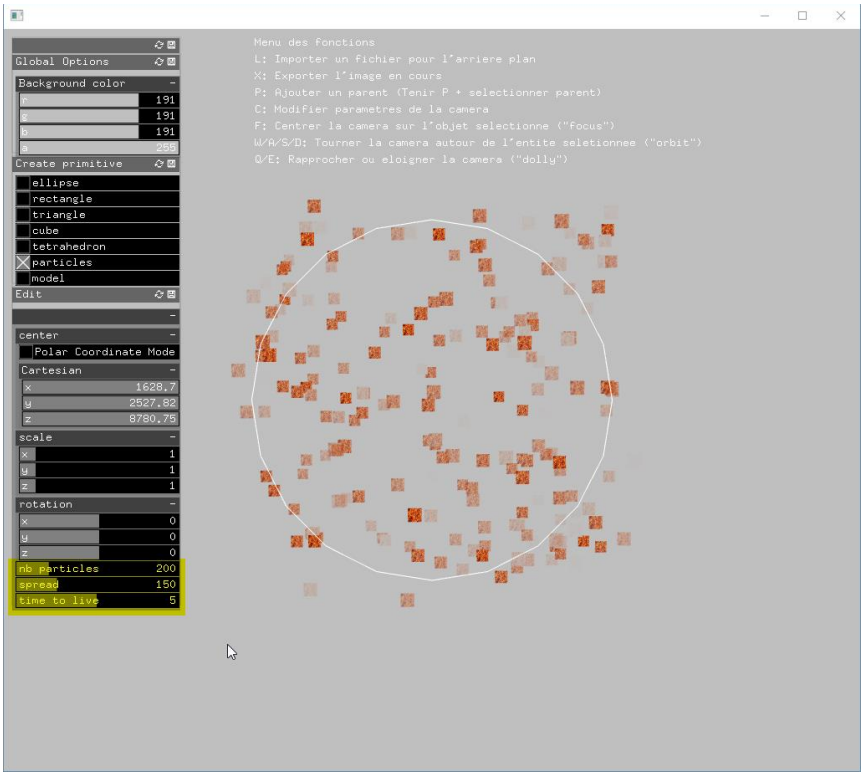
4. Géométrie

4.1 Particules

L'application offre la possibilité de créer un nuage de points avec une texture, nommé « particules » dans le menu de création. Ce nuage de point créée de façon aléatoire des points autour d'un centre. Lorsque l'usager sélectionne « particles » du menu de création et ensuite clique dans l'espace écran, une fenêtre explorateur Windows apparaît, permettant à l'usager de sélectionner une texture pour les points.

L'application offre trois paramètres d'ajustement pour le nuage de points, soit le nombre de points (« nb particles »), la dispersion des points (« spread ») et le temps de vie (« time to live ») de chaque particule. Le temps de vie représente la durée d'affichage d'un point. La valeur alpha à la création d'un point est maximale et cette dernière décroit progressivement pendant le temps de vie spécifié jusqu'à ce que la valeur alpha soit à zéro, et la particule ne soit plus visible.

Nuage de points :



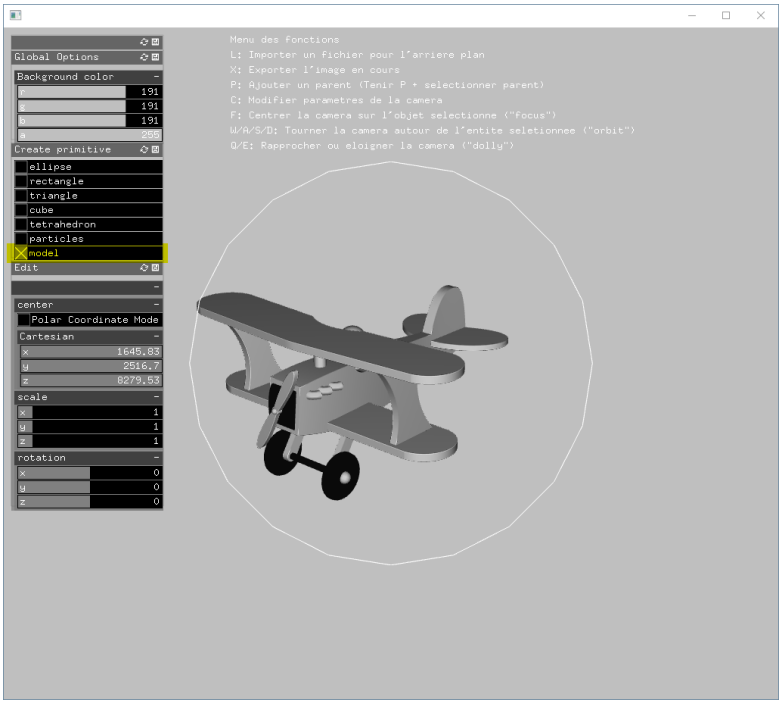
4.2 Primitives 3D

L'application permet la création de deux primitives 3D, de type solide platonique, qui sont générés à partir d'algorithme sans aucune donnée externe. Ces solides sont un cube et un tétraèdre. Les classes Cube et Tetrahedron créent un buffer de géométrie en mémoire, et ensuite s'occupent du transfert de ce buffer vers le GPU. Le buffer de géométrie définit les sommets et les faces, et spécifie les normales pour chaque sommet. Par la suite, les commandes OpenGL pour le dessin du maillage sont utilisées. Les paramètres d'ajustement de la couleur, de la position, de la largeur, de la profondeur et de la hauteur sont disponibles pour que l'utilisateur puisse modifier la forme.

4.3 Modèle

L'application permet l'importation de modèles 3D de format .obj et grâce à la classe Model, crée une instance sans aucune fonction fournie par Openframeworks. Après avoir sélectionné « model » dans le menu de création, lorsque l'usager clique dans l'espace écran, une fenêtre explorateur Windows permet de choisir un fichier pour le modèle. La classe Model crée un objet en lisant les valeurs des sommets, des faces et des normales (si disponible) du fichier .obj et génère le buffer de géométrie correspondant. Ce buffer est ensuite transféré au GPU pour permettre l'affichage du modèle à l'aide des commandes OpenGL de dessin. Les modèles fournis (Statue of Liberty, Toyplane, et Stormtrooper) affichent correctement dans l'application, et comme la classe Model dérive de SceneNode, il est possible de modifier la position, la rotation et la proportion des modèles.

Importation de modèle :

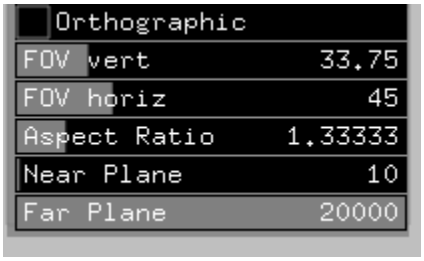


5. Caméra

5.1 Propriétés de caméra

La classe Camera est une classe qui encapsule un objet de type ofCamera et dès le lancement de l'application une caméra est créée par ofApp. Les paramètres permettant de modifier les attributs de la caméra tels que les angles de champ de vision (horizontale et verticale), le ratio d'aspect et la distance du plan de clipping avant et arrière sont disponibles lorsque la touche « C » est enclenchée.

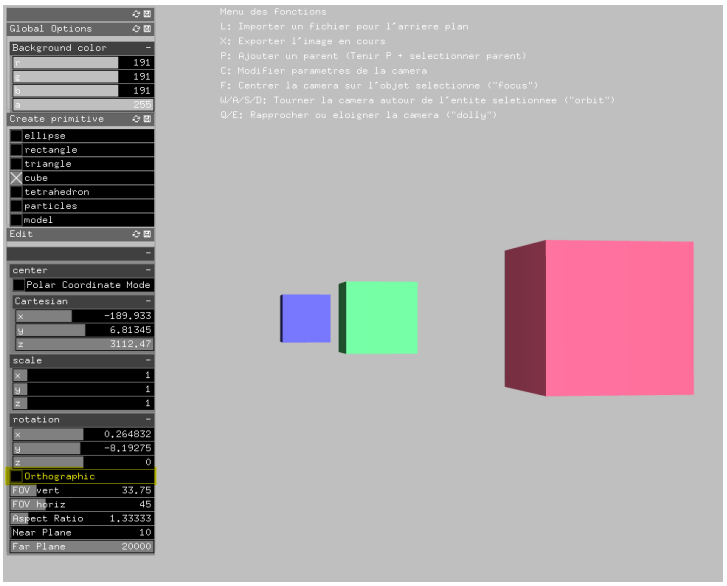
Paramètres camera :

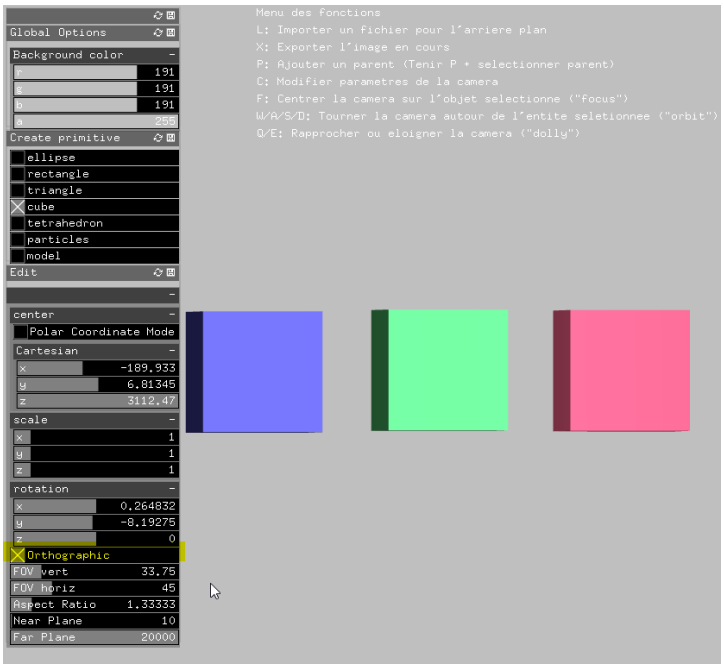


5.2 Mode de projection

Les deux modes de projection orthogonale et perspective sont disponibles dans le menu des paramètres de la caméra grâce à la case à cocher. La fonction orthoChanged de la classe Camera gère le passage entre les deux modes de projection.

Projection perspective :





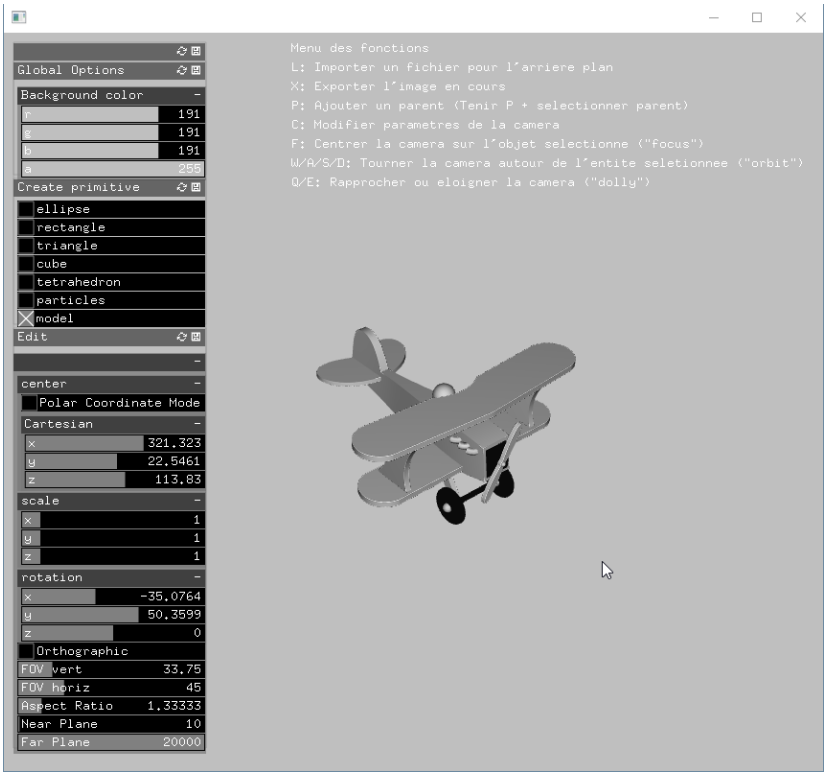
5.3 Caméra interactive

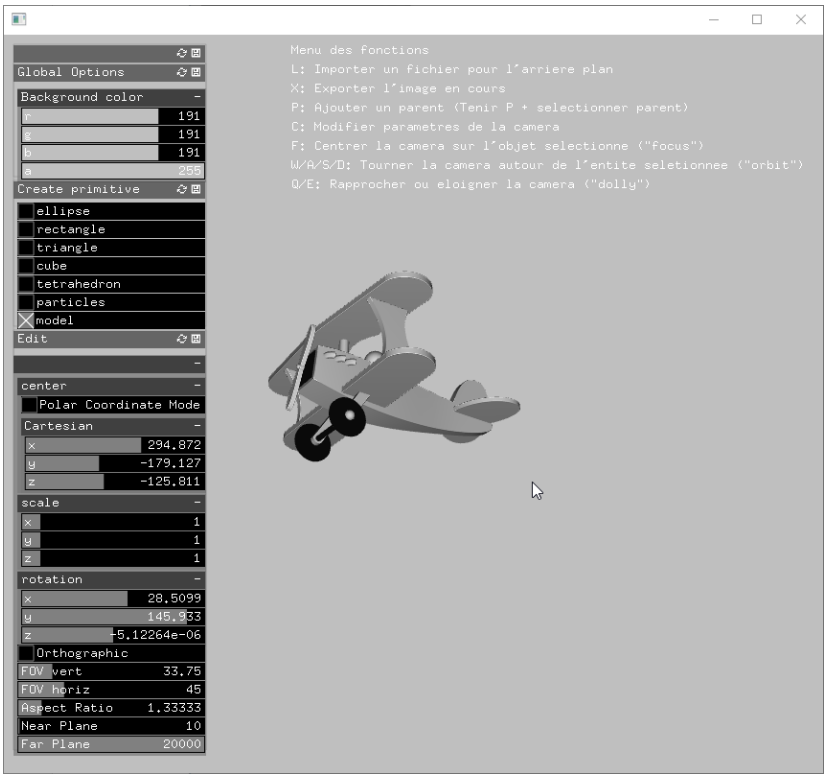
La caméra de l'application est interactive, ce qui signifie qu'elle peut s'approcher et s'éloigner d'une entité grâce aux touches « E » et « Q » respectivement, et de tourner autour de l'entité à l'aide des touches « W », « A », « S », et « D ».

