1. Sommaire

Le projet a pour but de développer et d'approfondir nos connaissances dans le domaine de l'infographie. Ainsi, le projet a avant tout un aspect de prototype. Ce qui implique qu'avant tout le projet mise à démontrer la faisabilité ainsi que la capacité de mise en place de certaines techniques d'infographie. L'interface et l'architecture ne sont présents que pour des raisons de facilité d'accès et de développement aux fonctionnalités implémentées.

De cette manière, il est possible de se concentrer sur les techniques spécifiques au domaine de l'infographie. La première partie du projet se divise en deux sections distinctes: Le 2D et le 3D.

Dans la section 2D on explore les outils de dessin et les formes vectorielles. On crée des lignes, des cercles, des ellipses, des rectangles et du texte à l'aide des compétences acquises lors du cours d'infographie et lors de recherches personnelles.

Par exemple, pour la création de lignes et de rectangle on utilise la rastérisation de triangle d'OpenGL, pour les cercles et les ellipses on utilise un shader et pour le texte on utilise des textures de police de caractères.

Pour la section 3D on vise davantage à expérimenter les géométries ainsi que leur transformation. On crée différents types d'objets: des sphères et des cubes. On importe des modèles géométriques et on les affiche à l'écran avec des caméras selon différents points de vue. On utilise une hiérarchie d'objet et on joue avec leurs composantes de position, rotation et proportion afin d'expérimenter les matrices de transformation.

Bref, ce projet est une porte d'entrée à l'infographie, un bac à sable. C'est une opportunité d'apprendre et d'approfondir les bases de l'affichage 2D et 3D et de mieux comprendre les défis que font face les programmes graphiques.

2. Interactivité

2.1. Menu d'accueil

Le menu d’accueil présente trois options qui s’offrent à nous. Pour sortie de l’application, l’utilisateur peut appuyer sur « Échap » en tout temps. Chaque menu d’interface peut être bougé.

2.2. 2D editor

Le 2D editor nous permet de créer des objets 2D comme des images et des lignes

On peut modifier la couleur d’arrière-plan et de remplissage des primitives créées. On peut sélectionner le mode HSV si on préfère aussi.

2.2.1. Outils

Chacun des outils peut être sélectionné dans le menu « Properties » à l’aide du radio bouton correspondant.

2.2.2. Point

On peut créer un point en sélection l’option point et avec un click sur l’arrière plan.

2.2.3. Rectangle

On peut créer un rectangle en appuyant à un endroit et en relâchant à un autre. Il est également possible de faire un carré en appuyant sur la touche «shift» gauche.

2.2.4. Ellipse

On peut créer une ellipse en appuyant à un endroit et en relâchant à un autre. Il est également possible de faire un cercle en appuyant sur la touche «shift» gauche.

2.2.5. Texte

On peut créer du texte en appuyant à un endroit et en appuyant sur des touches du clavier.

Il est possible de changer la police d'écriture en changeant le champ « font path », en donnant les bonnes mesures de cellules (« font width» et « font heigh ») et en appuyant sur le bouton «Change font».

2.2.6. Image

Il est possible de créer une image à un endroit précis en sélectionnant l'outil «image» et en appuyant n'importe où sur l'arrière-plan. Il est possible de changer l'image en changeant le champ « texture path» en appuyant sur le bouton change texture.

2.3. 3D editor

Le 3D editor nous permet de créer et de modifier des objets 3D comme des sphères et des cubes.

On peut déplacer et «docker» les différents menus de l'éditeur.

On peut créer des cubes, des sphères ou des caméras avec les boutons «Cube»,«Sphere», «Camera» dans le menu «Create things».

On peut visualiser les éléments de la scène dans l'arborescence sous «Elements»

On peut sélectionner un objet avec 1 click lorsque le sous-arbres de l'objet est ouvert. On peut sélectionner plusieurs objectifs en appuyant sur «CTRL» gauche et en sélectionnant sur plus d'un objet.

On peut désélectionner un objet avec un click sur «Elements»

On peut visualiser les objets sélectionnés sous le bouton «Delete»

On peut supprimer des éléments avec le bouton «Delete»

On peut activer la rotation ou la désactiver. Il y a certains problèmes avec la rotation. (Elle peut causer un déplacement infini d'un enfant si la composant de position de ce parent est différente de 0)

On peut modifier la position, la rotation et la proportion des objets sélectionnés.

2.4. 3D Features Scene

Le 3D Features Scene nous présente une scène avec plusieurs éléments créés préalablement. L'objectif étant de présenter rapidement certaines fonctionnalités ainsi que de montrer les fonctionnalités qui ne sont pas

possibles de faire dans le 3D editor. On y présente entre autres le shader de géométrie (gazon).

Tout ce que l’on peut faire c’est se déplacer avec «WASD», «Espace» et «CTRL».

3. Technologie

On utilise OpenGL moderne pour faire le projet puisque c'est la version standard d'OpenGL maintenant. De plus, on avait déjà un début de projet.

Afin d'accéder aux fonctionnalités d'OpenGL moderne on doit utiliser GLEW qui nous offre la définition des fonctions OpenGL. C'est mandatoire.

On utilise également GLFW pour ce qui est de la création du contexte OpenGL et de la gestion des entrées.

De plus, pour nous faciliter l'implémentation de la caméra ainsi que des transformations des matrices des objets de la scène on utilise GLM. Il aurait pu être intéressant de programmer ces transformations, mais on a voulu mettre de l'énergie dans les choses qui sont un peu plus concret et visuel.

Pour ce qui est de l'importation et l'exportation des images on utilise STB_IMAGE. L'importation et l'exportation d'image ne sont vraiment un objectif d'apprentissage du projet cependant on a besoin pour faire ce que l'on veut faire. Donc on se facilite la vie avec l'utilisation de cette librairie

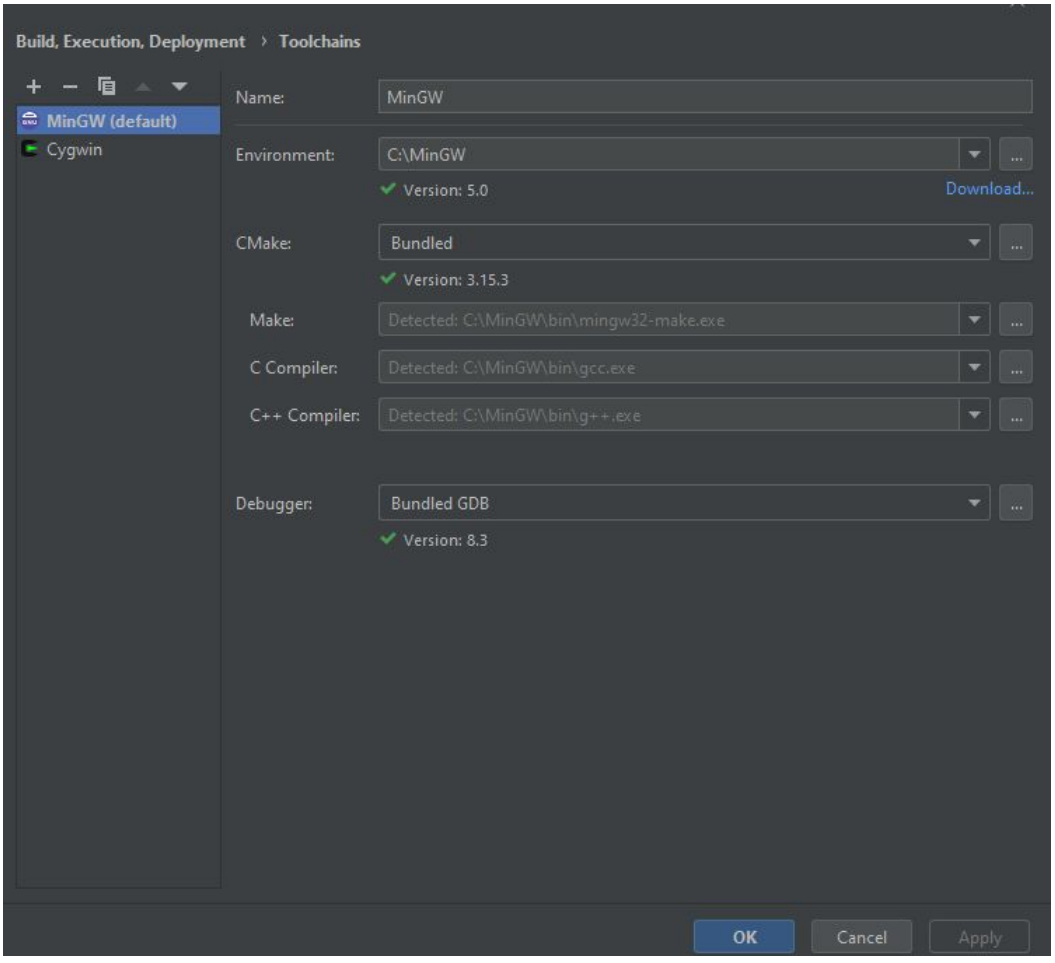
Finalement, pour ce qui est de l'interface on utilise ImGui. Au début, on voulait faire notre propre interface avec OpenGL, cependant parce que les critères fonctionnels n'accordaient aucun point à une interface programmée soi même, on a décidé de revoir nos plans.

4. Compilation

Pour compiler le programme ,il suffit d'installer MinGW sur une platforme Windows puis de lancer la compilation avec un IDE qui supporte CMake et MinGW . Nous avons utilisé Clion mais n'importe quel logiciel utilisant le mode de compilation CMake avec minGW serait suffisant(ex:Eclipse). Toutes les librairies nécessaires à la compilation son intégré dans le projet CMake.

Avec Clion:

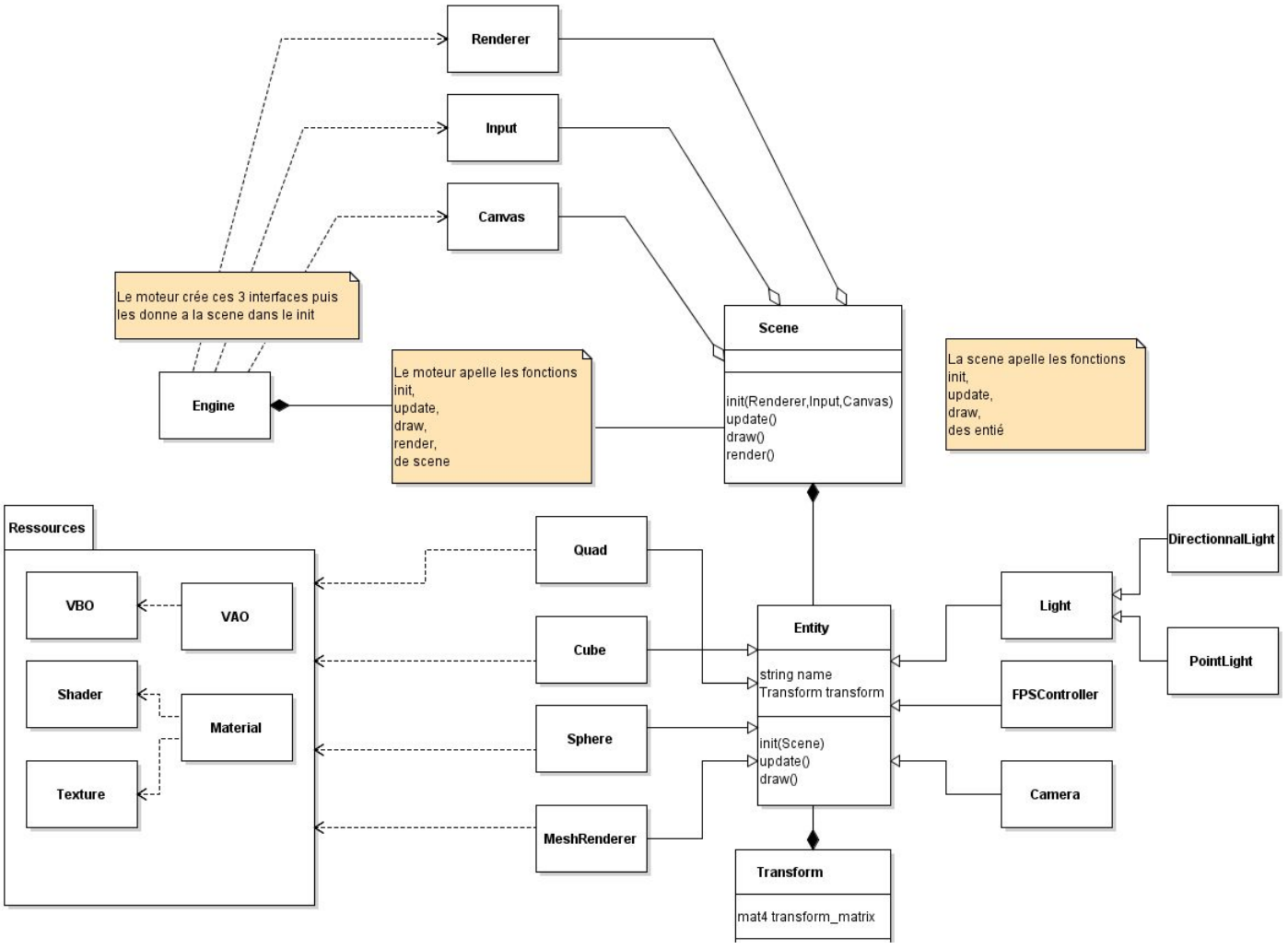
1. Installer minGW https://osdn.net/dl/mingw/mingw-get-setup.exe.Ne pas ooublier de mettre MinGW dans la variable d'environnement PATH.
2. Installer Clion https://www.jetbrains.com/clion/download/#section=windows
3. Ouvrir le projet.
4. File->Settings->Build,Execution,Deployment->ToolChain
5. S'assurer d'avoir MinGW bien installé et en défaut et appuyer sur Apply:



6. Ensuite il vous reste seulement à cliquer sur le bouton vert pour démarrer la compilation et l'exécution.

5. Architecture

Le program est organisée sous 2 couche:moteur et scene.La couche moteur s'occupe d'initialiser les objets nécessaires au rendu en plus de fournir des interface facile d'utilisation à la couche des scènes:le Renderer qui s'occupe des commande de rendu openGL,Input est pour les événement d'entrée clavier et souris,Canvas est une fenêtre virtuelle qui contient le framebuffer.La couche des scènes contient des entités qui on tous une position, une rotation et une taille.Certaine entités ont besoin de d'autre ressource comme un matériel(contient un shader et une ou des textures) et une mesh pour ce faire dessiner.Si celle si veule se faire dessiner ils appelle la fonction draw du Renderer avec un matériel, un transform et une mesh à chaque image.Dans notre projet il y a trois Scènes: l'éditeur 2D, l'éditeur 3D et une scène de démonstration 3D.



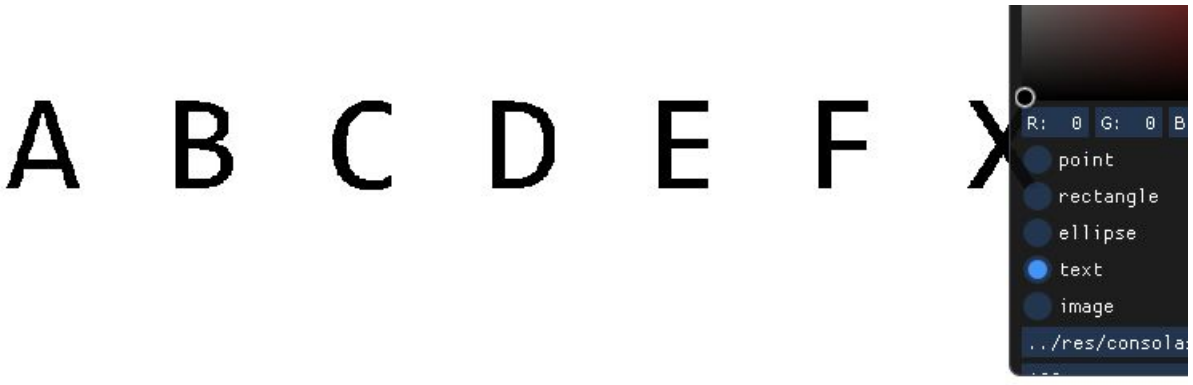
6. Fonctionnalités

1.1 Importation d'images

On utilise la librairie STB_IMAGE pour importer une image dans l'application. Après on convertir l'image en texture qu'on applique à un Quad.

1.3 Échantillonnage d'images

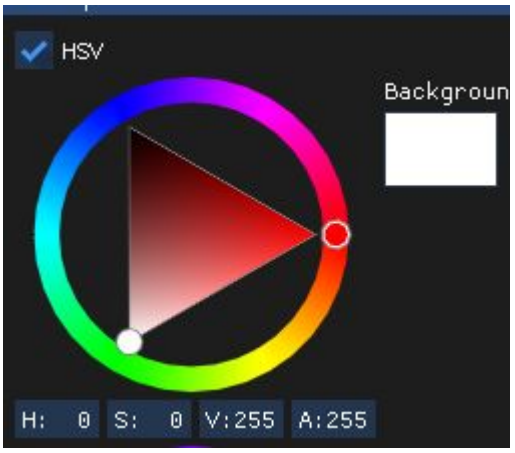
On utilise la librarie STB_IMAGE pour importer une texture créer avec Bitmap Font Builder Setup. La texture représente une police de caractères et est divisé en cellule. Chaque cellule contient une lettre. Après on créer un maillage constitué de plusieurs Quad un à côté de l'autre. On configure leurs UVS pour qui face référence à la case de la texture de police de caractères. Il y a un gros espacement entre les lettres parce qu'on ne fait pas attention à la largeur des lettres dans chaque cellule.

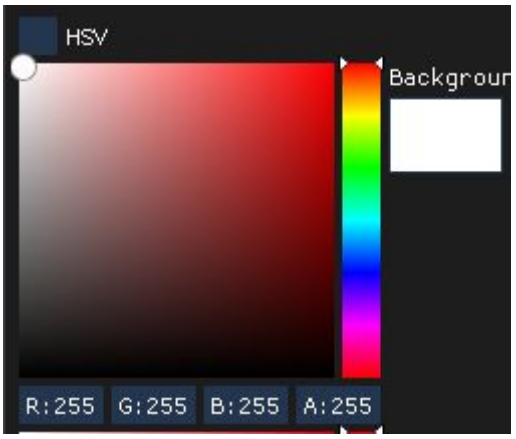


1.4 Espace de couleur

On utilise ImGui et il implémente déjà cette fonctionnalité. Tout ce qu'on avait à faire c'est de donner les bons drapeaux.

```
ImGui::ColorPicker4( label: "Background Color", &background_color.r,  flags: isHSV ? ImGuiColorEditFlags_DisplayHSV | ImGuiColorEditFlags_PickerHueWheel : ImGuiColorEditFlags_DisplayRGB)
```





2.1 Curseur dynamique

Pour les curseur dynamique, nous utilisons la librairie GLFW où il y a 6 choix de curseur standard: Hand,Arrow,IBeam,Crosshair,HorizontalResize et VerticalResize.Pour Créer un curseur il suffit d'appeler la méthode glfwCreateStandardCursor:

```
arrow_cursor = glfwCreateStandardCursor( shape: ARROW);
i_beam_cursor = glfwCreateStandardCursor( shape: IBEAM);
hand_cursor = glfwCreateStandardCursor( shape: HAND);
v_resize_cursor = glfwCreateStandardCursor( shape: HRESIZE);
h_resize_cursor = glfwCreateStandardCursor( shape: VRESIZE);
crosshair_cursor = glfwCreateStandardCursor( shape: CROSSHAIR);
```

Pour l'utiliser il suffit de faire la fonction glfwSetCursor:

```
void Input::setCursor(Input::CursorImage image) {
    switch (image) {
        case HAND:
            glfwSetCursor( window: window,  cursor: hand_cursor);
            break;
        case ARROW:
            glfwSetCursor( window: window,  cursor: arrow_cursor);
            break;
        case IBEAM:
            glfwSetCursor( window: window,  cursor: i_beam_cursor);
            break;
        case CROSSHAIR:
            glfwSetCursor( window: window,  cursor: crosshair_cursor);
            break;
        case HRESIZE:
            glfwSetCursor( window: window,  cursor: h_resize_cursor);
            break;
        case VRESIZE:
            glfwSetCursor( window: window,  cursor: v_resize_cursor);
            break;
    }
}
```

2.2 Outils de dessin

Pour ce qui est des outils de dessin, on a pu seulement implémenter la couleur d'arrière plan et la couleur de remplissage. La ligne de contour et la couleur de la ligne on a pas pu l'implémenter parce que on a pas de ligne de contour sur nos primitives.

2.3 Primitives vectorielles

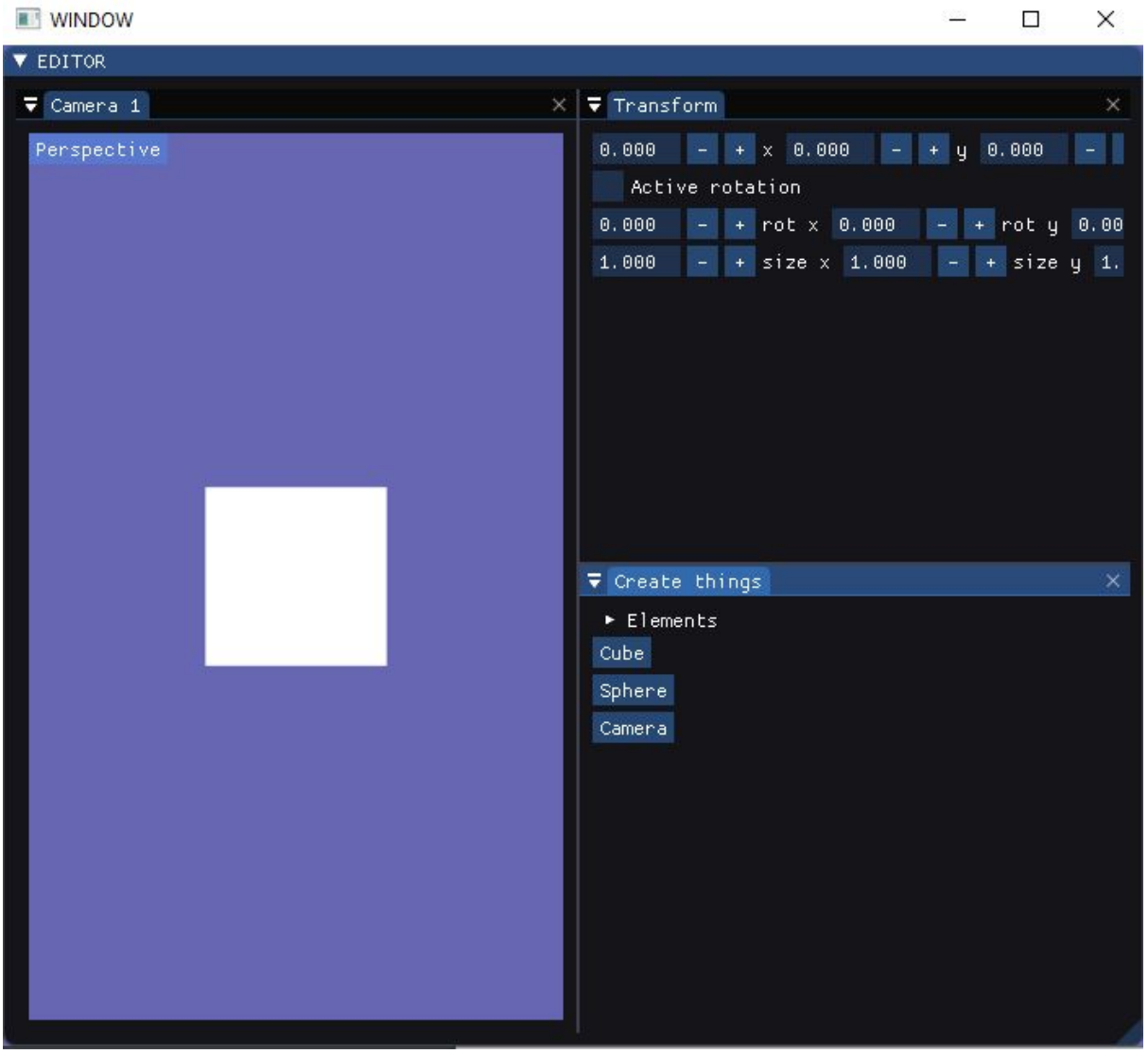
Les primitives vectoriels implémentés sont: Point, Rectangle, Ellipse. L'outil de Rectangle peut nous permettre de faire des carré parfait en appuyant continuellement sur la touche « Shift » en créant un rectangle. La même chose pour l'outil Ellipse excepté qu'on crée des circles. Bref, les trois primitives sont Point, Rectangle, Carré, Ellipse, Cercle. Pour faire, le point on instancie un cercle à la position du click de la souris. Le Rectangle est créer avec l'aide d'un Quad et un matériel bien standard. Le Carré est créer de la même manière. L'Ellipse et le cercle est créé à l'aide d'un shader.