Afin de rendre la caméra interactive, il fallait en premier lieu élaborer un mécanisme de sélection d'entités dans la scène. Ceci a été accompli via la fonction mousePressed dans ofApp. Tous les SceneNode fournissent une définition de la fonction virtuelle *getSelectionRadius,* qui représente le rayon de la sphère qui englobe l'objet. La position de chaque SceneNode de la scène est vérifiée en faisant une projection dans l'espace écran, et en déterminant si la position de la souris se retrouve à l'intérieur du rayon du centre de l'entité. Si oui, le SceneNode est sélectionné et le cercle de la zone de sélection de l'entité est affiché à l'écran.

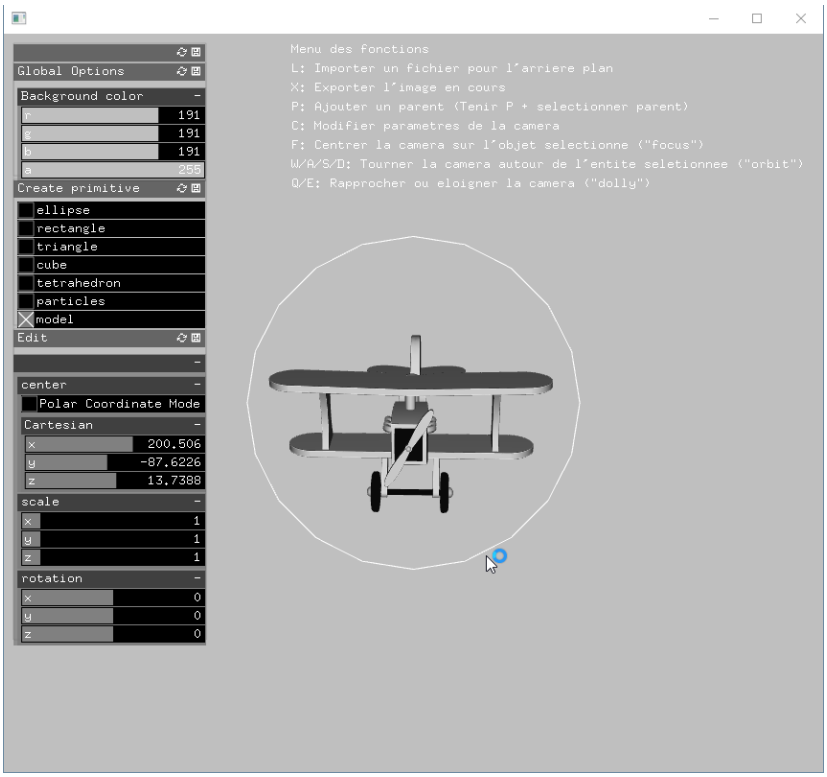
De plus, il est possible de centrer la caméra sur l'entité sélectionné. Ceci est accompli en faisant la touche « F » (pour focus).

Déplacement de la camera:





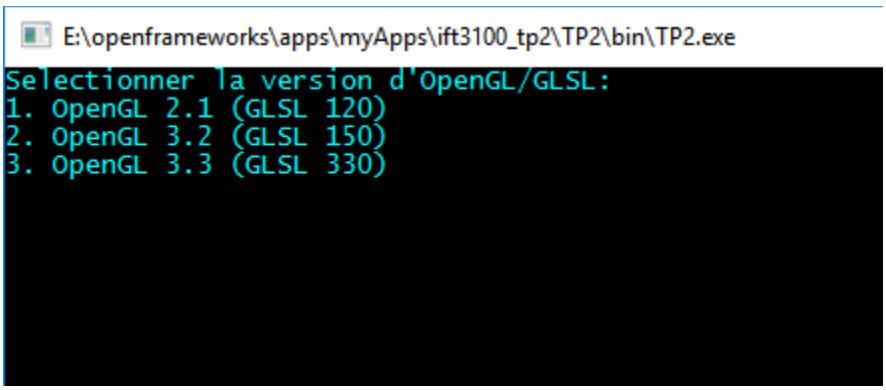
Cercle de sélection :



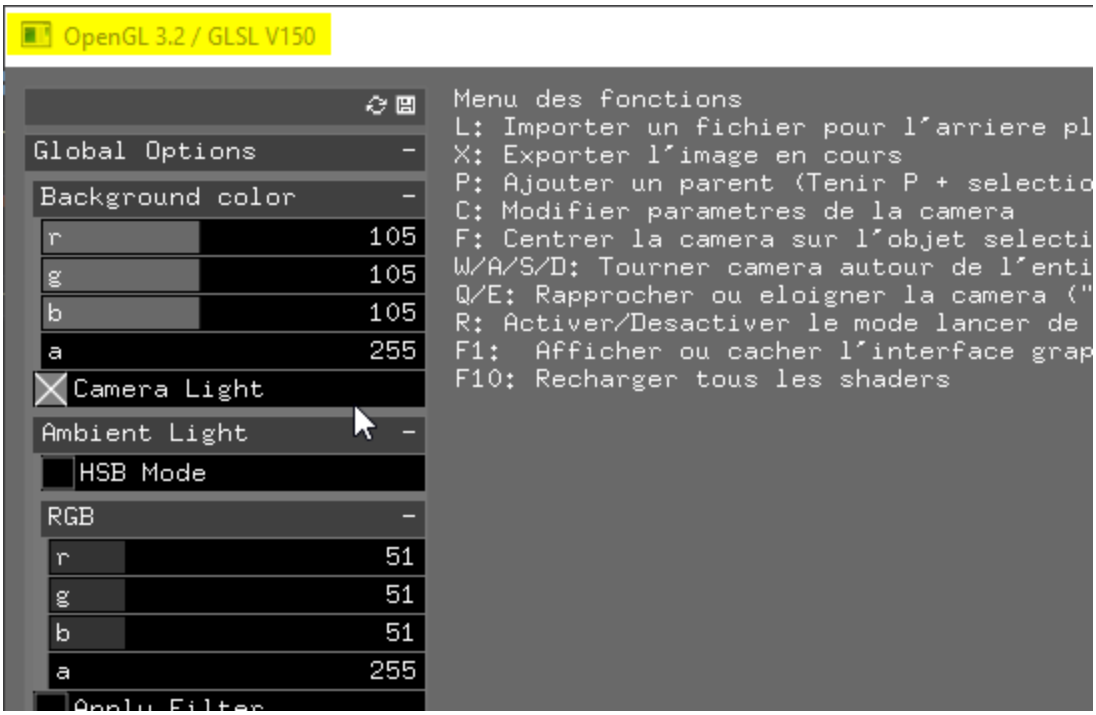
6. Pipeline de rendu

6. 1 Portabilité

L'application supporte trois différentes versions de GLSL, soit 120, 150 et 330. La sélection de la version supportée se fait dès le départ dans le terminal, afin de lancer l'application avec la version GLSL souhaitée.



Par la suite, la version apparaît en tant que titre de la fenêtre principale.



Des shaders pour tous les modes d'illumination sont offerts sous ces trois versions : le shader ColorFill, Lambert, Gouraud, Phong et Blinn-Phong. Ci-dessous, les trois versions du fragment shader Gouraud.

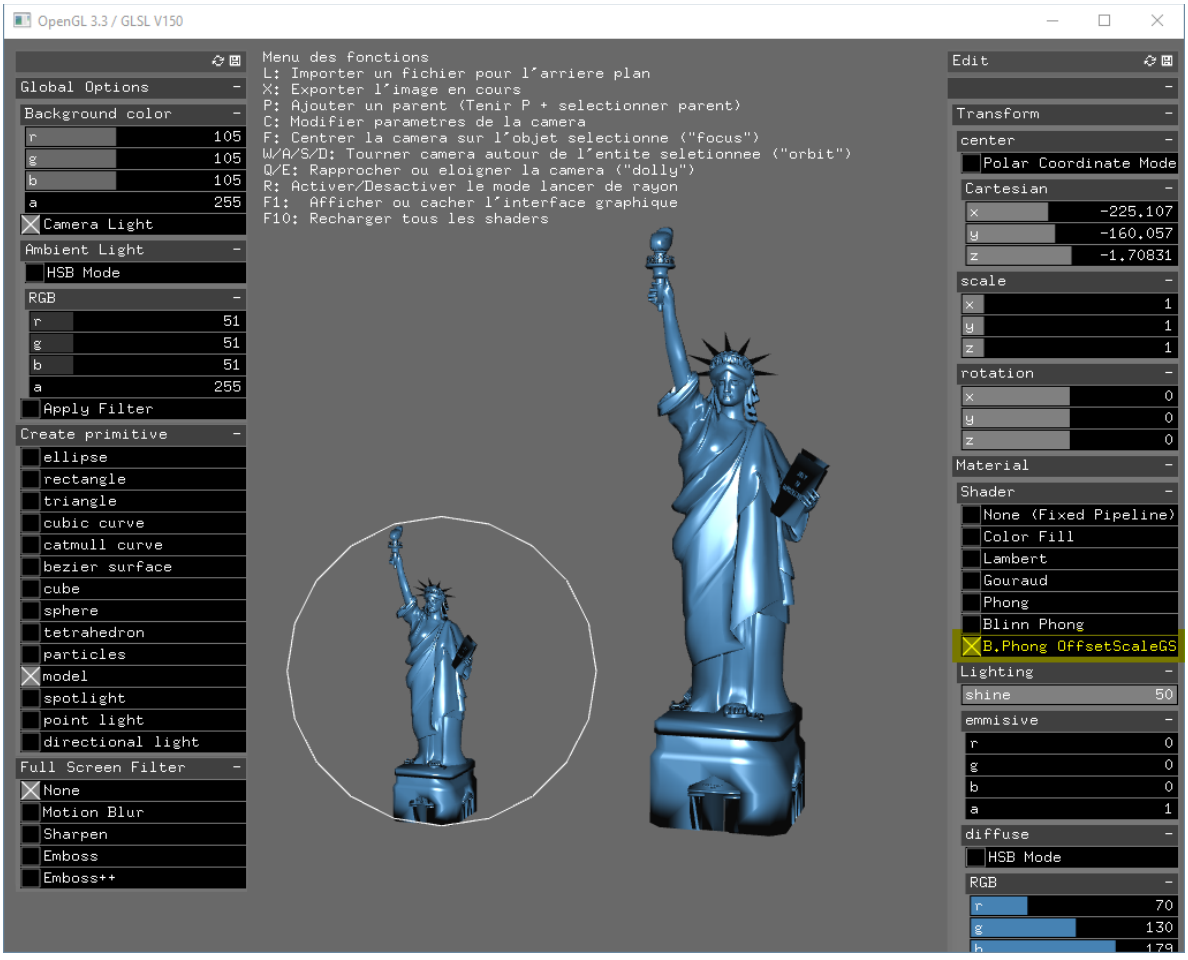
J'ai aussi développée une classe afin de gérer ces divers shaders et ces diverses versions GLSL, nommée ShaderManager. C'est cette classe qui permet de charger le shader voulu selon le modèle d'illumination et la version GLSL choisie, et d'envoyer les uniforms requis au shader. De plus, afin de faciliter le développement des shaders lors de leurs rédactions, j'ai écrit une fonction nommée reloadShaders pour recharger les shaders même lorsque l'application est en fonctionnement, ce qui a beaucoup faciliter le travail itératif.

6.2 Shader de géométrie

Pour ce critère, j'ai développé un shader de géométrie combiné avec le shader Blinn-Phong. Le shader nommé OffsetScaleGS.glsl crée une deuxième primitive à partir d'une première primitive en entrée. Cette nouvelle primitive correspond à une translation dans l'espace écran de l'entité originale, et applique un facteur d'agrandissement de 2.

Le shader de géométrie se trouve à être appler entre le shader de sommets et le shader de fragments, ce qui implique que je devais m'assurer que les sorties (out vec) et les entrées (in vec) pour chaque shader étaient cohérents, et que les uniform requises au shader de géométrie étaient spécifiées. Le shader de géométrie comporte deux parties, la première étant un « pass-through » de la primitive originale, c'est-à-dire la primitive est transmise sans changement. La deuxième partie crée la deuxième primitive à partir des sommets, applique un changement de proportion via une matrice de proportion et effectue une translation. La même transformation a été effectuée sur les interpolationPosition afin d'assurer que les calculs d'illumination pour ce deuxième objet soient corrects.

Shader de géométrie :



7. Illumination

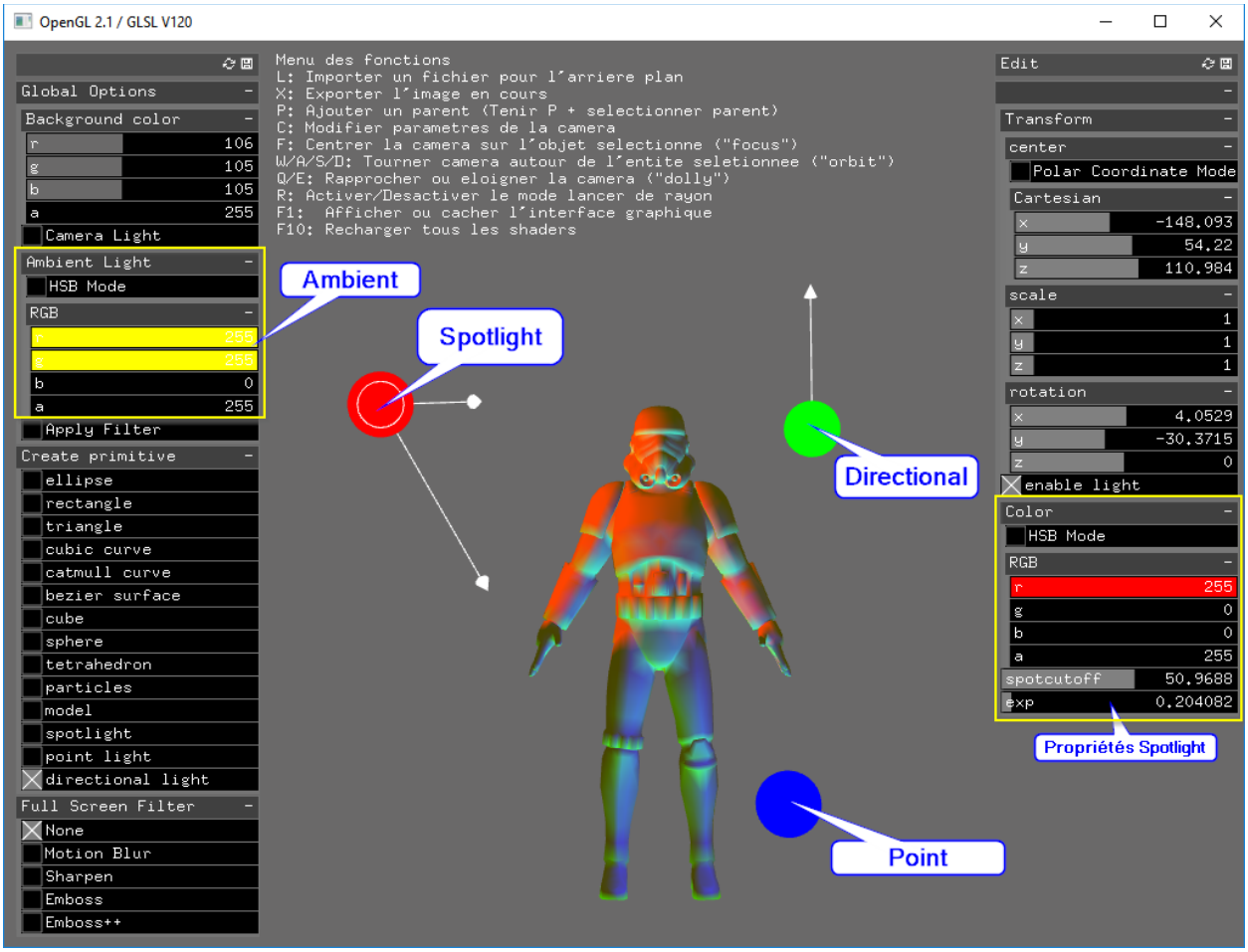
7.1 Types de lumières

Dans le mode de pipeline fixe, les quatre type de lumières suivantes peuvent être ajoutées à une scène : ambiante, directionnelle, ponctuelle et projecteur. Les lumières héritent d'une classe de base nommée Light, qui dérive de SceneNode, ce qui signifie que les lumières sont ajoutées au SceneGraph de la scène.

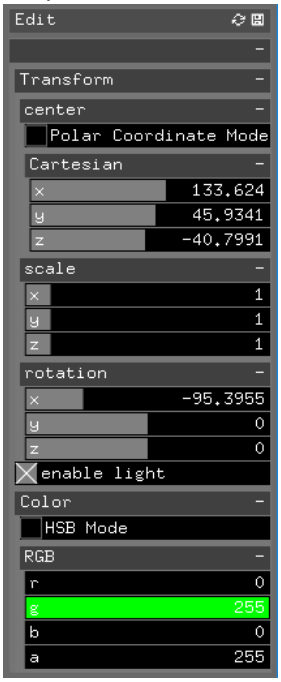
Les paramètres généraux, tels que la couleur de la lumière, sont modifiables, ainsi que les paramètres spécifiques aux types de lumières. Dans le cas d'une lumière de type projecteur, le spotCutoff et l'exposant, peuvent être ajustés tel que démontré ci-dessous.

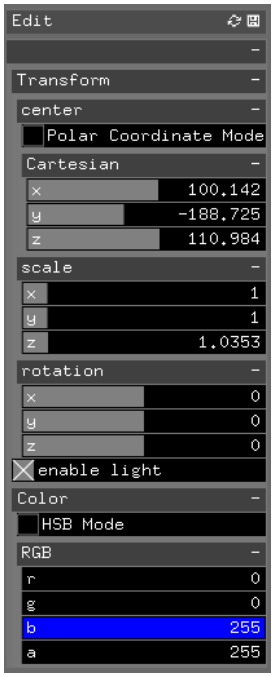
Les vecteurs démontrant le cône d'illumination sont dessinés pour les sources projecteurs, de même qu'un vecteur indiquant la direction est dessiné pour la source directionnelle.

Types de lumières : ponctuelle, projecteur, ambiante et directionnelle



Propriétés de la lumière directionnelle (gauche) et propriétés de la lumière ponctuelle (droite) :



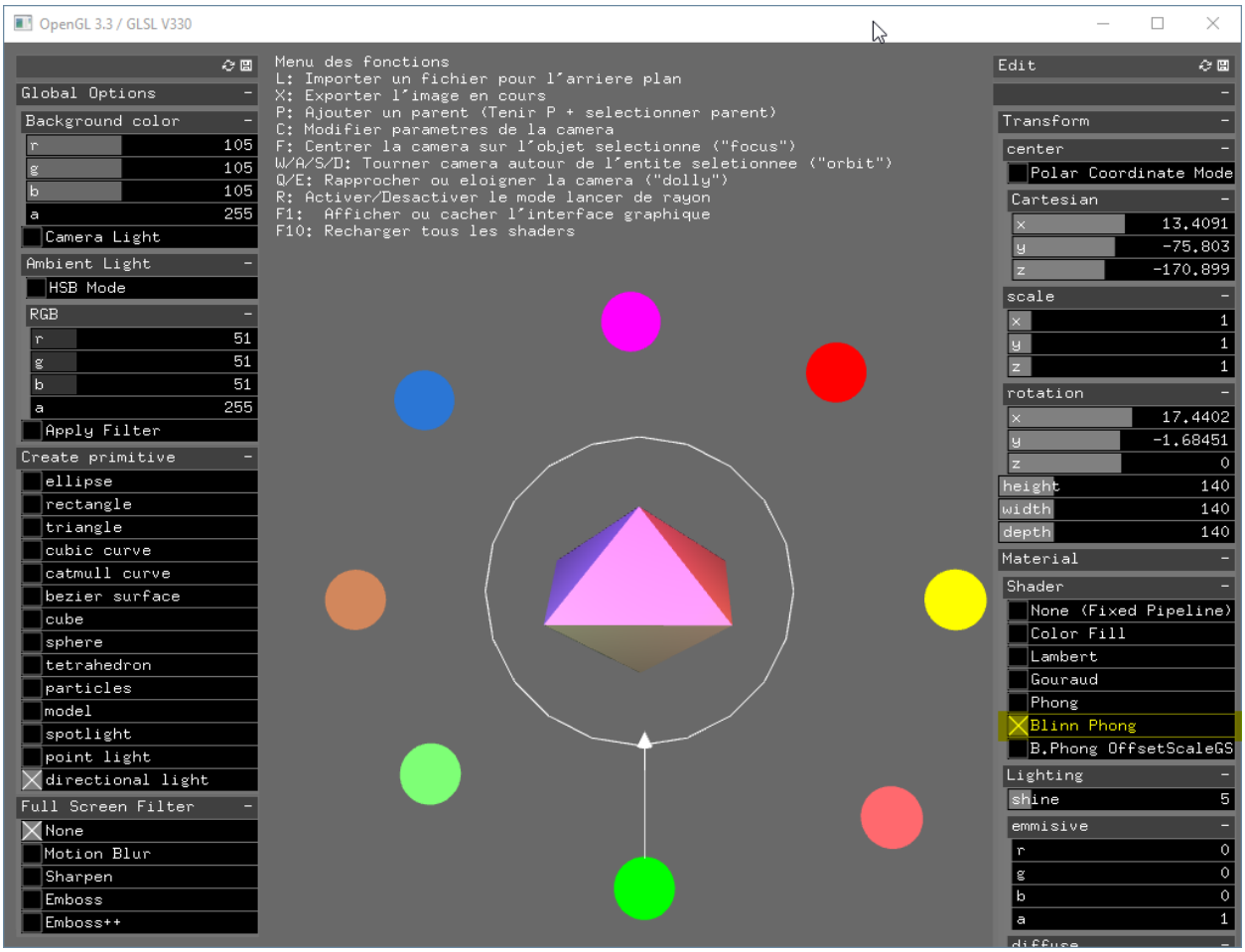


7.2 Lumières multiples

Le shader Blinn-Phong disponible dans toutes les versions GLSL supportées permet d'avoir 8 instances de lumières dynamiques. En plus de la lumière ambiante, deux types de sources lumineuses additionnelles sont possibles : les lumières de type ponctuelle et les lumières de type directionnelle. Pour les lumières de la scène, la couleur, la position et la direction sont prises en compte lors des calculs d'illumination Blinn-Phong. De plus, toute lumière peut être activée ou désactivée et ceci est réflétée dans les calculs.

Comme il a déjà été mentionné, les uniforms des sources lumineuses sont envoyés au shader par le ShaderManager.

Modèle d'illumination Blinn-Phong avec huit sources lumineuses actives :



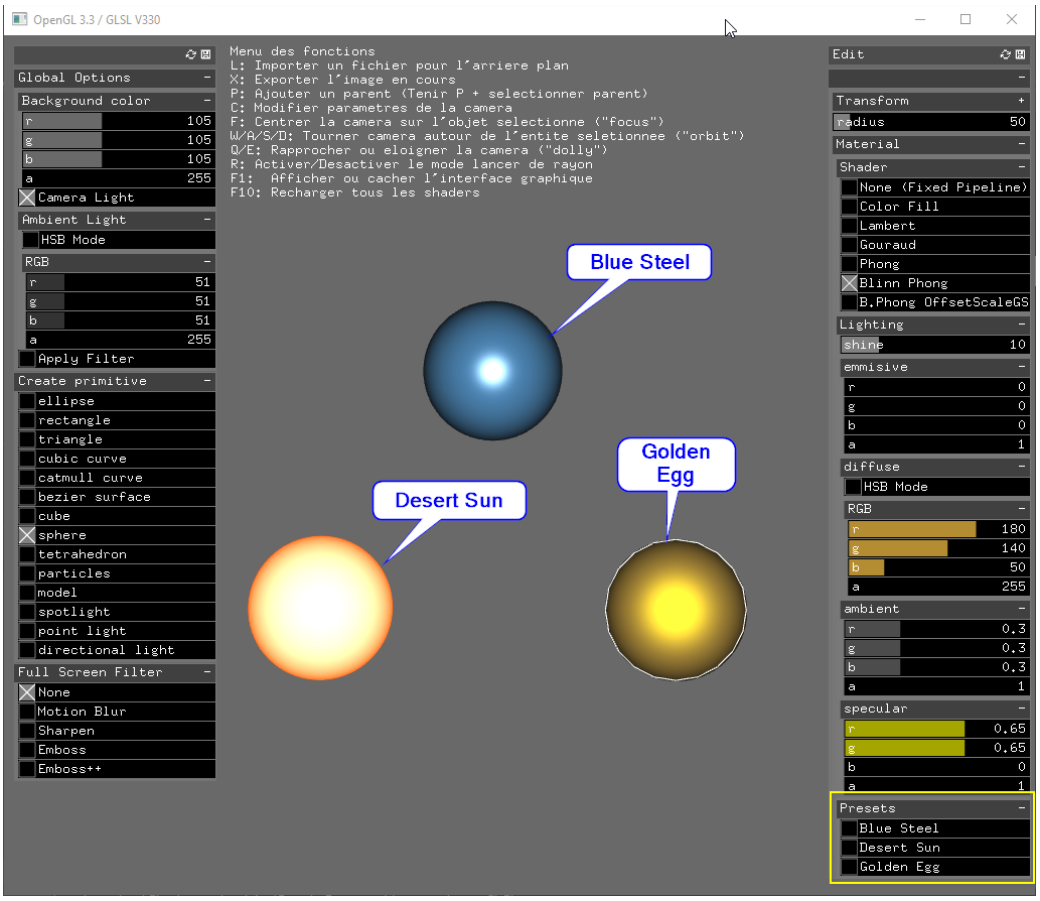
7.3 Matériaux

Les matériaux des objets sont gérés par une classe nommée Material, qui permet l'édition des paramètres, tels que la couleur diffuse, la couleur spéculaire, la couleur émissive et le facteur de brillance

(shine). Tout objet possédant un matériel aura la possibilité de modifier ses paramètres grâce aux fonctions offertes par Material.