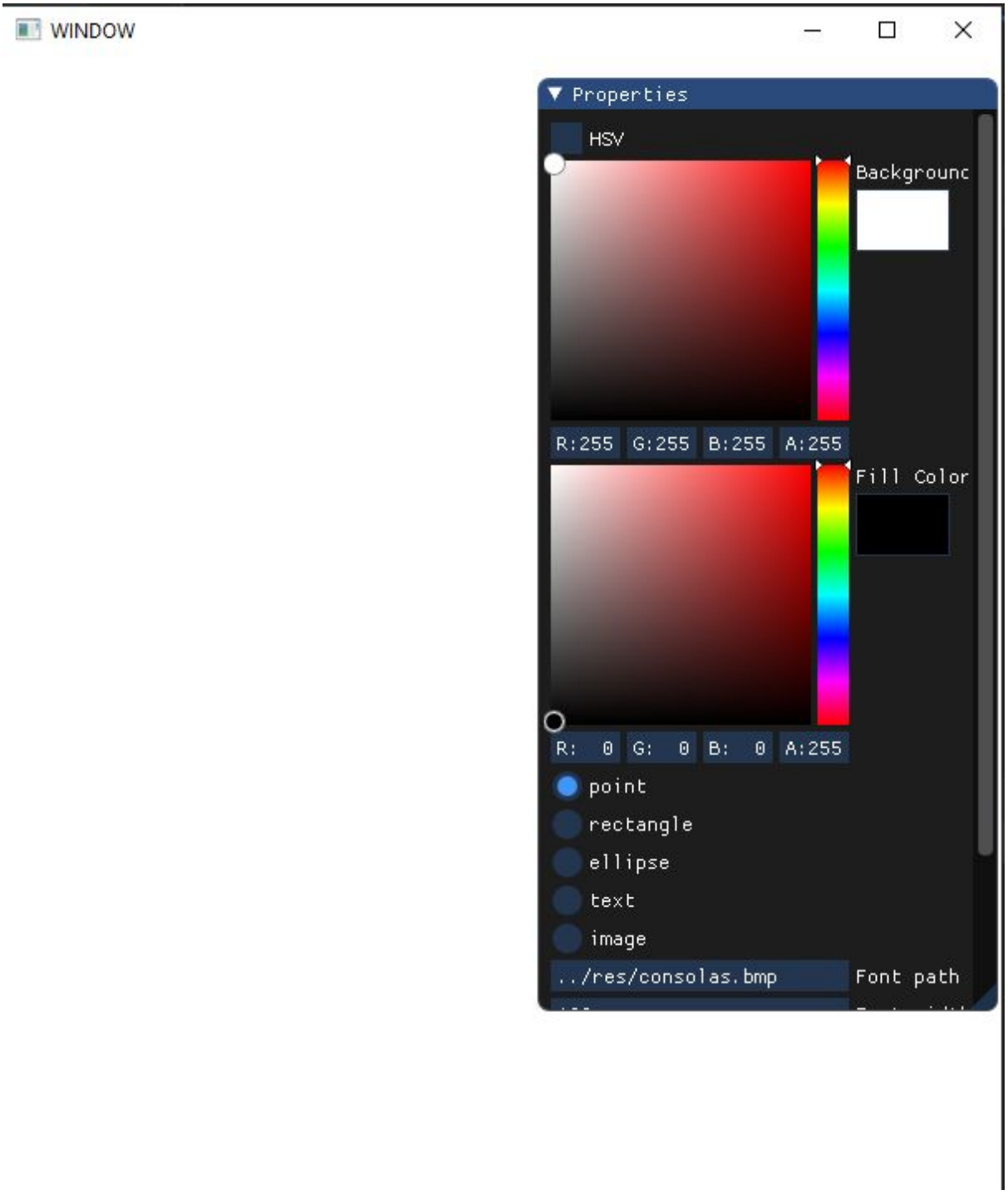
```
const Shader ELLIPSE_SHADER = Shader( vertexShader: "#version 330 core\n"
                                      "uniform mat4 projection;\n"
                                      "uniform mat4 transform;\n"
                                      "layout(location=0)in vec3 vertexPosition;\n"
                                      "layout(location=1)in vec2 vertexUv;\n"
                                      "out vec2 uv;\n"
                                      "void main()\n"
                                      "{\n"
                                      "    uv=vertexUv;\n"
                                      "    gl_Position=projection*transform*vec4(vertexPosition.xyz,1.);\n"
                                      "}",
                                       fragmentShader: "#version 330 core\n"
                                      "uniform vec4 material_color;\n"
                                      "in vec2 uv;\n"
                                      "out vec4 fragColor;\n"
                                      "void main()\n"
                                      "{\n"
                                      "    if (distance(uv, vec2(0.5, 0.5))>0.5)\n"
                                      "    { fragColor=vec4(0, 0, 0, 0); }\n"
                                      "    else\n"
                                      "    { fragColor=material_color; }\n"
                                      "}",  source: true);;
```

On aurait pu faire l'ellipse et le cercle à l'aide d'un maillage, mais c'est plus compliqué et plus long.

2.5 Interface

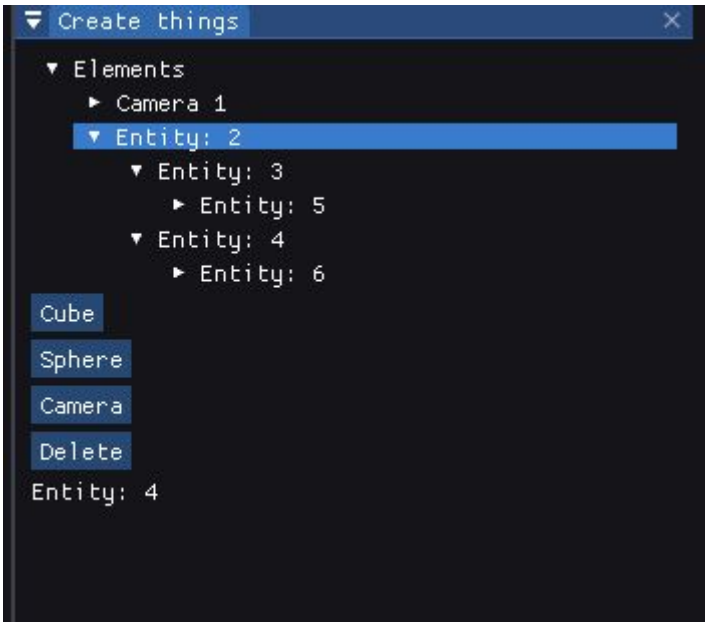
Pour l'interface on a utilisé ImGui. On peut au début de l'application sélectionner une scène parmis trois. Les deux premières permettent de créer et de modifier des objets dans une scène, soit 3D ou 2D. En 3D, la principal interaction est la modification de la position, la rotation et de la proportion des objets. En 2D, on peut créer des primitives et modifier les paramètres de leur création. On voit, entre autre, l'outil sélectionné.


WINDOW
EDITOR
Camera 1
Perspective
Transform
0.000 - + x 0.000 - + y 0.000 -
Active rotation
0.000 - + rot x 0.000 - + rot y 0.00
1.000 - + size x 1.000 - + size y 1.
Create things
Elements
Cube
Sphere
Camera



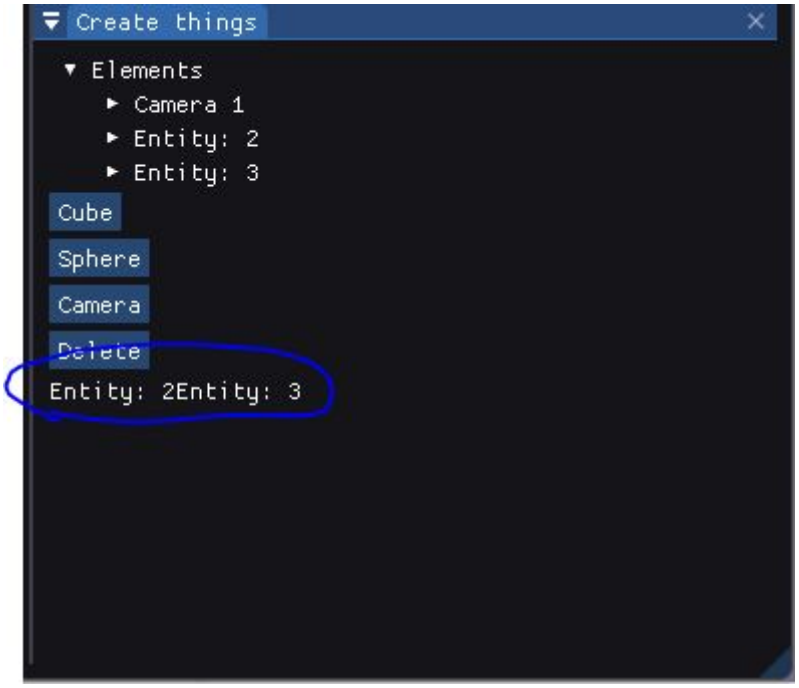
3.1 Graphe de scène

On peut ajouter des entités en appuyant sur « Cube », « Sphere » ou « Camera ». On peut supprimer des entités. On peut également sélectionner les différents éléments de la scène. Les éléments sélectionnés sont montrés sous le bouton « Delete ». Cependant, lorsque la composante x,y,z du parent n'est pas à 0 et que la rotation est activée, si on sélectionne un enfant il bouge indéfiniment.



3.2 Sélection multiple

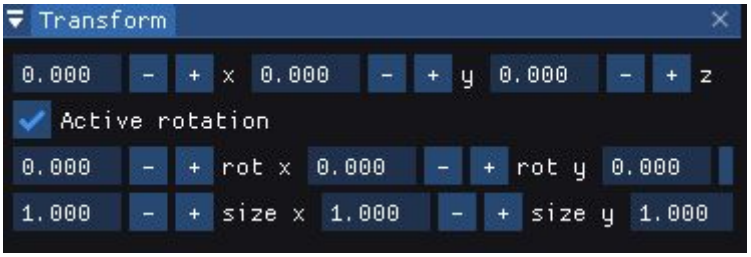
On peut sélectionner plusieurs éléments de la scène à l'aide de la touche « CTRL ». De plus, quand on modifie une composante de transformation, on modifie toutes les entités s'ils ont la même composante. Sinon, on modifie rien.



3.3 Transformations interactives

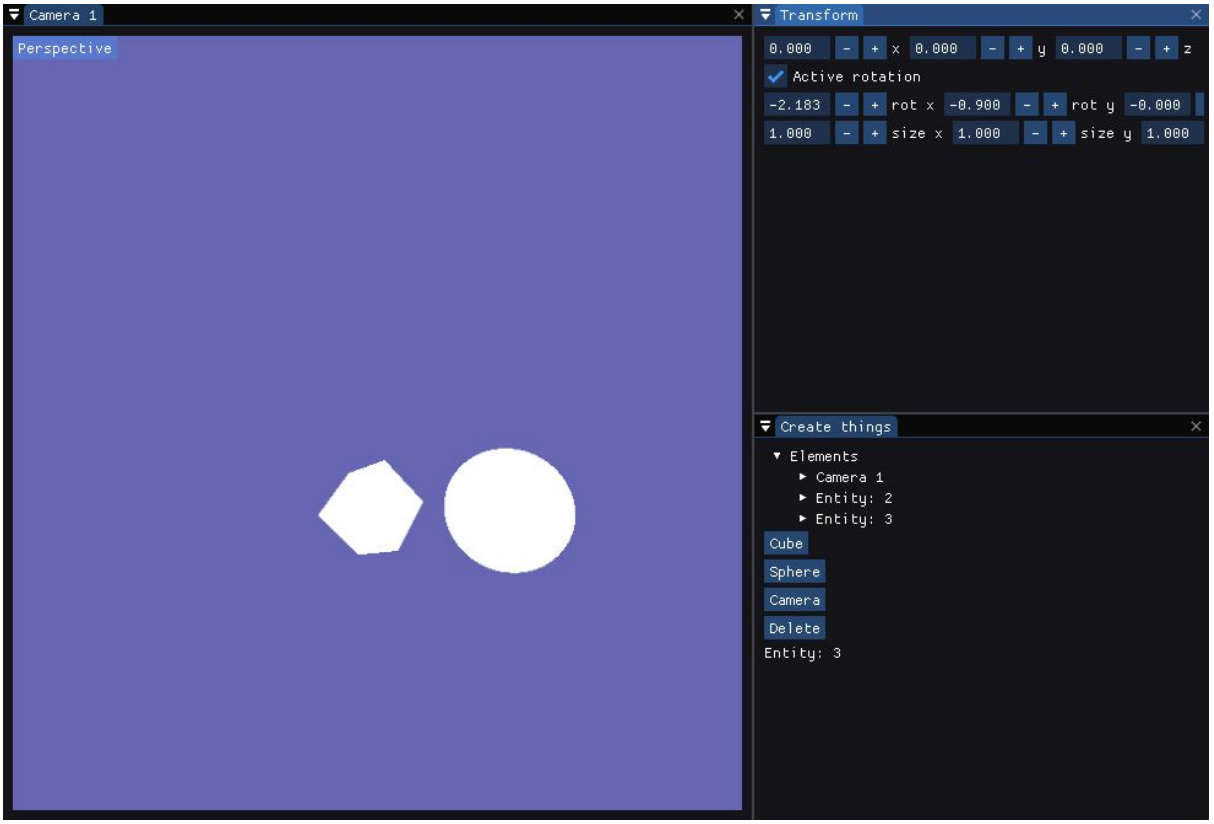
Il est possible de modifier les composantes de position, rotation et proportion des éléments sélectionnés. Cependant, lorsqu'on à un parent avec une composante en x,y,z non nulle et

qu’on sélectionne un enfant, l’enfant bouge indéfiniment. C’est causé par la rotation. Pour pouvoir tester le parent faut désactiver la rotation. Bref, la position, la proportion marche très bien, mais la rotation marche pas très bien. De plus, la rotation bloque à certains angles de rotation ce qui fait qu’on peut difficilement atteindre une rotation voulu.