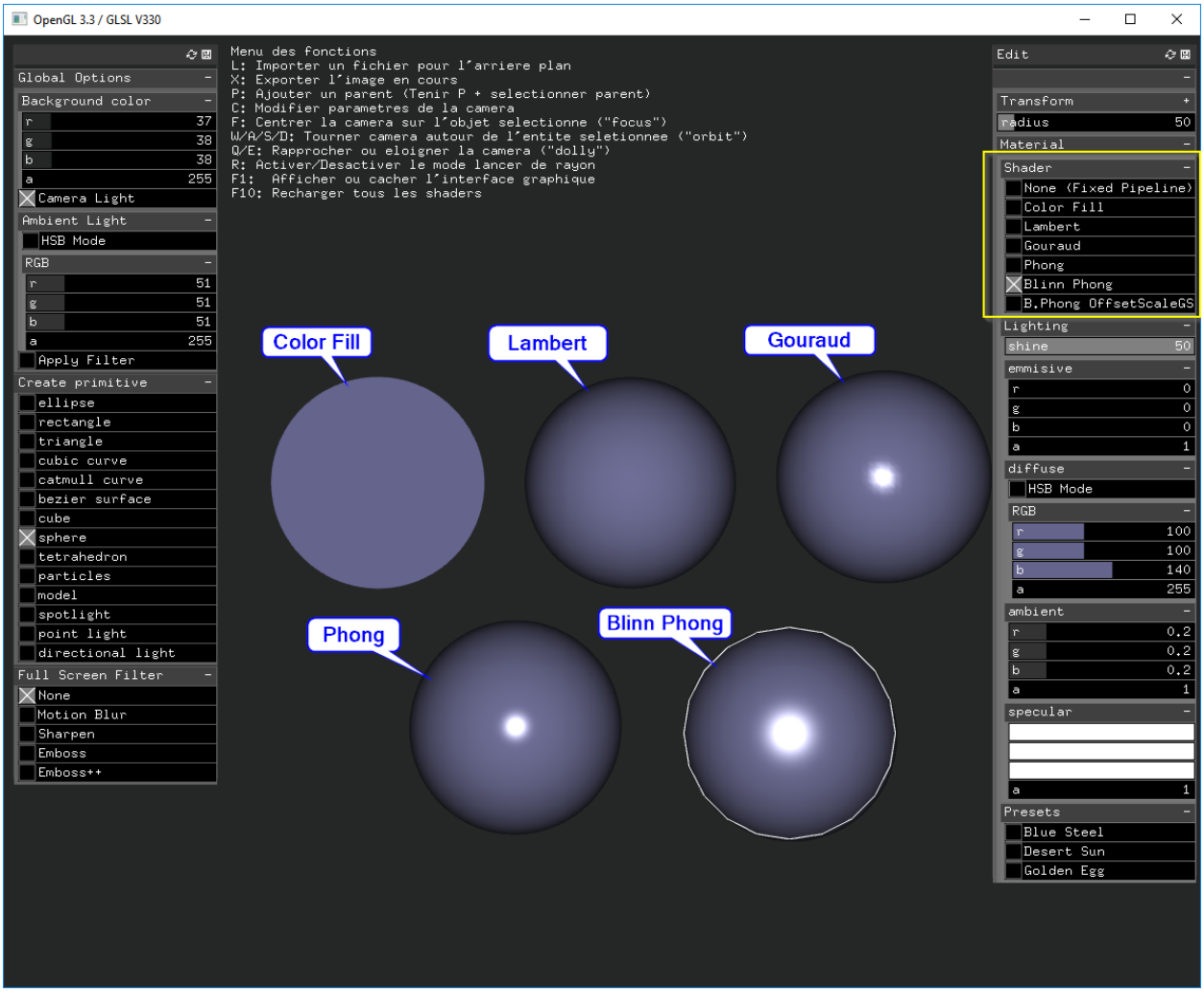
De plus, la class Material offre des matériaux de base (« preset »), avec les paramètres déjà configurés. Ces derniers sont « Blue Steel » un matériel de couleur bleu-gris avec une réflextion spéculaire et un haut niveau de brillance, « Desert Sun » qui est plutôt un matériel émissif, avec aucune réflexion spéculaire, et un matériel nommé « Golden Egg », qui est un matériel dorée avec de la réflexion spéculaire. Ces matériaux peuvent être sélectionnés via un menu, et après avoir été choisis, peuvent être ajustés via les paramètres déjà

Trois matériaux offerts au menu Presets :



7.4 Modèles d'illumination

Tel qu'a déjà été mentionné, des shaders pour chacun des quatre modèles d'illumination sont offerts (Lambert, Gouraud, Phong et Blinn-Phong) tous supportant une lumière dynamique ponctuelle et tenant compte du matériel de l'objet sélectionné. Seul le shader Blinn-Phong permet l'ajout de plus d'une source de lumières, allant jusqu'à un maximum de 8, de types ponctuelles ou directionnelles.


OpenGL 3.3 / GLSL V330
Menu des fonctions
L: Importer un fichier pour l'arriere plan
X: Exporter l'image en cours
P: Ajouter un parent (Tenir P + selectionner parent)
C: Modifier parametres de la camera
F: Centrer la camera sur l'objet selectionne ("focus")
W/A/S/D: Tourner camera autour de l'entite seletionnee ("orbit")
Q/E: Rapprocher ou eloigner la camera ("dolly")
R: Activer/Desactiver le mode lancer de rayon
F1: Afficher ou cacher l'interface graphique
F10: Recharger tous les shaders
Global Options
Background color
r 37
g 38
b 38
a 255
Camera Light
Ambient Light
HSB Mode
RGB
r 51
g 51
b 51
a 255
Apply Filter
Create primitive
ellipse
rectangle
triangle
cubic curve
catmull curve
bezier surface
cube
sphere
tetrahedron
particles
model
spotlight
point light
directional light
Full Screen Filter
None
Motion Blur
Sharpen
Emboss
Emboss++
Color Fill
Lambert
Gouraud
Phong
Blinn Phong
Edit
Transform
radius 50
Material
Shader
None (Fixed Pipeline)
Color Fill
Lambert
Gouraud
Phong
Blinn Phong
B.Phong OffsetScaleGS
Lighting
shine 50
emmisive
r 0
g 0
b 0
a 1
diffuse
HSB Mode
RGB
r 100
g 100
b 140
a 255
ambient
r 0.2
g 0.2
b 0.2
a 1
specular
a 1
Presets
Blue Steel
Desert Sun
Golden Egg

8. Lancer de rayon

8.1 Intersection

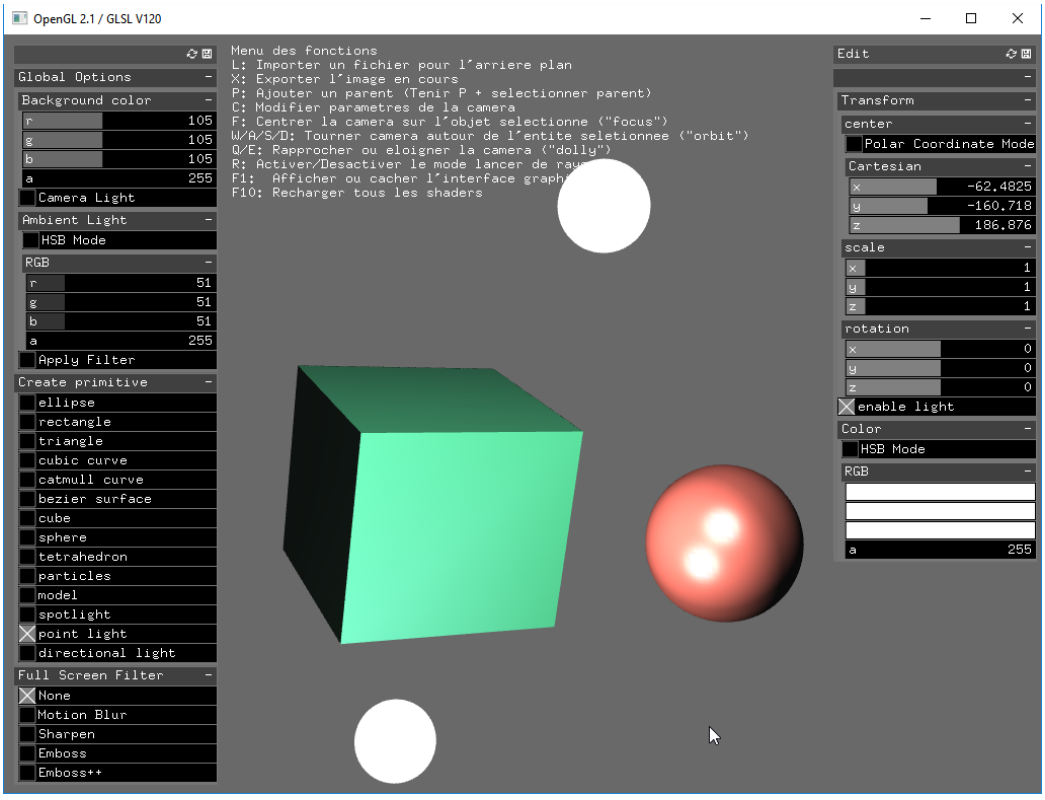
L'application offre le mode de calcul de l'illumination d'une scène par lancer de rayon. Le mode de lancer de rayon est disponible pour des scènes contenant les sphères et les cubes. Ce mode est activé ou désactivé en faisant la touche « R », et l'algorithme calcule pour chaque pixel l'intensité de couleur de ce pixel via l'intersection d'un objet avec un rayon de la caméra.

Afin d'accomplir ceci, j'ai d'abord défini une structure Ray, qui définit un rayon, en précisant son vecteur de direction et son point d'origine.

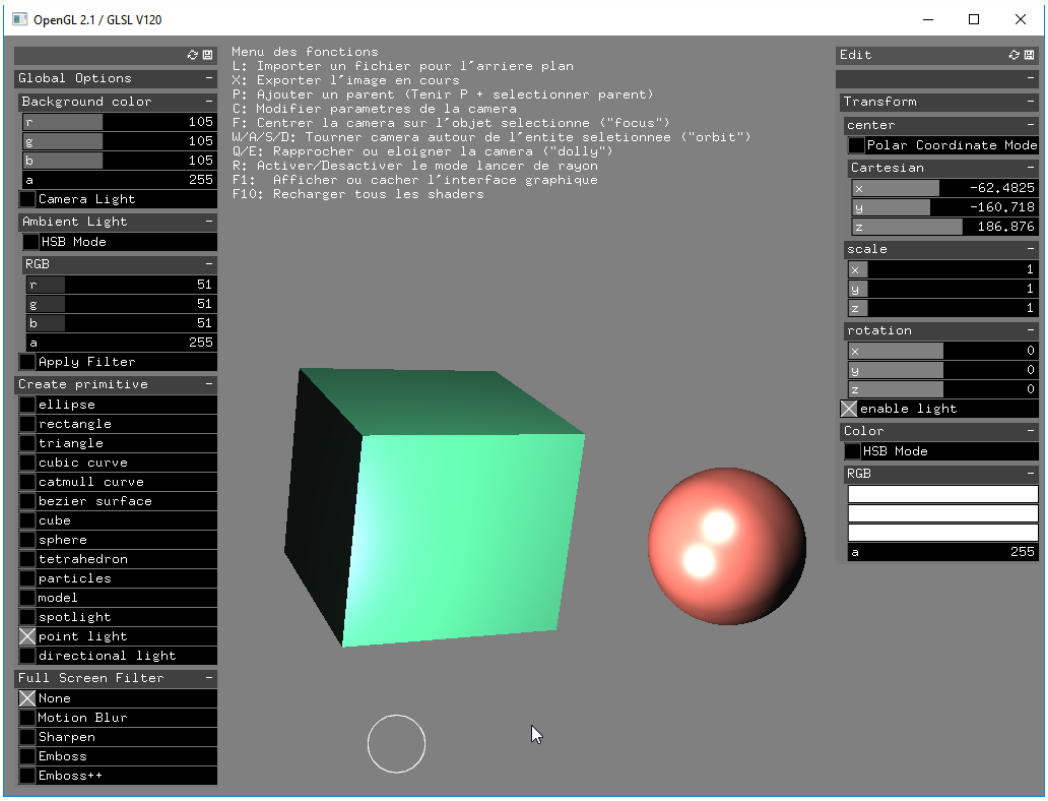
Par la suite, la classe SceneNode a été modifiée afin d'ajouter une fonction virtuelle getRayIntersection, qui par défaut retourne aucune intersection. Seules les classes Sphere et Cube ont été modifiées afin d'offrir une implémentation de la fonction getRayIntersection, qui calcule s'il y a intersection avec un rayon défini. Cette fonction retourne un booléen.

Finalement, dans ofApp, la fonction drawRayTraceScene effectue le lancer des rayons partant de chaque pixel de l'écran et vérifie via les fonctions getRayIntersection si une intersection a lieu. S'il y a intersection, les sources lumineuses éclairant cet endroit sont considérées dans le modèle d'illumination Blinn-Phong qui est utilisé pour calculer la valeur du pixel. La valeur du pixel est ensuite conservée dans un buffer. Lorsque tous les pixels ont été vérifiés, le contenu du buffer est conservé dans une texture, qui est affichée à l'écran. Ce mode offre un rendu de haute qualité, avec une très belle réflexion spéculaire, mais ce mode est très couteux en temps car chaque pixel doit être vérifié.

Mode lancer de rayon désactivé :



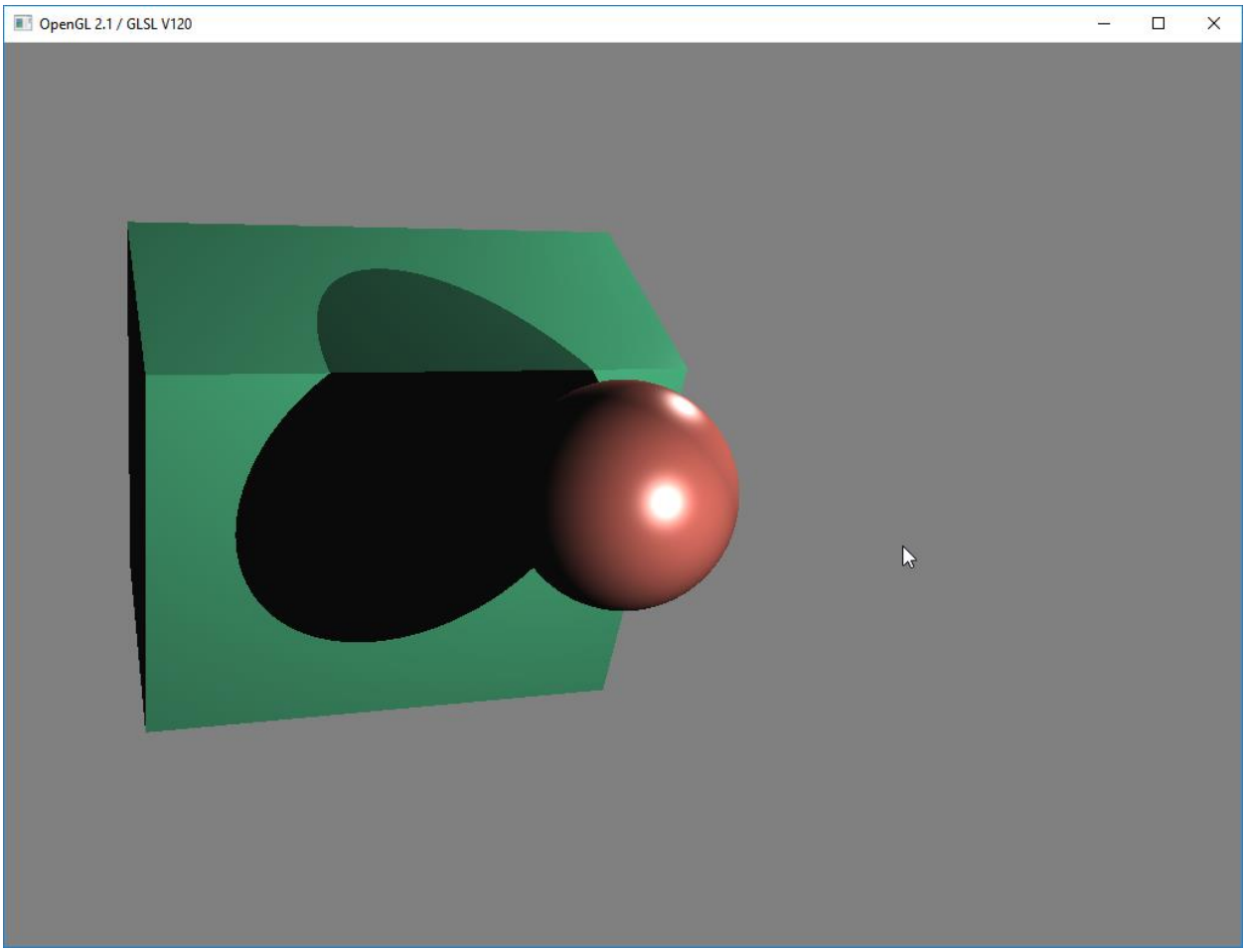
Mode lancer de rayon activé :



8.4 Ombrage

L'ombrage dans la scène est considéré dans les calculs de lancer de rayon. À l'origine, la couleur des pixels est fixée à noir. S'il y a illumination provenant d'une source lumineuse à un endroit, la contribution est ajoutée à la valeur du pixel. Ceci est répété pour toutes les lumières dans la scène. Par contre, s'il y a obstruction entre le point d'intersection et une source lumineuse, la contribution de cette dernière est ignorée. Une région ombragée apparaît donc sur la surface de l'objet dans ce cas.

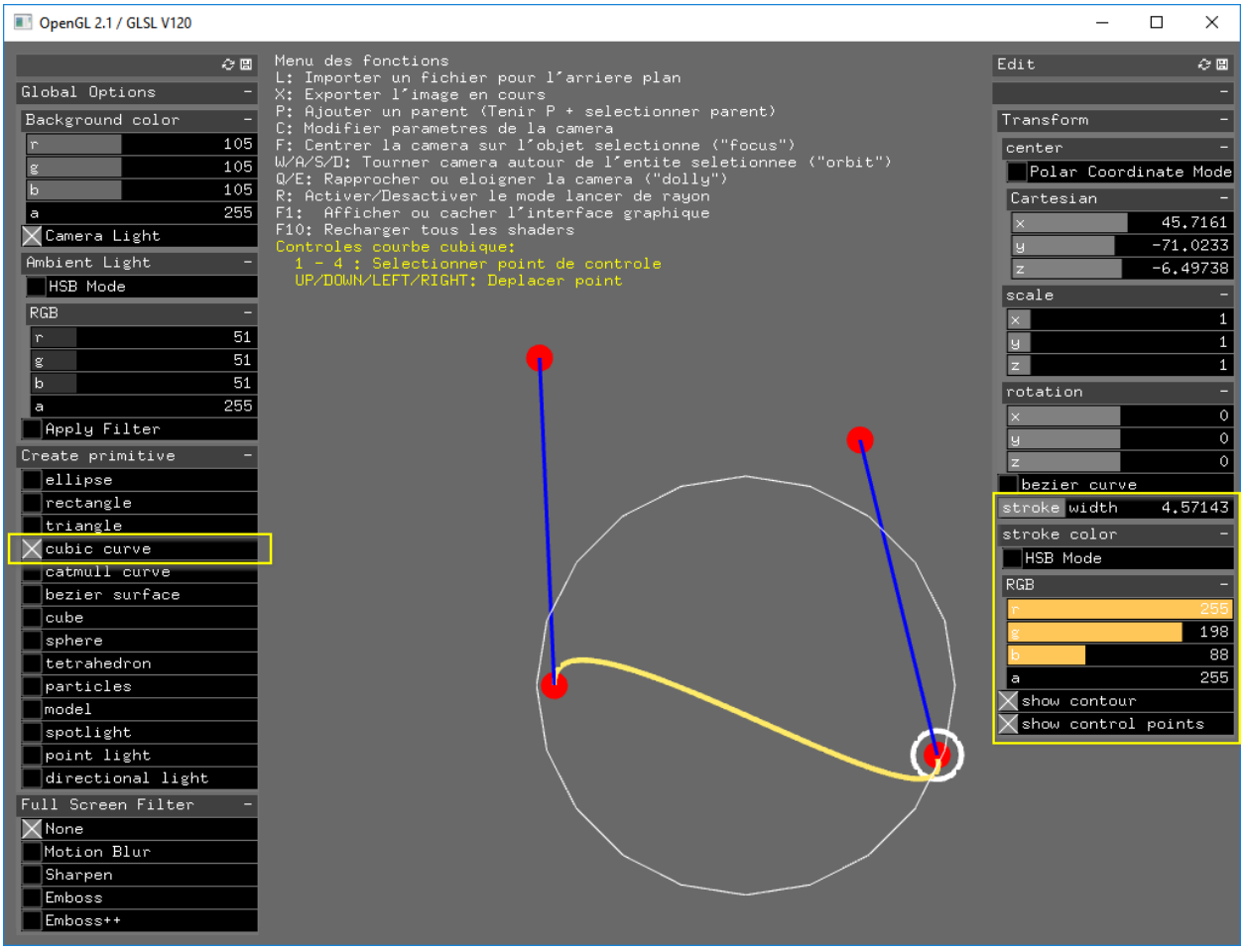
Ombrage en mode lancer de rayon :

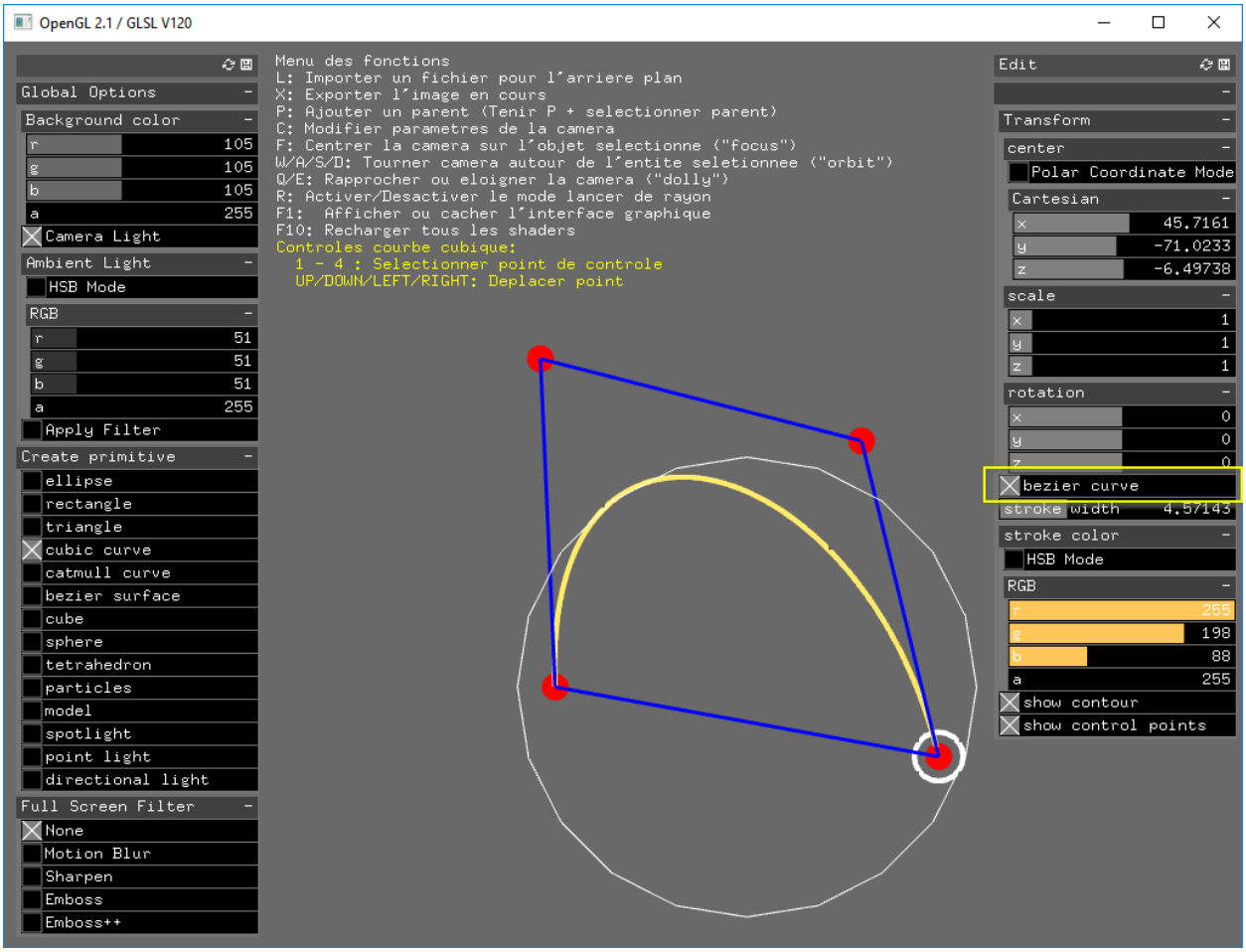


9. Topologie

9.1 Courbe cubique

Le menu de création de primitives offre la possibilité de créer une une courbe cubique de type Hermite par défaut. Cette courbe offre quatres points de contrôles. Ces derniers peuvent être sélectionnés en utilisant les touches 1 à 4, et peuvent être déplacés à l'aide des flèches. Il est possible d'enlever l'affichage des tangentes et des points de contrôles, ainsi que de convertir la courbe en une courbe Bézier cubique. La classe CubicCurve implémente les fonctions mathématiques pour l'affichage des courbes. CubicCurve hérite de SceneNode, ce qui signifie que la courbe peut être modifiée, déplacée, agrandie, etc tels que les autres objets de la scène.