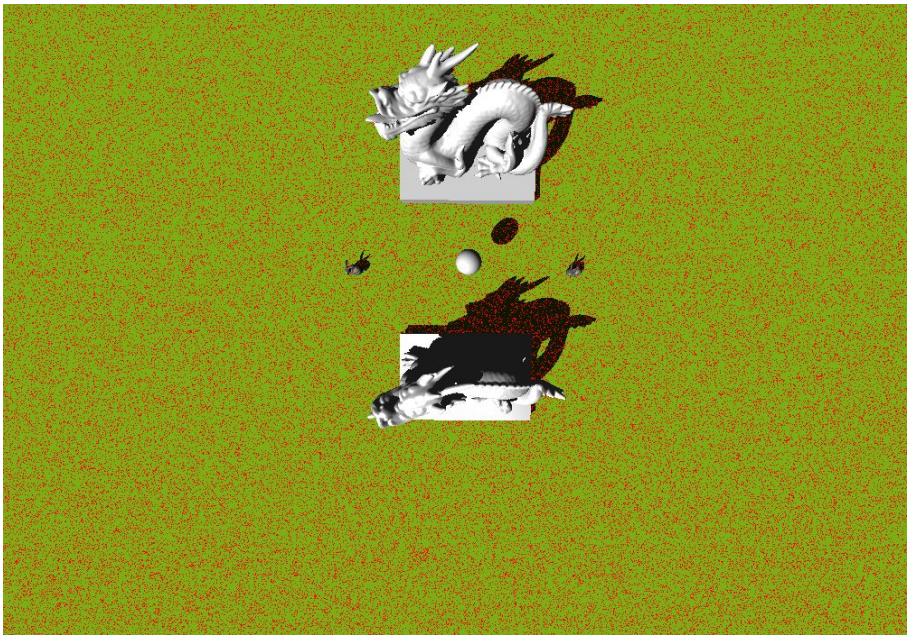
4.2 Primitives géométriques

Il est possible de générer un cube et une sphère. L’algorithme de la sphère a été pris sur le site https://www.songho.ca/opengl/gl_sphere.html.



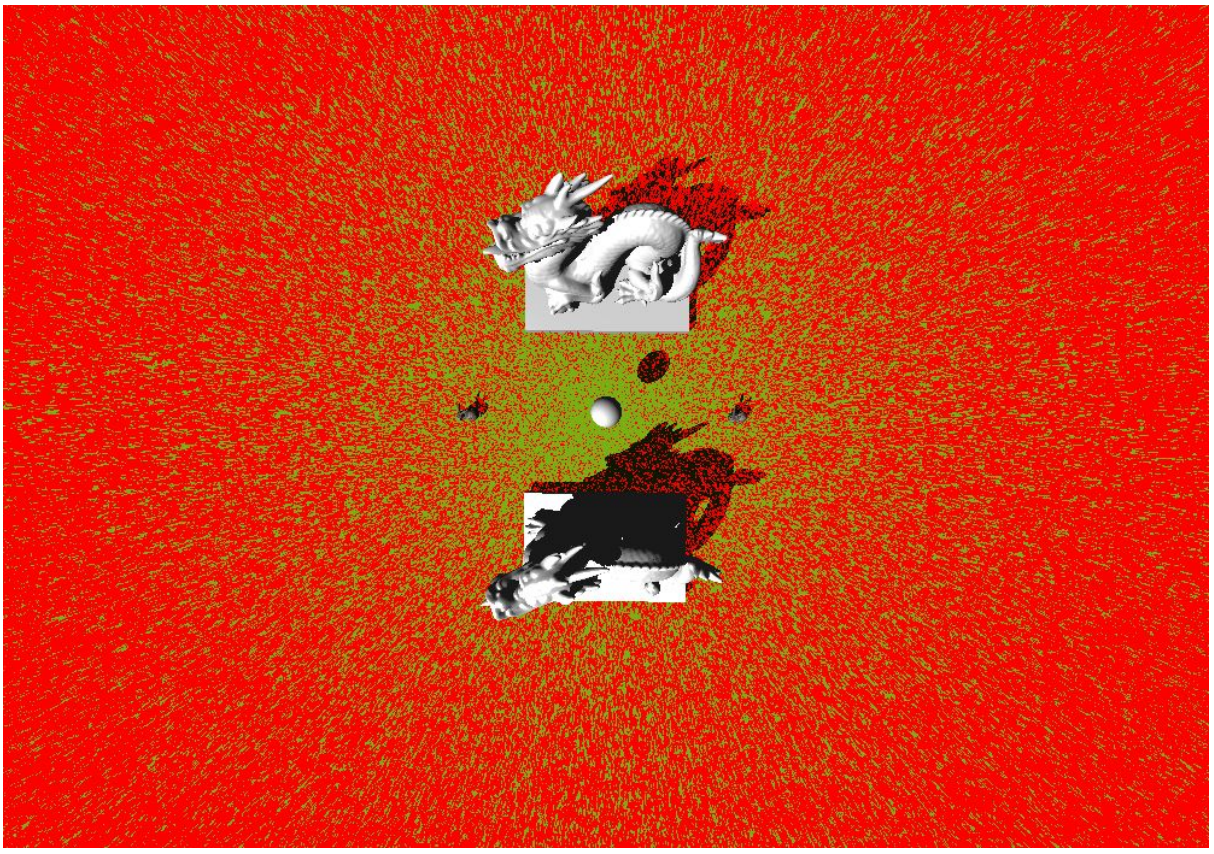
4.5 Shader de géométrie

Pour le Shader de géométrie on a décidé de créer un shader de gazon. Premièrement on choisit plusieurs points aléatoire dans une distance donné puis on les ajoute a un VertexArrayObject. Avec un shader normal ceci est le résultat:



Ensuite, on donne un vecteur de normal aléatoire a chaque point puis dans le shader de geometry on génère un triangle par point:

```
uniform float time;
uniform mat4 mvp;
in vec2 normal[];

float rand(vec2 co){
    return fract(sin(dot(co.xy ,vec2(12.9898,78.233))) * 43758.5453);
}
void main() {

    vec4 pos=gl_in[0].gl_Position;

    vec3 size=vec3(normal[0].y,rand(pos.zx),normal[0].x);

    gl_Position =mvp* (pos+vec4(size.x*0.1, 0,size.z*0.1, 0));
    EmitVertex();

    gl_Position = mvp*(pos+vec4(0, size.y, 0, 0));
    EmitVertex();

    gl_Position =mvp* (pos+vec4(-size.x*0.1, 0, -size.z*0.1, 0));
    EmitVertex();

    EndPrimitive();

}
```

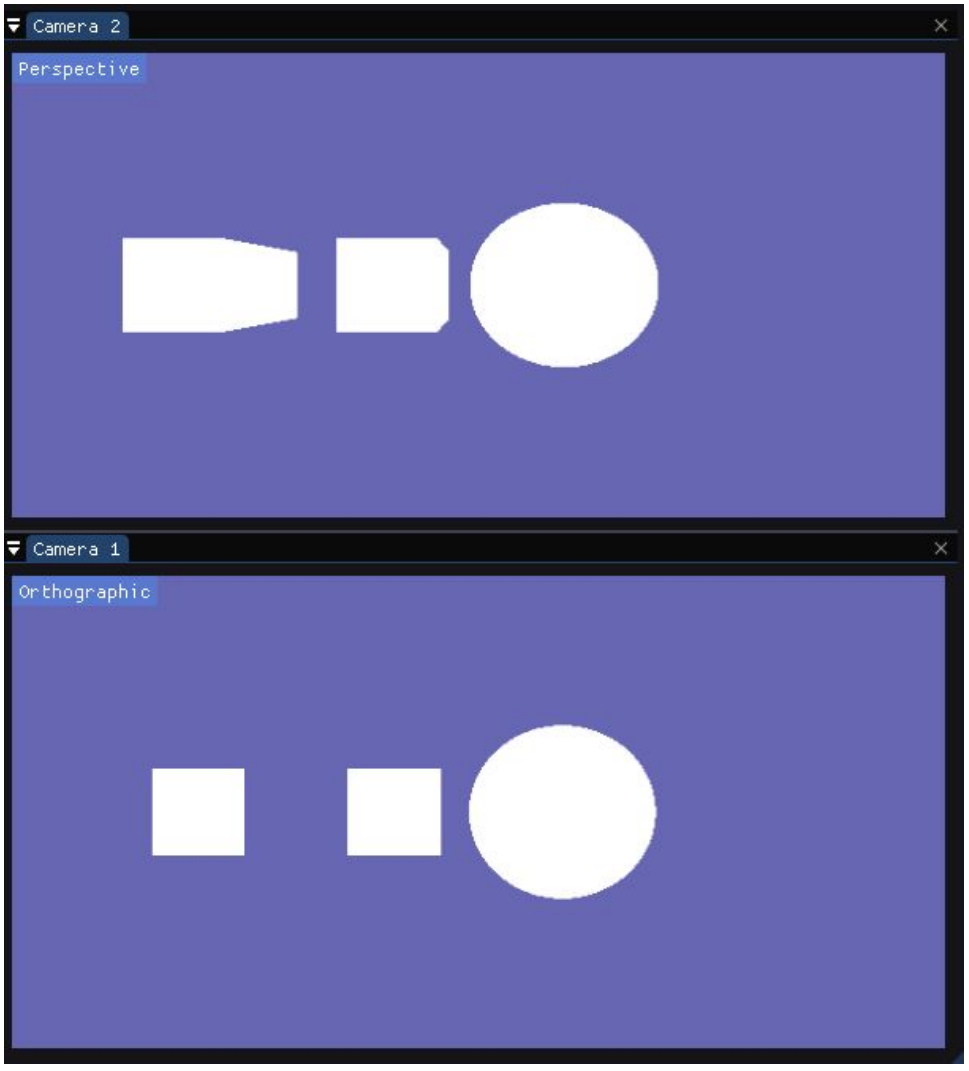


Il aurait été intéressant d'ajouter l'interaction de lumière pour chaque brin de gazon mais nous avons manqué de temps pour cet ajout qui aurait été de toute beauté.

5.2 Mode de projection

Pour le mode de projection, il nous a suffi de faire un bouton qui change la matrice de la caméra avec les fonction de la librairie glm ortho et perspective qui crée des matrice de projection selons quelques paramètres.

```
19  void Camera::setProjectionPerspective() {
20      this->projectionMode = PERSPECTIVE;
21      double w = (float) canvas->getPixelWidth();
22      double h = (float) canvas->getPixelHeight();
23      double aspect_ratio = w / h;
24      projection_matrix = glm::perspective<float>( fovy: glm::radians( degrees: fov), aspect_ratio,  zNear: 0.1f,  zFar: 200.0f);
25  }
26
27  void Camera::setProjectionOrtho(float width, float height) {
28      this->projectionMode = ORTHOGRAPHIC;
29      ortho_units.x=width;
30      ortho_units.y=height;
31      double w = (float) canvas->getPixelWidth();
32      double h = (float) canvas->getPixelHeight();
33      double aspect_ratio = w / h;
34      projection_matrix = glm::ortho<float>( left: -width / 2,  right: width / 2,  bottom: -(height / 2) / aspect_ratio,
35                                             top: (height / 2) / aspect_ratio,  zNear: -100,  zFar: 100);
36  }
37
```



5.3 Agencement

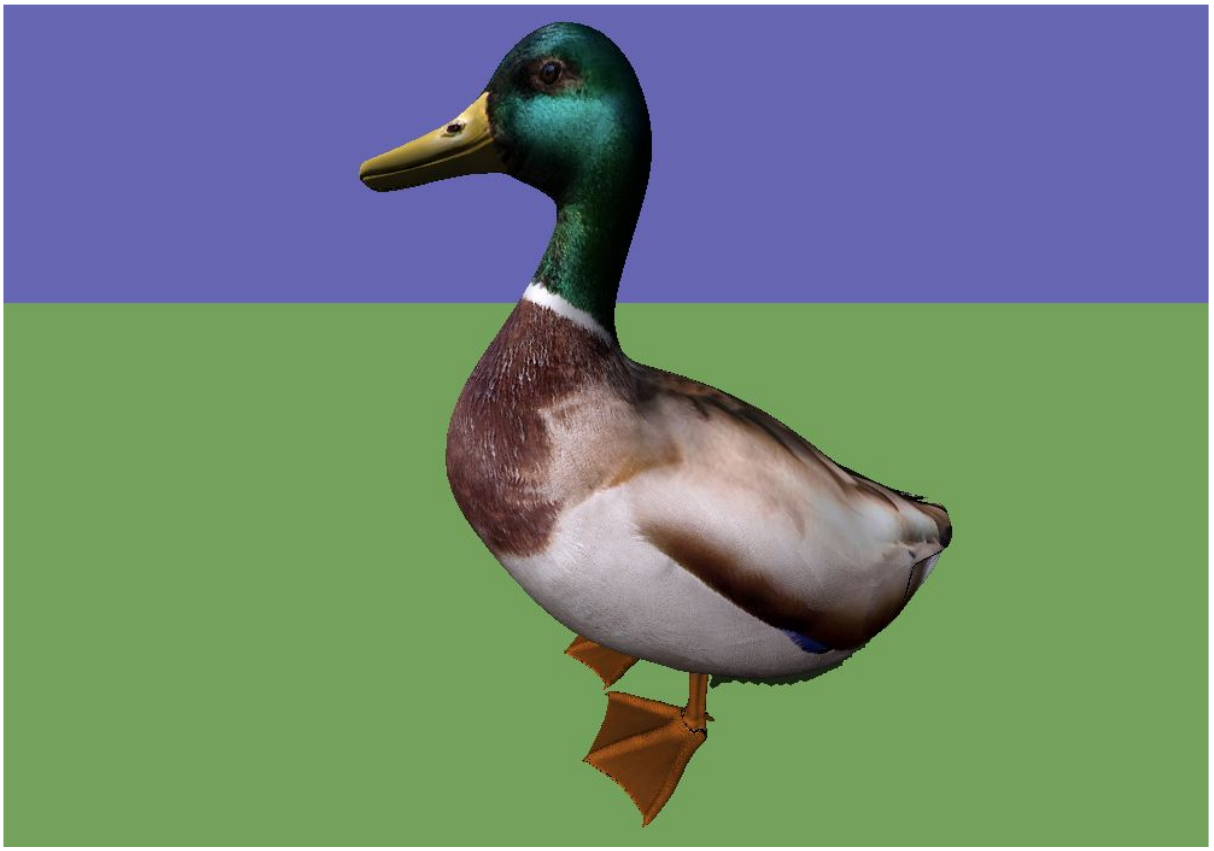
Pour créer plusieurs points de vue d'une scène nous avons créé un bouton qui donne l'option de créer plusieurs caméras.Ces caméras on tous un frame buffer associé.À chaque mise à jour, ont fait un rendu de la scène dans la texture du frame buffer pour chaque caméra.Ensuite, ce qui nous reste à faire est seulement d'afficher ces textures à chaques mise à jour dans l'interface utilisateur.

5.5 Portail

Pour le portail, nous avons décidé de faire un mirroir dans la scène.Il suffit de créer un frame buffer supplémentaire et une autre caméra. placer la Caméra en arrière d'un quad puis afficher la texture du frame buffer qui change à chaque mise à jour.Nous avons malheureusement pas été capable de faire un effet réaliste ou l'image de la caméra change en fonction du l'angle de la caméra.

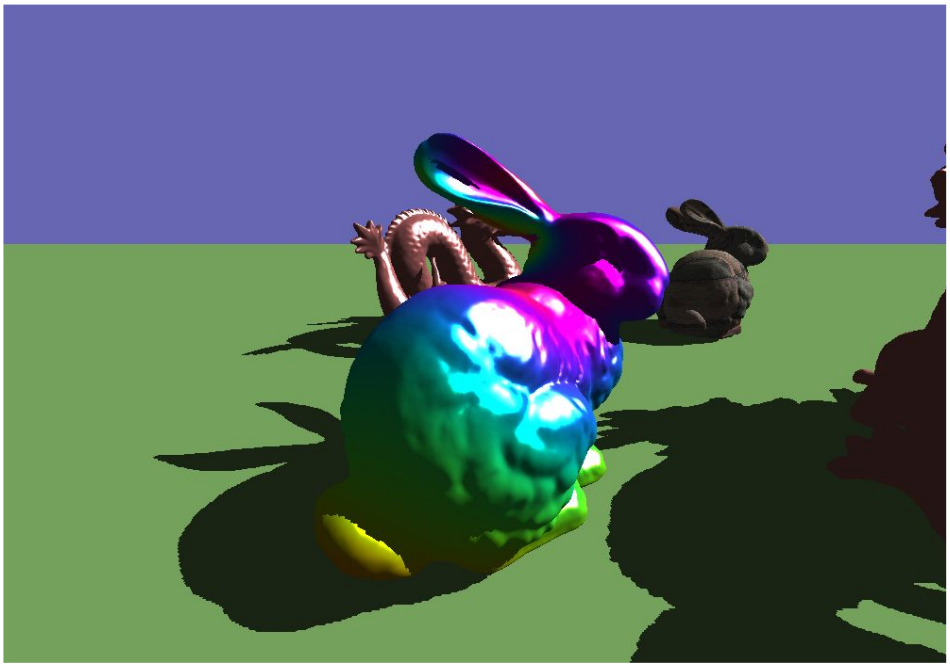
6.1 Coordonnées de texture

Pour ce point , nous avons pris ce modèle:
https://free3d.com/3d-model/bird-v1--282209.html.
Ensuite nous avons pris les normal et les coordonnées de textures inscrit dans le format obj pour les mettre sur un modèle dans l'application:

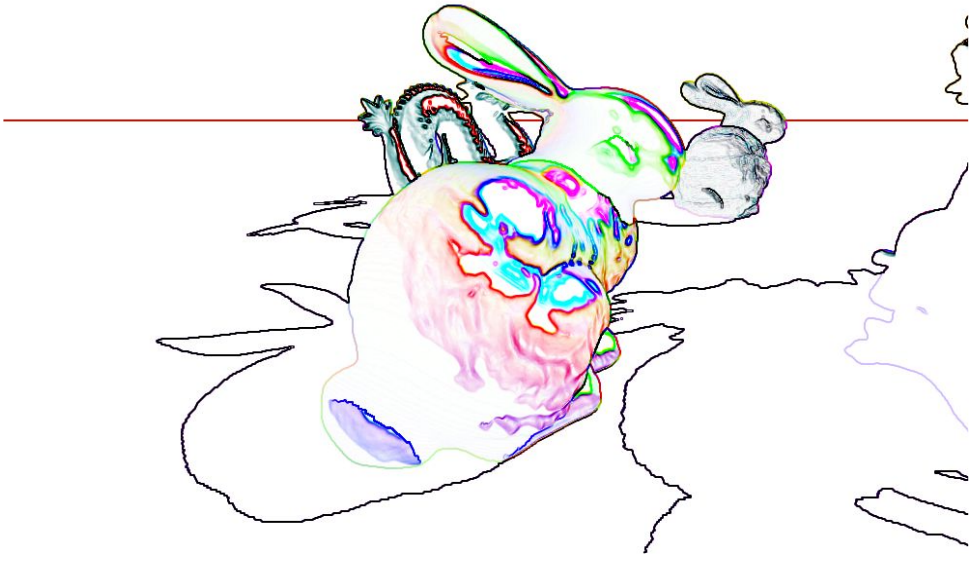


6.2 Filtrage

Pour le filtrage nous avons créé 3 effet de filtrage.Dans l'application il est possible d'appuyer sur la touche de 1 à 4 pour changer le monde de rendu a un de ces filtres
1:Normale

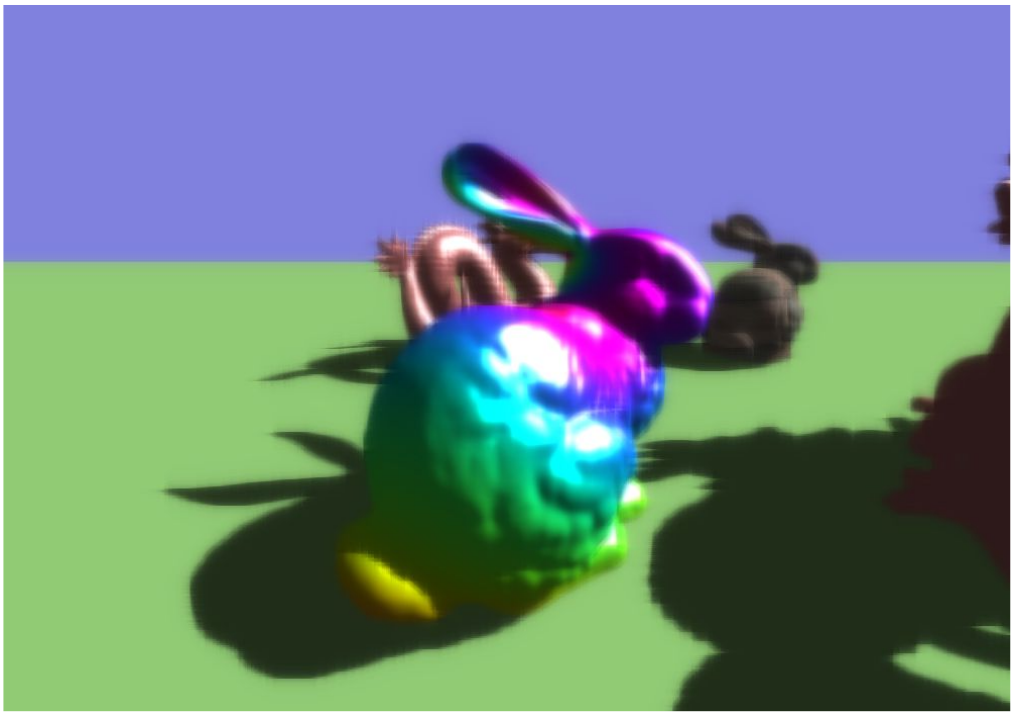


2:Détection des bordures:



3:Inverse des couleurs:

4:Floue

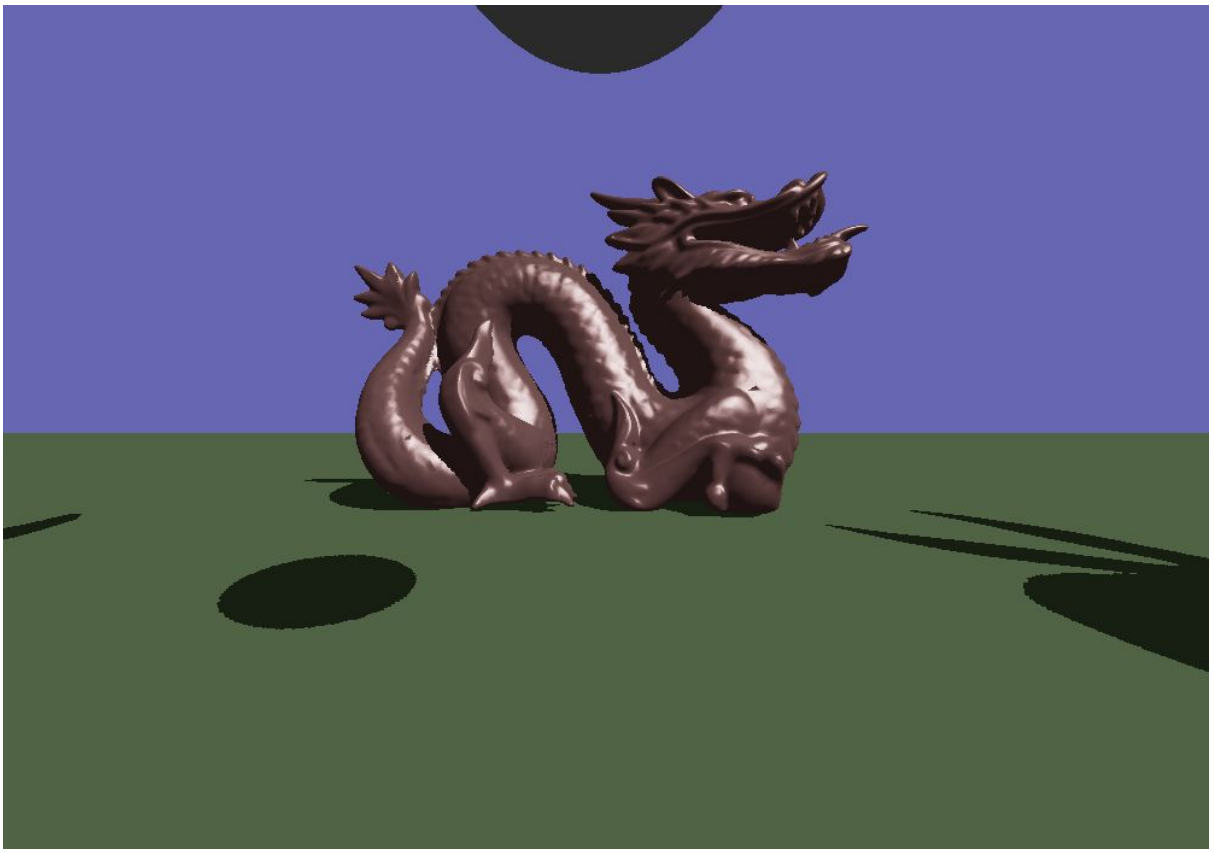


6.3 Mappage Tonal

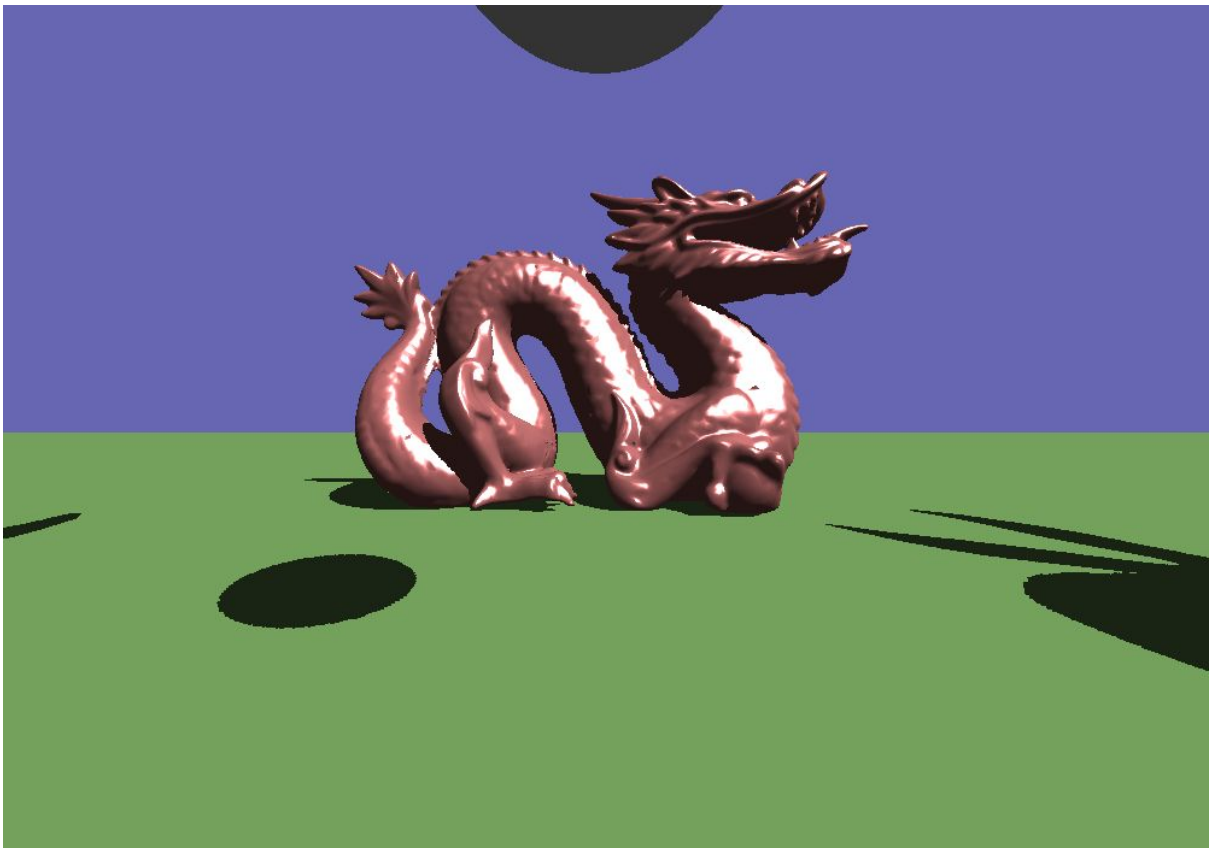
On a ajouté l'algorithme de reinhard directement dans le shader qu'on utilise pour l'illumination.

```
if(tonal_mapping) {
    fragColor = vec4(hdrColor.rgb / (hdrColor.rgb + vec3(1.0)), 1.0);
}
else {
    fragColor = hdrColor;
}
```

Avec mappage tonal:

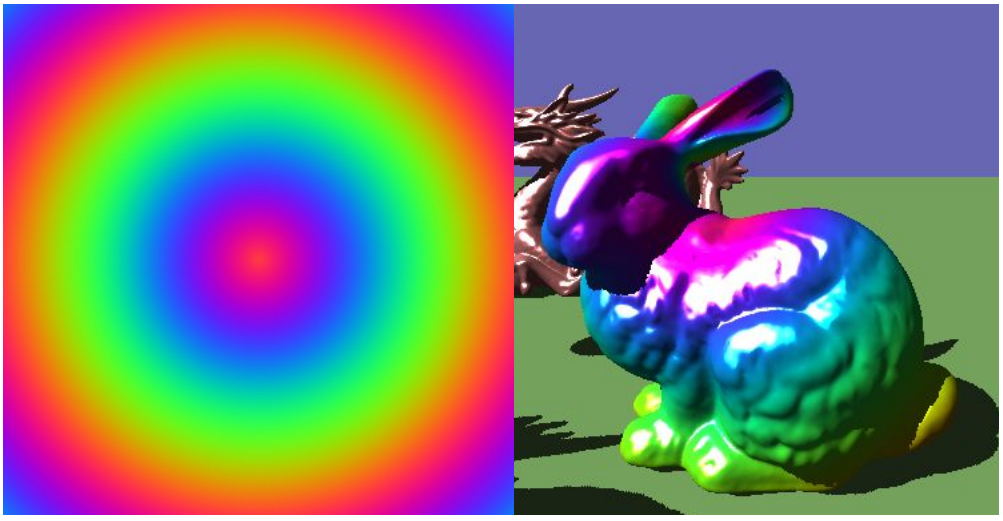


Sans mappage tonal:



6.5 Texture procédurale

Pour la texture procédurale, nous avons créé une texture qui fait un gradient de couleur à puis nous l'avons appliquer au modèle du lapin de stanford:

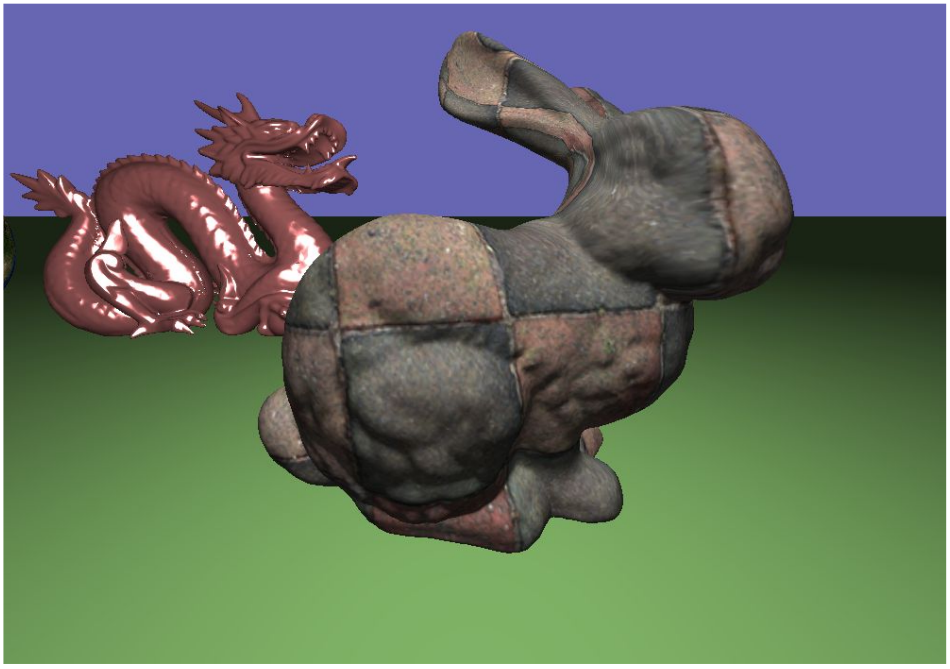


```
void createRainbowTexture(Texture &texture) {
    std::vector<unsigned char> data;
    data.resize( new_size: 1024 * 1024 * 3);
    const float PI = 3.1416;

    for (int x = 0; x < 1024; ++x) {
        for (int y = 0; y < 1024; ++y) {
            vec2 uv( _x: (float)x/1024.0f, _y: (float)y/1024.0f);
            uv-=0.5;
            int index = ((1024 * x) + y) * 3;

            float ratio = distance(uv, p1: vec2( _x: 0, _y: 0));
            ratio*=2;
            double r, g, b;


            r =(cos( _X: ratio * PI * 2.) + 1)/2;
            g =  (cos( _X: (ratio + 0.33333) * PI * 2.) + 1)/2 ;
            b = (cos( _X: (ratio + 0.66666) * PI * 2.) + 1)/2;

            data[index] = static_cast<unsigned char>(r * 255);
            data[index + 1] =static_cast<unsigned char>(g * 255);
            data[index + 2] =static_cast<unsigned char> (b * 255);
        }
    }
    texture.setTexturePixelData(data.data(),  width: 1024,  height: 1024,  smoothed: true,  type: Texture::Type::RGB);
}
```

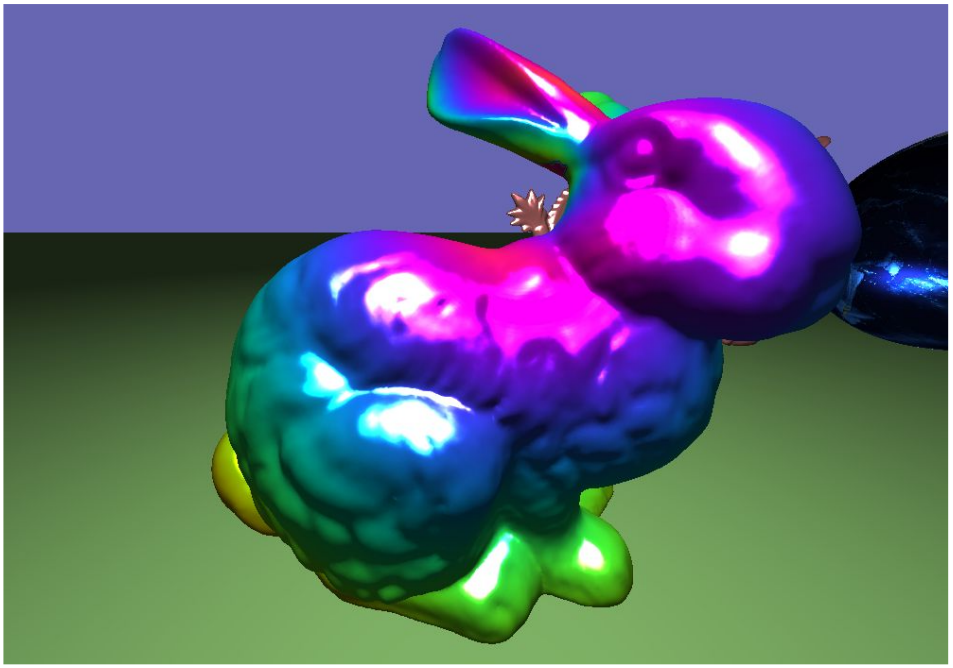
7.2 Matériaux

Pour nos matériaux, ils ont une couleur,une texture,une normal map,un paramètre de “shiness” et un de “dampness” qui contrôle la réflection de la lumière.il est possible davoir un infinité de matériaux avec différents paramètres.voici quelques exemples:

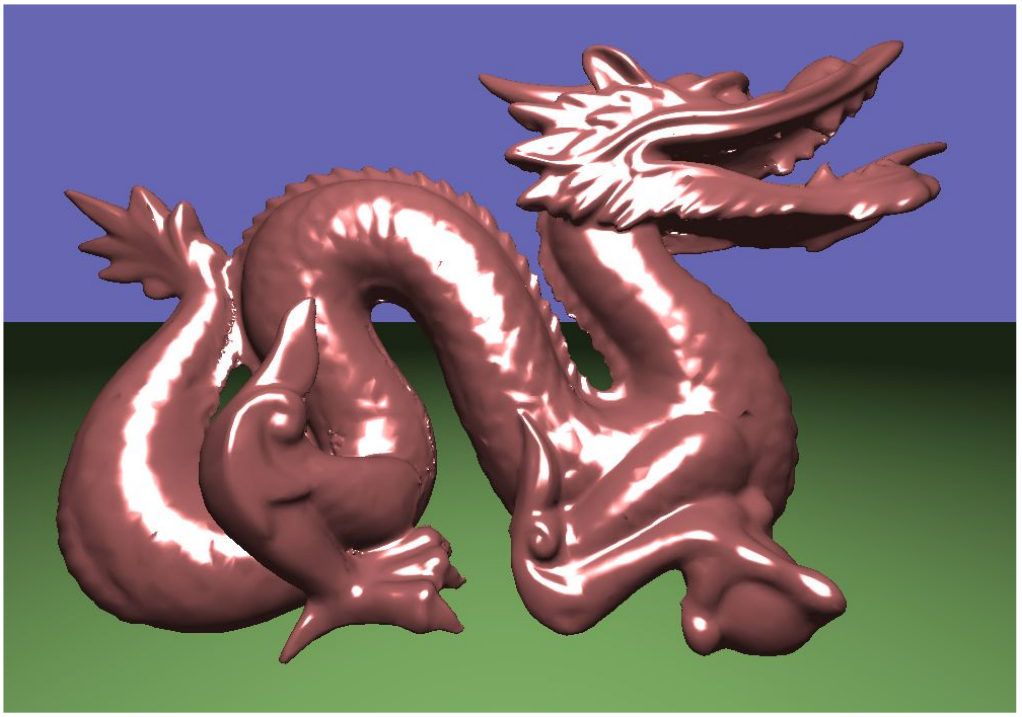
Lapin avec un matériaux 0 shiness 0 dampness et une texture de tuiles.



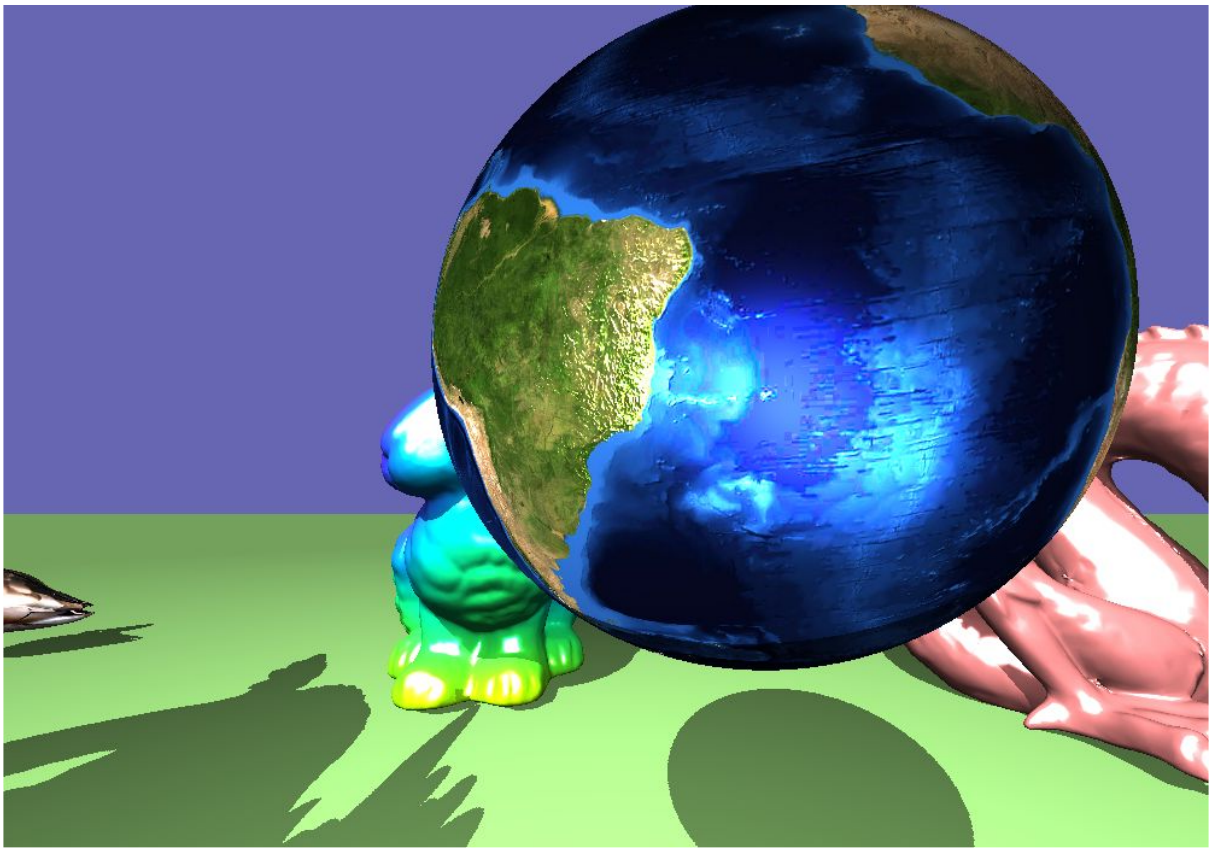
Le même Lapis mais avec une texture procédurale et 16 dampness et 64 shiness



Un dragon sans texture avec les paramètre de matériaux:couleur(r=0.7,g=0.4 et b=0.4). shiness=20 et dampness =16

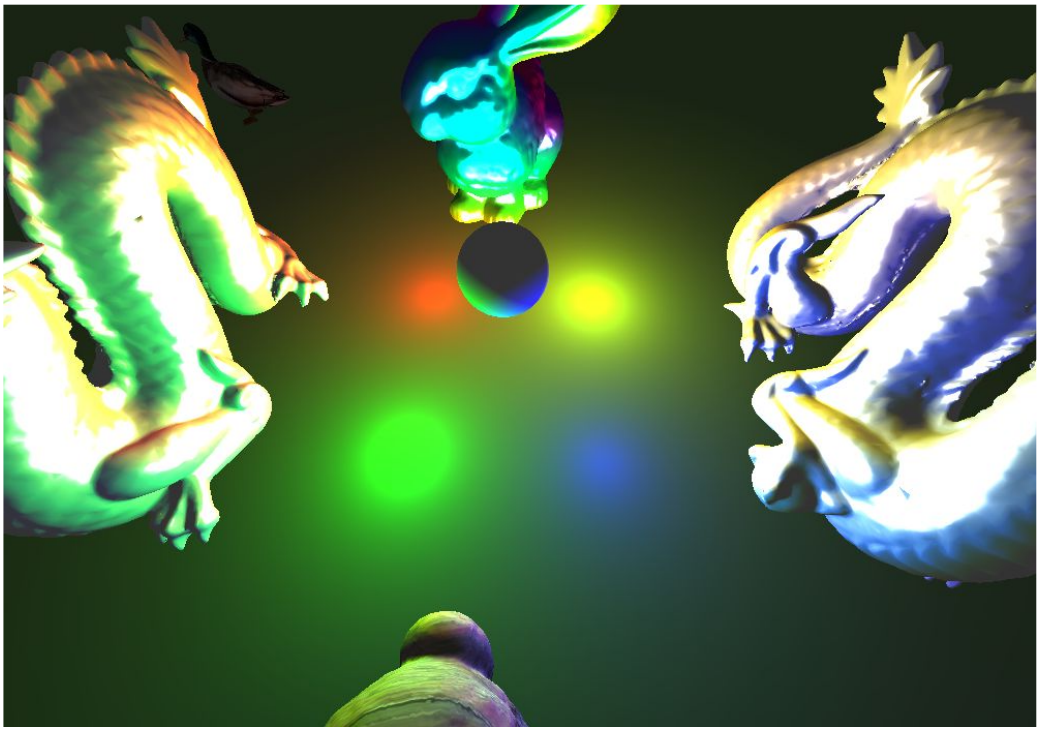


Une sphere avec um Matériax avec shiness 20 et dampness 16 une texture et une normal map:



7.4 Lumières multiples

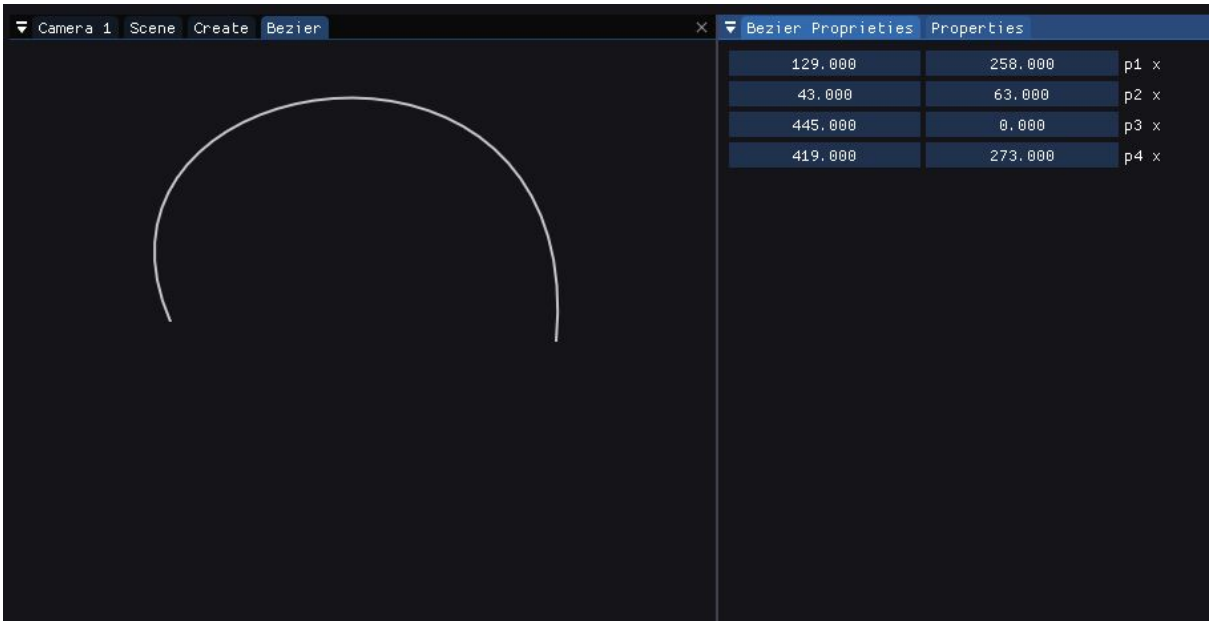
Pour les lumières multiples, nous supportons un total de 8 lumières en ponctuelle et 1 lumière directionnelle.Pour ce faire, nous envoyons un tableau des données de lumières au shader puis dans le shader on calcule toutes les lumières puis on multiplie le tout:



```
//point
uniform int point_light_count;
uniform float point_light_radius[8];
uniform vec3 point_light_colors[8];
uniform vec3 point_light_positions[8];
uniform float shine_factor;
uniform float damp_factor;
uniform vec3 view_pos;
//directionnal
uniform int directional_light_count;
uniform vec3 directional_light_color;
uniform sampler2D directional_light_shadowMap;
uniform vec3 directional_light_direction;
```

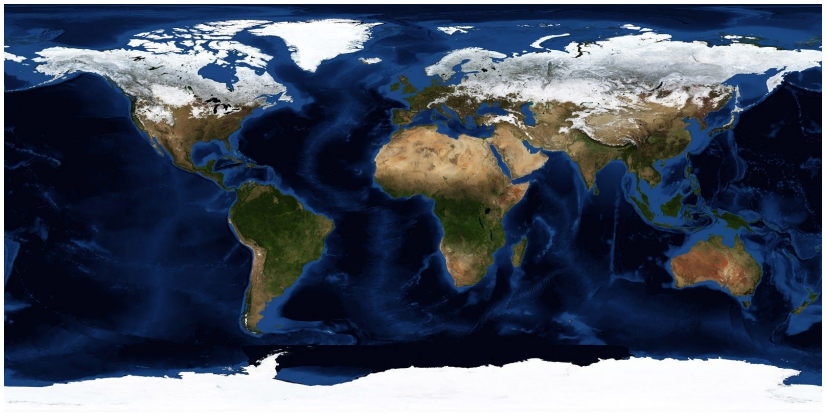
9.1 Courbe paramétrique

On a ajouté une courbe de bézier implémenter à l'aide de ImGui.