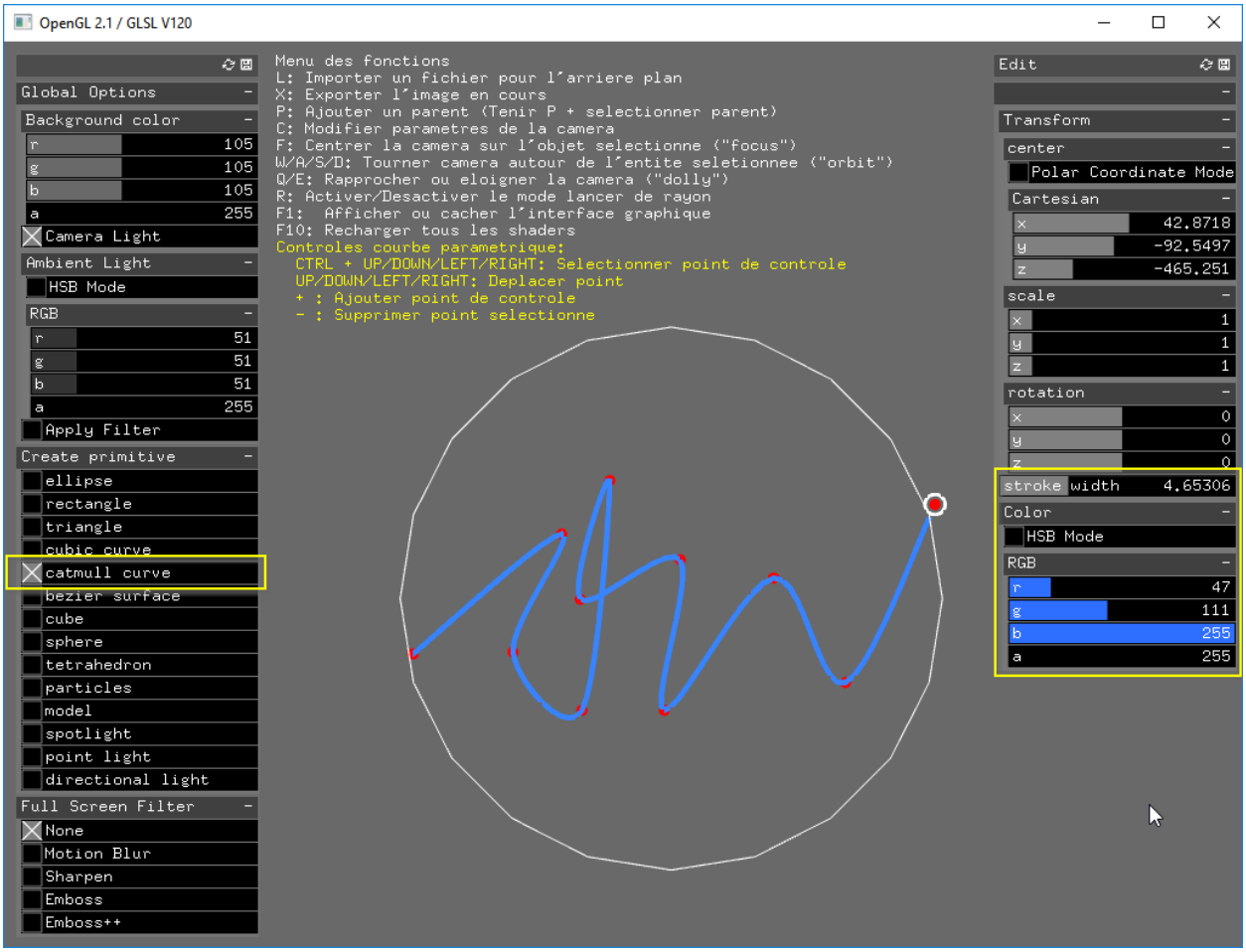


9.2 Courbe paramétrique

Le menu de creation de primitive offre aussi la possibilité de créer une courbe paramétrique de type Catmull-Rom. Par défaut, la courbe est créée avec 10 points de contrôles.

Il est possible de sélectionner un point de contrôle en faisant CTRL + UP/DOWN/LEFT/RIGHT, et de le déplacer à l'aide des flèches. Il est aussi possible d'ajouter un point de contrôle additionnel grâce à la touche «+ », ou d'enlever le point de contrôle sélectionné avec la touche « - ». La classe ParamCurve hérite de SceneNode et donc tout objet ParamCurve peut être déplacé, agrandi et modifié.

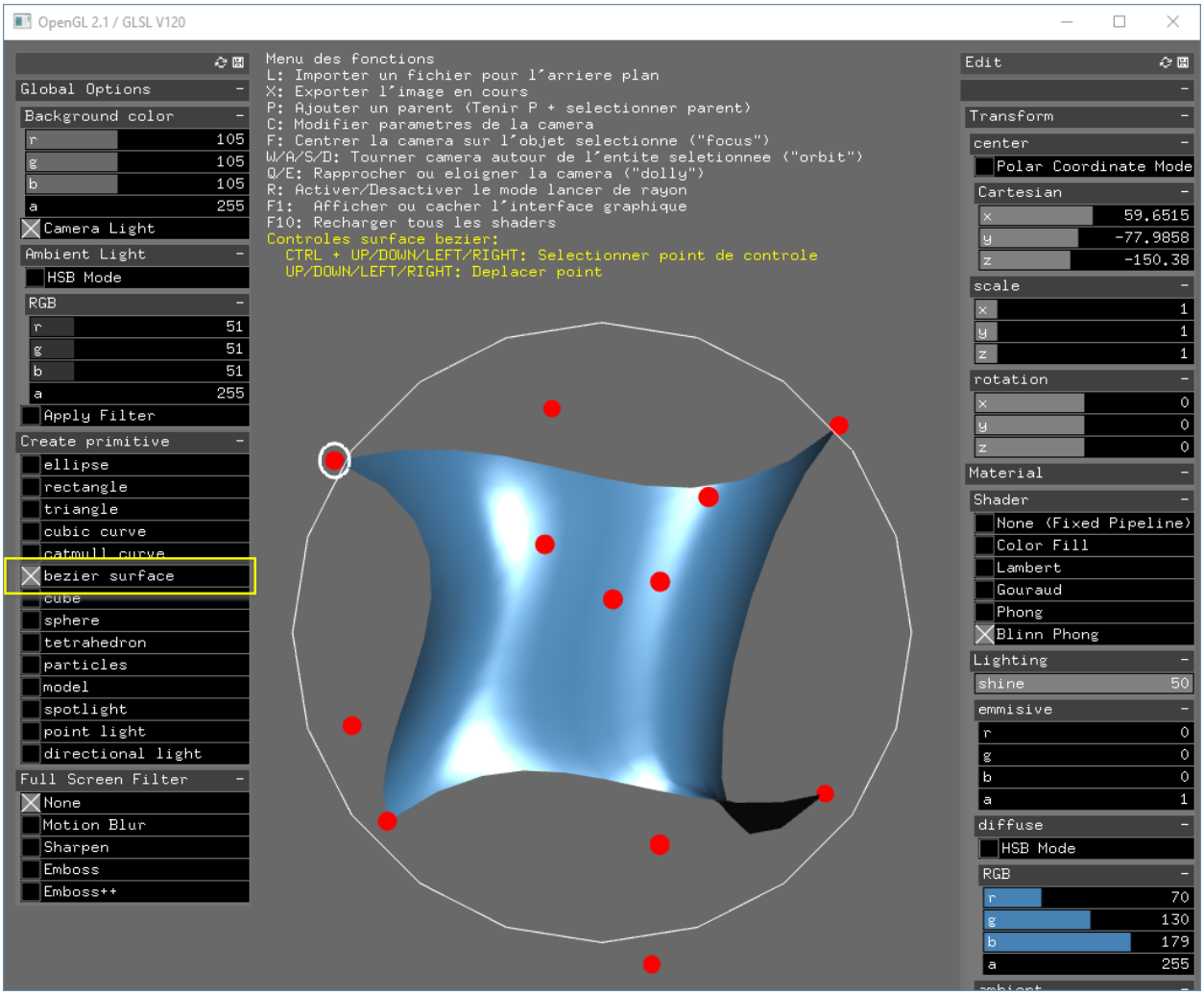
Courbe Catmull-Rom:



9.3 Surface paramétrique

L'application offre aussi la possibilité de rendre une surface Bézier bicubique avec 16 points de contrôles. Il est possible de sélectionner un point de contrôle en faisant CTRL + UP/DOWN/LEFT/RIGHT, et de le déplacer à l'aide des flèches. La classe BezierSurface hérite de SceneNode et donc tout objet BezierSurface peut être déplacé, agrandi et modifié.

Surface Bezier :



10. Techniques de rendu

10.1 Effet de relief

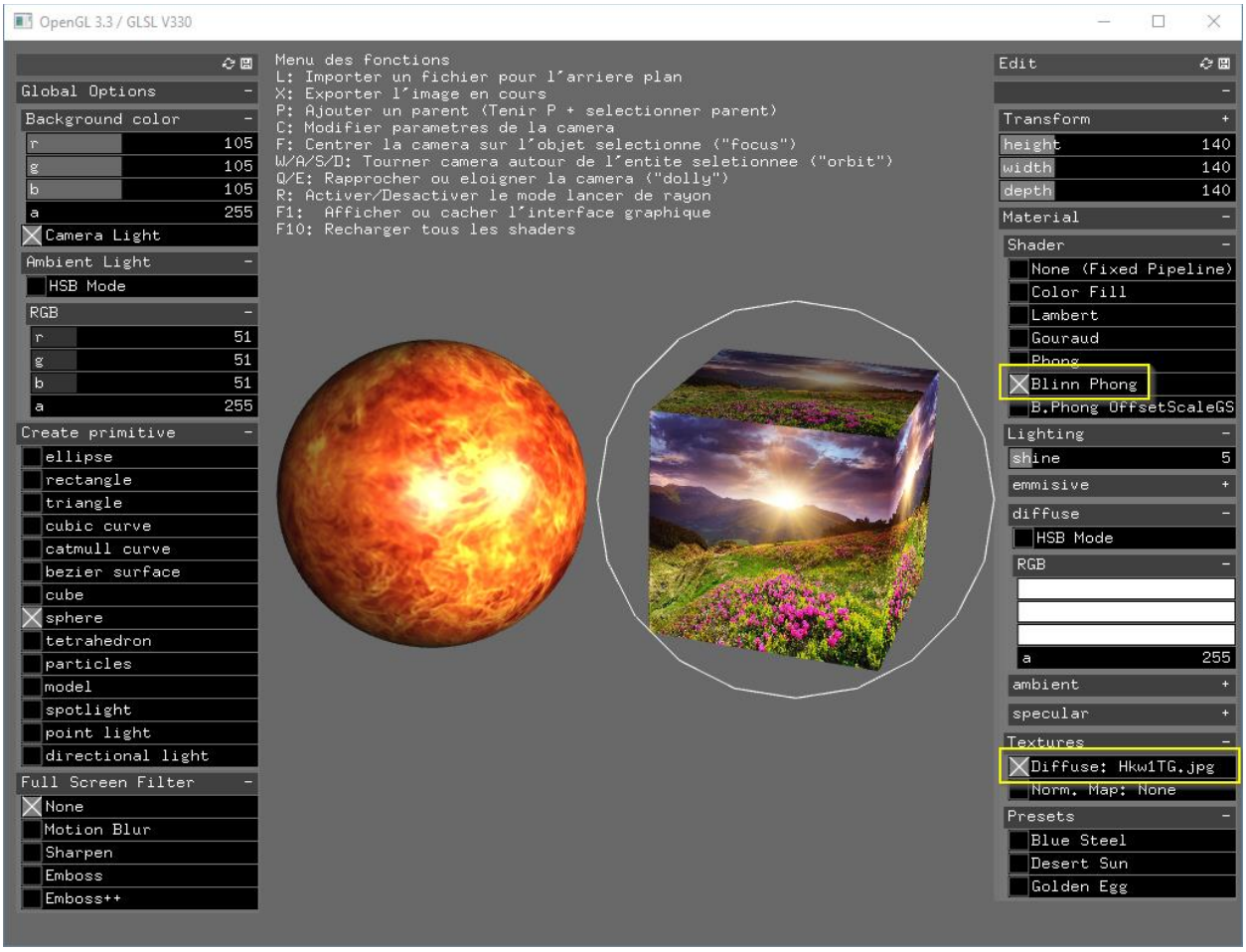
L'application permet de simuler un effet de relief sur un cube grâce à la technique de normal mapping. La technique consiste à utiliser une texture afin de moduler l'orientation de la normale. Afin d'implémenter cet effet, il était nécessaire d'ajouter la possibilité d'appliquer une texture diffuse à un objet. Ensuite, l'option d'utiliser une texture afin de modifier la normale a été ajoutée. Un sous menu nommé « Textures » de Material est présenté à l'usager et offre ces deux options. Lorsque l'usager sélectionne l'option d'ajouter une texture diffuse, une fenêtre de dialogue apparaît et permet à l'usager de choisir un fichier pour la texture. Le nom du fichier apparaît ensuite dans le menu. De même, lorsque l'usager choisit normal map du menu Textures, une fenêtre Windows lui permet d'aller sélectionner la mappe voulue.

L'application d'une texture diffuse est possible pour les objets de type Cube et Sphere. Pour réaliser ceci, la structure des sommets a été modifiée afin d'ajouter des coordonnées de texture et un vecteur pour la tangente. La classe Cube s'occupe d'assigner les coordonnées de texture aux sommets et calcule les tangentes pour chacun, ces données sont ensuite ajoutées au VBO.

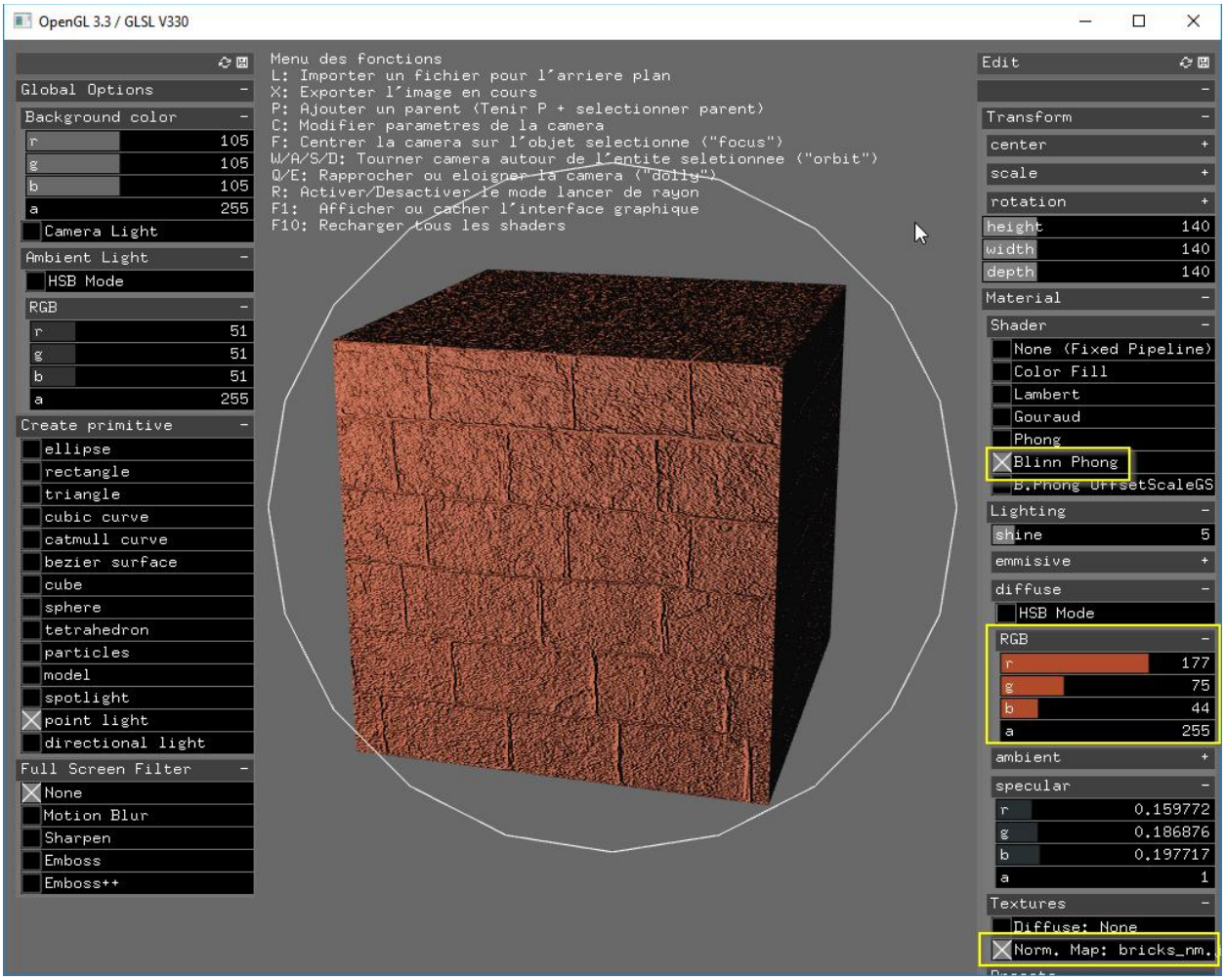
Par la suite, la classe Material a été modifiée afin d'ajouter l'envoi des uniforms associés au matériau au shader selon le cas d'une texture diffuse ou d'une normal map.

Finalement, les shaders de Blinn-Phong ont été modifiés, et plus particulièrement le fragment shader, afin de tenir compte de la texture diffuse dans le calcul de la couleur du pixel, et de modifier la normale si une normal map est utilisée.

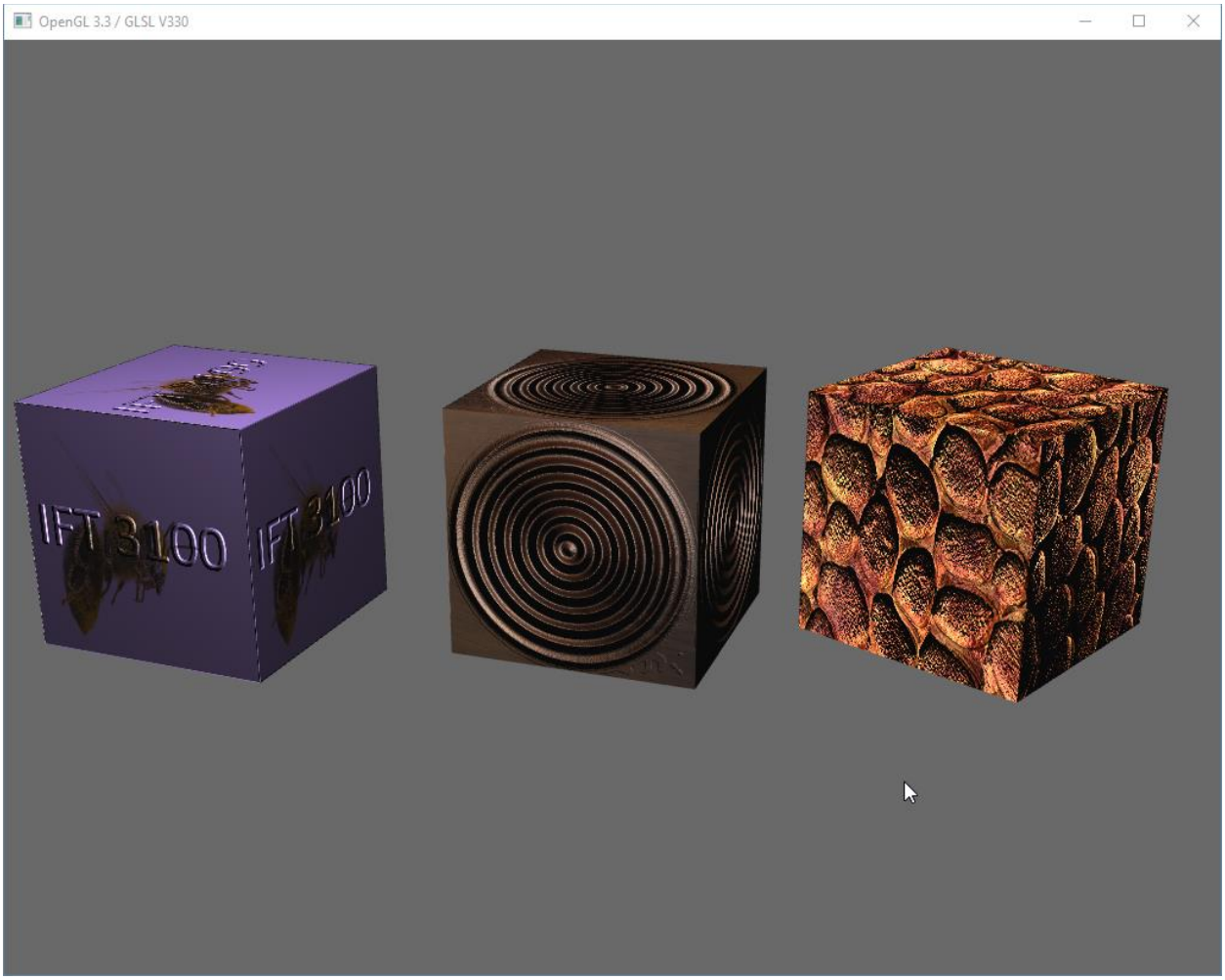
Texture diffuse :



Normal map :


OpenGL 3.3 / GLSL V330
Menu des fonctions
L: Importer un fichier pour l'arriere plan
X: Exporter l'image en cours
P: Ajouter un parent (Tenir P + selectionner parent)
C: Modifier parametres de la camera
F: Centrer la camera sur l'objet selectionne ("focus")
W/A/S/D: Tourner camera autour de l'entite seletionnee ("orbit")
Q/E: Rapprocher ou eloigner la camera ("dolly")
R: Activer/Desactiver le mode lancer de rayon
F1: Afficher ou cacher l'interface graphique
F10: Recharger tous les shaders
Global Options
Background color
r 105
g 105
b 105
a 255
Camera Light
Ambient Light
HSB Mode
RGB
r 51
g 51
b 51
a 255
Create primitive
ellipse
rectangle
triangle
cubic curve
catmull curve
bezier surface
cube
sphere
tetrahedron
particles
model
spotlight
point light
directional light
Full Screen Filter
None
Motion Blur
Sharpen
Emboss
Emboss++
Edit
Transform
center
scale
rotation
height 140
width 140
depth 140
Material
Shader
None (Fixed Pipeline)
Color Fill
Lambert
Gouraud
Phong
Blinn Phong
Lighting
shine 5
emmisive
diffuse
HSB Mode
RGB
r 177
g 75
b 44
a 255
ambient
specular
r 0.159772
g 0.186876
b 0.197717
a 1
Textures
Diffuse: None
Norm. Map: bricks_nm.

Normal map et texture diffuse combinées :



10.3 Effet plein fenêtre

L'application offre plusieurs filtres de convolution pour appliquer un effet visuel sur toute la surface de la fenêtre. Les filtres disponibles sont Motion Blur, Sharpen, Emboss et Emboss ++ (Emboss avec un effet plus marqué). La fonction applyFullScreenFilter dans ofApp est responsable d'appliquer les filtres sur la fenêtre en cours. Cette fonction calcule pour chaque pixel la contribution des pixels avoisinants selon une matrice filtre.

Image originale :

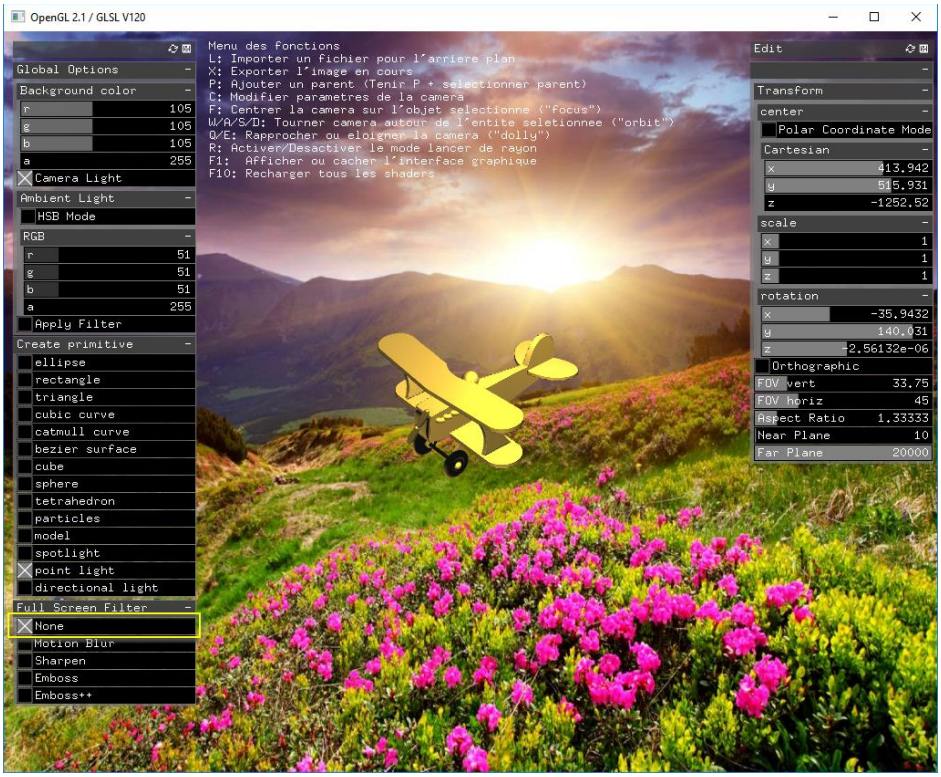


Image avec filtre « Motion Blur » :