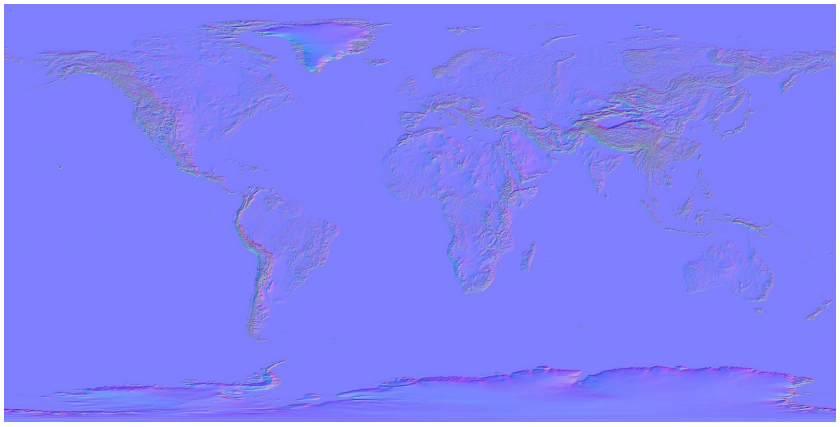


9.5 Effet de relief

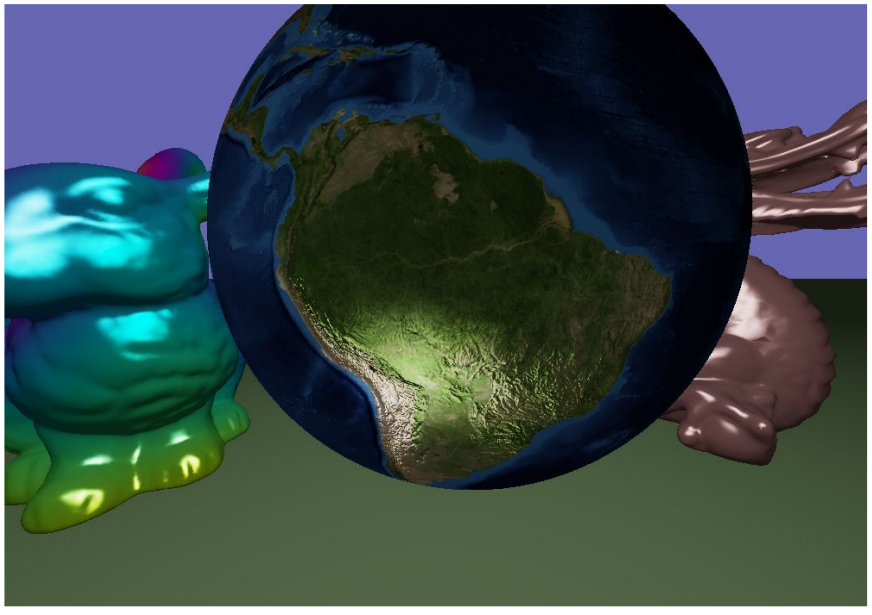
Pour l'effet de relief nous avons utilisé une sphère généré avec un algorithme puis avons appliqué la texture suivante:



Ensuite , nous avons multiplié la normal par la texture suivante:



Ce qui nous a donné ce résultat:



Malgré le normal mapping fonctionnel, nous avons éprouver des difficulté avec la réflection spéculaire de la sphère qui n'est pas au bon endroit et nous avons malheureusement pas été en mesure de l'améliorer.

10.1 HDR

Tous les calcules d'illumination sont faite à l'aide de nombre à virgule flottante normalisée. Nos couleurs de matériaux sont normalisés entre 0 et 1 et lorsqu'on charge une texture en OpenGl les couleurs de la texture sont automatiquement normalisées.
Charger la texture:

Lire la texture:

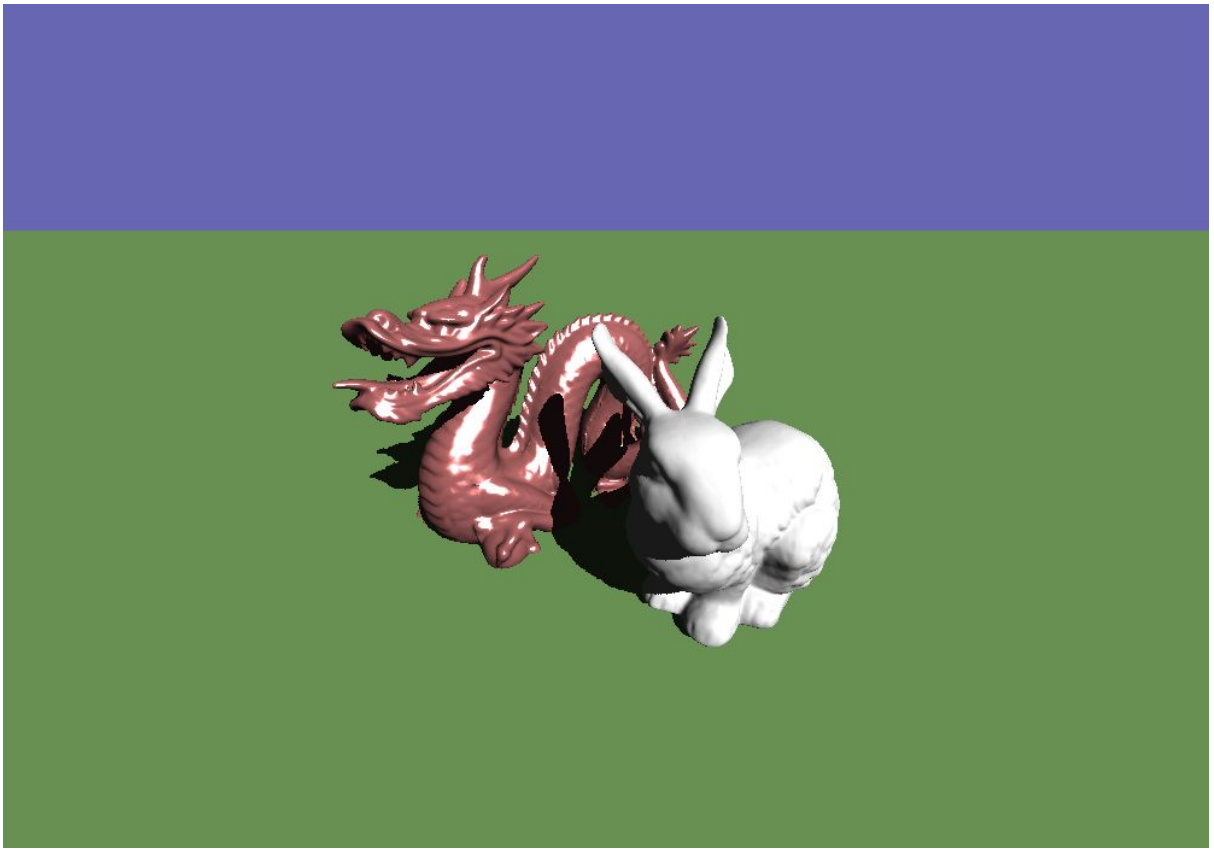
```
vec4 textureColor=texture(texture_0, uv);
```

Représentation des couleurs:

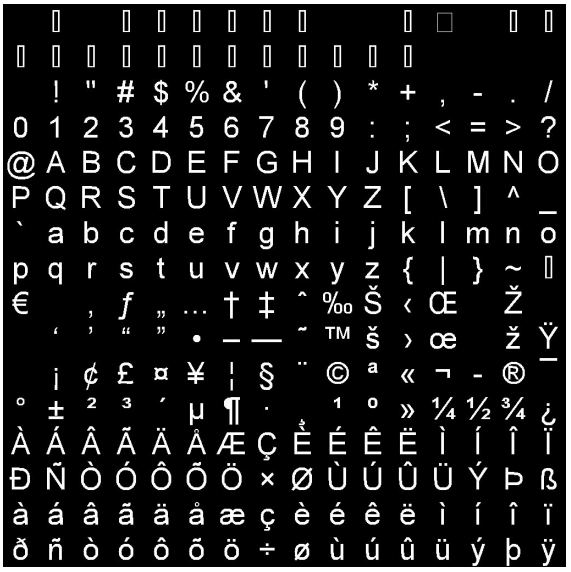
```
Color(float r=0,float g=0,float b=0, float a=1);
```

Ressources

Pour ce qui est des ressources 3D, nous avons utilisé les modèles de l'université de stanford :http://graphics.stanford.edu/data/3Dscanrep/. En autres , le lapin et le dragon ce sont des modèles assez massif car ils contiennent plus de 300 000 sommets chacun ce qui nous a permis de tester notre algorithme d'importation de models 3D.



Nous avons créer des police de caractère pour nos texte avec le logiciel Bitmap Font Builder qui créait des images comme celle ci dessous.



Nous avons utilisées référence pour travailler avec openGL:
https://learnopengl.com/
https://www.opengl-tutorial.org/
http://docs.gl/
https://www.songho.ca/opengl/gl_sphere.html.

7. Présentation

L’équipe est constituée de deux membres. Simon Robidas et Gabriel Bouchard. Tous les deux on fait une technique en programmation jeux vidéo au Cégep Sainte-Foy. Avant d’avoir débuté le cours Gabriel avait commencé déjà à explorer OpenGL dans son temps libre.

On a donc décidé de poursuivre le travail déjà commencé. Gabriel avait avant tout développé la base de l’engin graphique 3D. Il a fait notamment l’importation de modèle et la visualisation de ceux-ci à l’aide de caméras. Il avait fait également l'utilisation de texture et la création d’un modèle de lumière.

Partie 1.

Pour aider, Simon, à travailler davantage sur la partie 2D de l’engin graphique et sur l’implémentation des critères fonctionnels à l’aide de l’engin 3D de Gabriel. Les critères qu’il a fait sont:

- 1.1 Importation d’images
- 1.3 Échantillonnage d’images
- 1.4 Espace de couleur
- 2.2 Outils de dessin
- 2.3 Primitives vectorielles
- 2.5 Interface
- 3.1 Graphe de scène
- 3.2 Sélection multiple
- 3.3 Transformations interactives
- 4.2 Primitives géométriques

Ce qui a lasser à Gabriel à implémenter ou à perfectionner les critères:

- 2.1 Curseur dynamique
- 4.5 Shader de géométrie
- 5.2 Mode de projection
- 5.3 Agencement
- 5.5 Portail

Même si les tâches semblent n’être pas vraiment bien divisé ce n’est pas vraiment le cas puisque la majorité du travail était d’implémenter les fonctionnalités (dessiner des ellipses, créer les transformations et les caméras) en arrière ce que Gabriel a fait plus que Simon. Simon c’est surtout assuré de rendre les fonctionnalités disponibles à l'utilisateur et non seulement disponible dans le code.

Partie 2

La répartition des tâches va comme suit:
Gabriel:

- Coordonnées de texture
- Filtrage
- Texture procédurale
- Matériaux
- Lumière multiple
- Effet de relief

Simon:

- Mappage tonal
- Courbe Paramétrique
- HDR