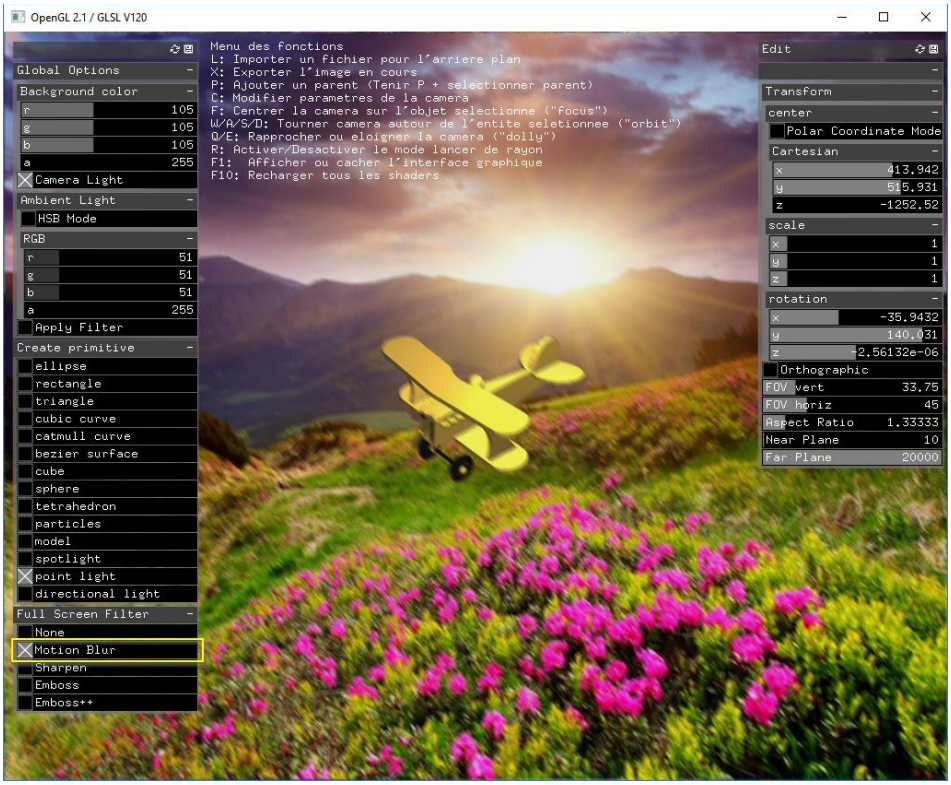


Image avec filtre « Sharpen » :

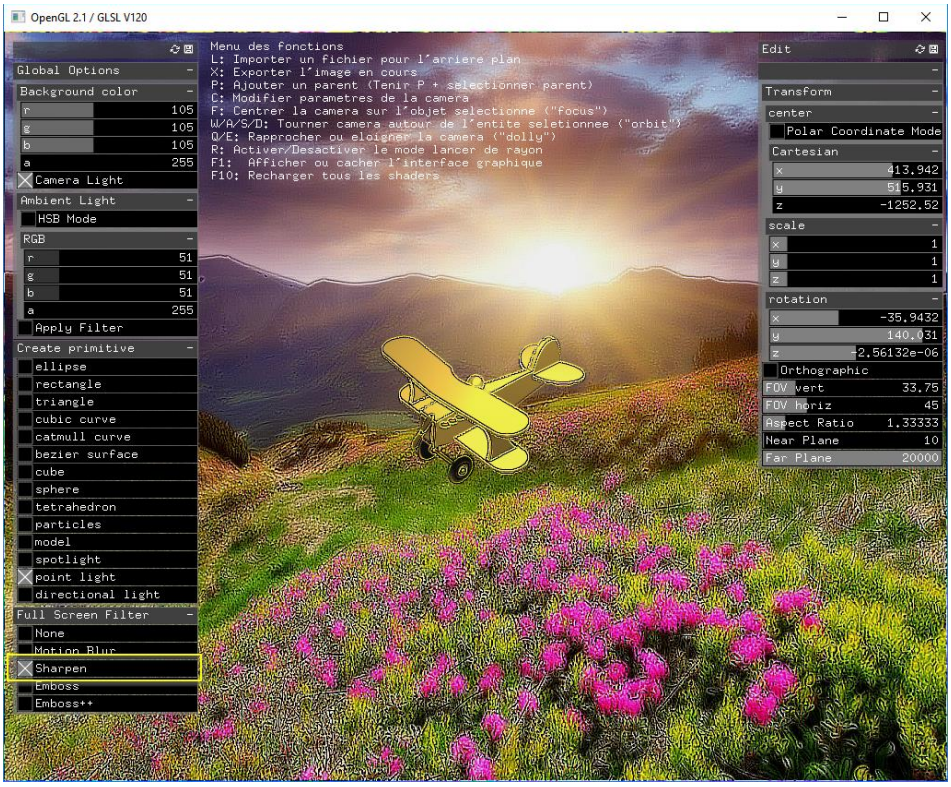


Image avec filtre « Emboss » :

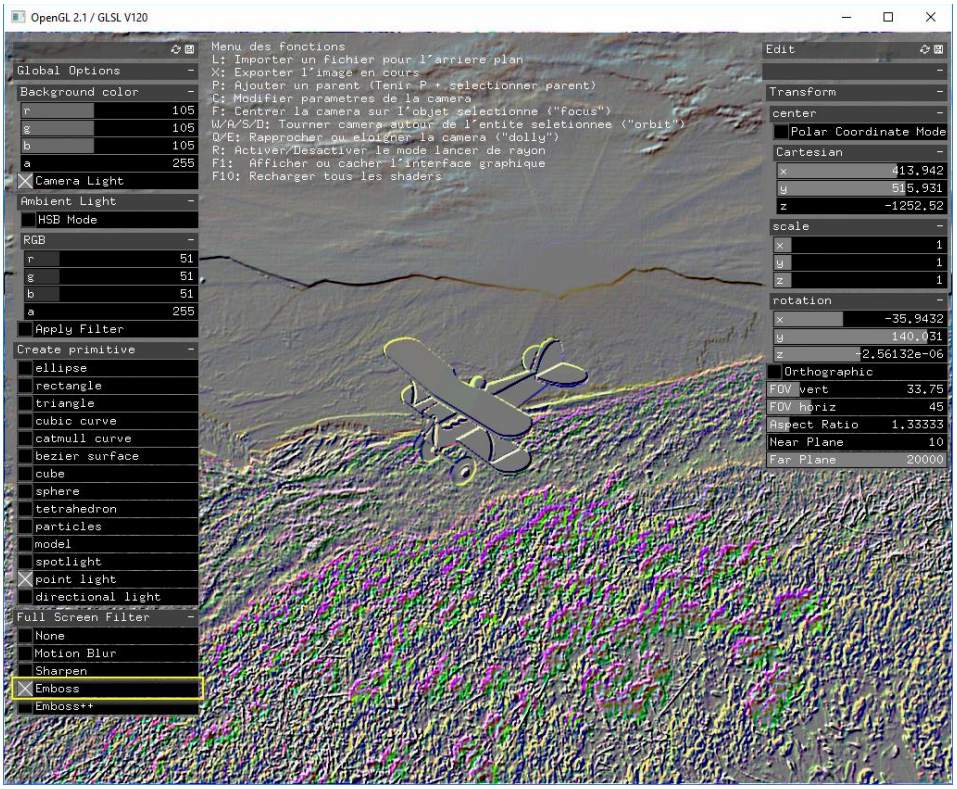
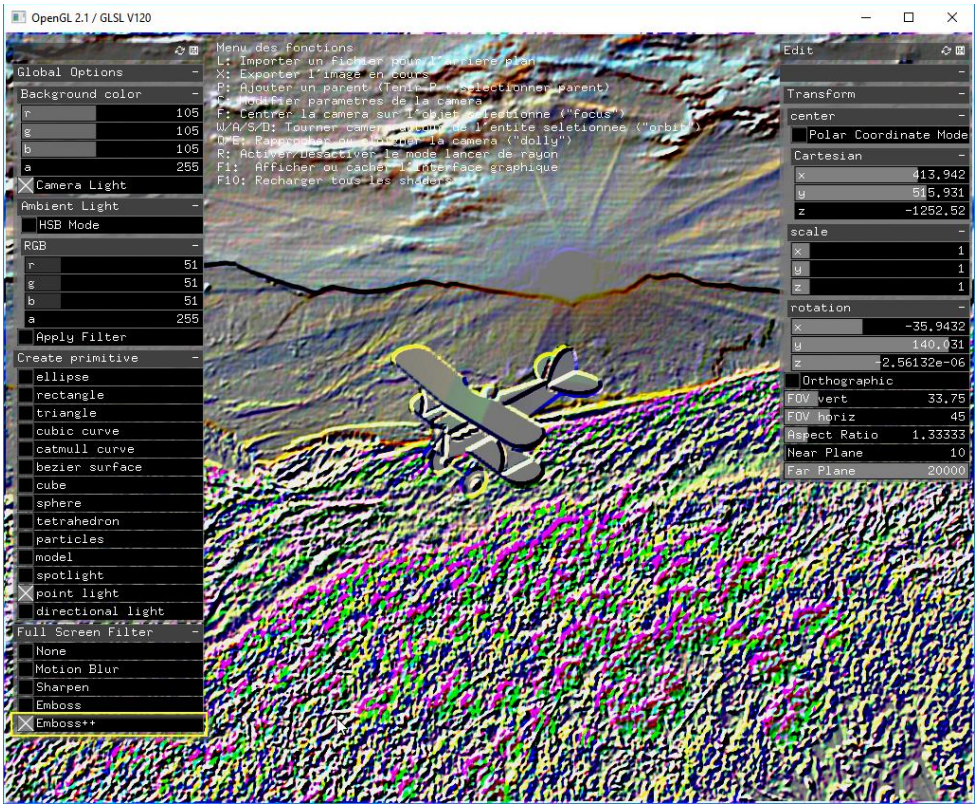


Image avec filtre Emboss plus marqué (« Emboss ++ ») :



6. Ressources

Algorithmes de conversion de modes de couleur :

http://www.rapidtables.com/convert/color/rgb-to-hsv.htm

Modèles 3D :

Toy plane: http://www.oyonale.com/modeles.php?page=56

Stormtrooper: http://tf3dm.com/download-page.php?url=puo-4041-7174

Statue of Liberty: http://tf3dm.com/download-page.php?url=statue-of-liberty-73656

Images:

Particles (fire): http://blueblots.com/textures/free-fire-texture/

Abeille : https://www.friendsofthehoneybee.com/learn-about-bees/honey-bee-key-facts/

Briques : http://ogldev.atspace.co.uk/www/tutorial26/tutorial26.html

Spirale : http://cpetry.github.io/NormalMap-Online/

Pierres : http://cpetry.github.io/NormalMap-Online/

Exemples de code consultés :

Openframeworks – exemple Swarm

Openframeworks – exemple advanced3DExample

Philippe Voyer – exemples présentés en classe

GLSL :

https://en.wikipedia.org/wiki/OpenGL_Shading_Language

https://www.khronos.org/opengl/wiki/GLSL_:_common_mistakes

Courbe Catmull-Rom :

https://www.cs.cmu.edu/~462/projects/assn2/assn2/catmullRom.pdf

https://github.com/larsberg/ofxSimpleSpline/blob/master/src/ofxSimpleSpline.cpp

https://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline

Surface Bezier :

https://en.wikipedia.org/wiki/B%C3%A9zier_surface

https://www.gamedev.net/resources/_/technical/graphics-programming-and-theory/bezier-curves-and-surfaces-r1808

http://www.programming-techniques.com/2012/03/bezier-curves-and-bezier-surfaces.html

Shader de geométrie :

https://www.khronos.org/opengl/wiki/Geometry_Shader

https://www.khronos.org/opengl/wiki/Geometry_Shader_Examples

https://www.khronos.org/opengl/wiki/Geometry_Shader_Examples#Short_Example

http://stackoverflow.com/questions/10031530/passing-varying-variables-through-geometry-shader

Filtre (convolution) :

http://lodev.org/cgtutor/filtering.html

https://wallpapersafari.com/flower-field-wallpaper/

Ray tracing :

Akenine-Möller, Tomas, Haines, Eric, Real-Time Rendering, Second Edition, AK Peters, 2002.

Texture & Normal Mapping:

https://www.khronos.org/opengl/wiki/Sampler_(GLSL)

http://ogldev.atspace.co.uk/www/tutorial26/tutorial26.html

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/

http://fabiensanglard.net/bumpMapping/index.php

7. Présentation

Je m'appelle Nathalie Chang et mon cheminement professionnel n'est pas typique. Je possède un baccalauréat en génie électrique, ainsi qu'une maîtrise en génie biomédicale. Je travaille dans le réseau de la santé en tant qu'ingénieure biomédicale et gestionnaire de projet d'équipements médicaux depuis environ 10 ans. J'ai décidé de faire un certificat en informatique et de rediriger ma carrière afin de relever de nouveaux défits. J'avais rédigé mon projet de recherche de maîtrise en Matlab, et donc la programmation m'a toujours grandement intéressée. Je suis très intéressée par l'infographie, car je suis fascinée par la technologie qui est derrière les jeux vidéo, les effets spéciaux cinématographiques, ainsi que toutes les applications (médicales, etc) de simulation 3D.

Ce projet est certainement le projet ayant le plus haut niveau de complexité que j'aille produit à ce jour. J'ai appris énormément, et parfois difficilement, sur la conception d'une application d'une telle envergure, le plus grand défi étant parfois l'organisation de mon code. J'ai aussi beaucoup apprécié l'opportunité de mettre en pratique les concepts infographiques que nous avons appris en classe. J'y ai mis énormément d'effort et je suis très fière du résultat!