Programmation Graphique 3D

TP noté - M1 I3D / CMI IIRVIJ

Encadrants : Jean-Michel Dischler / Pascal Guehl
Année : 2019-2020

durée : 2 heures

<u>CONTEXTE</u> : Applications concrètes dans l'Industrie

Vous postulez pour un job dans le domaine du cinéma en tant que "shader writer", "technical director", "graphics engineer" ou "look-dev artist". On vous propose de tester vos connaissances, notamment vos savoir-faire en programmation 3D sur la thématique suivante :

C'est la période de Noël, donc des cadeaux : vous allez devoir réaliser un programme webGL 3D temps-réel, qui consistera en la génération d'un bijou dont l'aspect visuel pourra être modifié en temps-réel.

Vous travaillerez avec des infographistes qui ont modélisé des objets 3D pour vous mais ne savent pas exactement comment en faire le rendu. Vous devrez les aider à écrire le code et les shaders en suivant leurs indications pour obtenir les différentes apparences attendues et les divers paramètres qu'ils souhaitent avoir dans une interface graphique afin de pouvoir contrôler les effets.

3 exercices : 10 points, 5 points, 5 points

- 1er : matrices de transformation
- 2er : rendu non-photoréaliste (NPR)
- 3ème : textures procédurales

NOTE : n'utilisez qu'un seul programme webGL pour les exercices. Vous utiliserez un entier pour switcher le type de rendu/apparence.

IMPORTANT : votre UX designer vous fournit une interface graphique avec tous les paramètres dont vous aurez besoin pour vos effets. Vous aurez à récupérer la valeur des widgets pour initialiser vos variables.
=> vous avez à disposition un "matériau" et deux "lumières".

EXERCICE #1 - Transformations dans le pipeline graphique

[10 points]

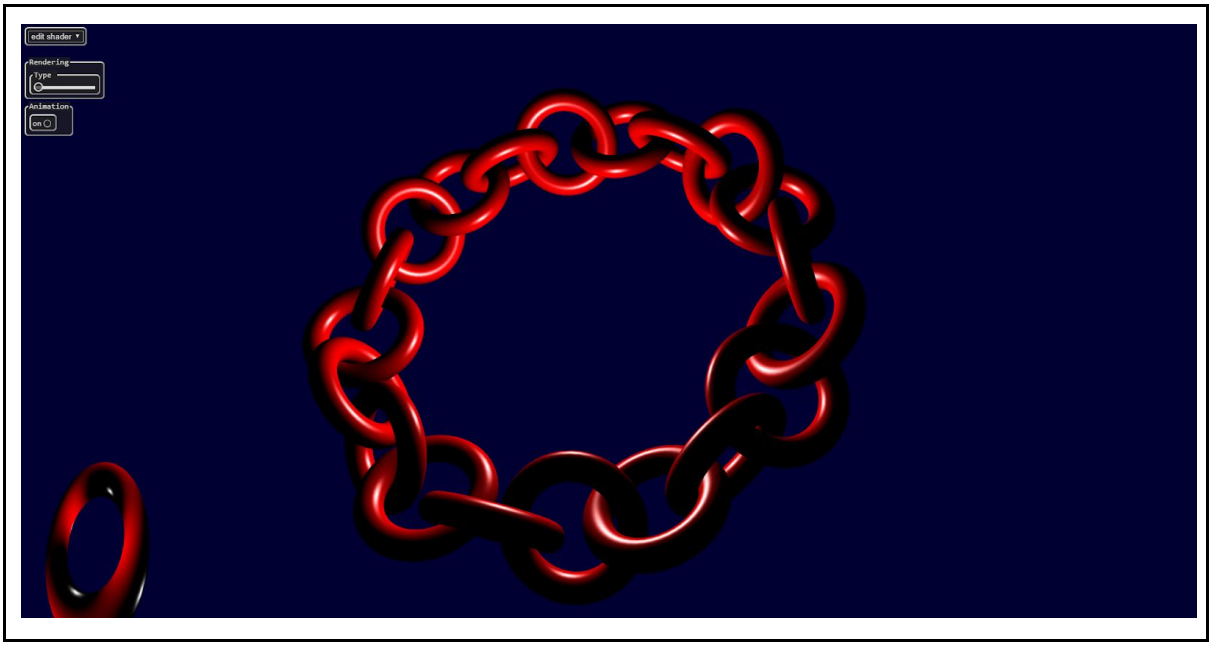
Gollum/Smergol et la communauté de l'anneau : "Mon précieux !"

Gollum va offrir un bijou à Smergol… … … ou l'inverse !? O_o

On veut du shiny, il faut que ça brille !

BIJOU - exemple de rendu attendu :

On va utiliser plusieurs anneaux !



Procédez par étape :

[a] créer puis dessiner un tore avec l'API de Sylvain.

A l'initialisation, utiliser l'API simplifiée :

```
// Create geometry
```

```
let mesh = Mesh.Tore( 50, 50, 0.5, 2 );
// Create GL objects : VAO + VBO (“index” des buffers de “positions”, “normales”, et “coordonnées de texture”)
mesh_rend = mesh.renderer( 0/*position*/, 1/*normal*/, 2/*texCoord*/ );

// placer la camera pour bien voir l'objet
wgl.scene_camera.show_scene( mesh.BB );
ewgl.scene_camera.set_scene_radius( 20 );
ewgl.continuous_update = true;
```

note : utiliser ces 2 deux rayons “0.5” (petit cercle) et “2.0” (grand cercle) pour les tores

Au rendu, utiliser l’API simplifiée :

```
// Draw command (“bind” automatique du VAO)
mesh_rend.draw( gl.TRIANGLES );
```

NOTE : utiliser une couleur en dur dans votre **fragment shader** sans lighting/shading particulier.

[b] dessiner le tore en bas à droite de votre écran avec un viewport particulier :
Rappel : vous avez accès aux variables “gl.canvas.width” et “gl.canvas.height”

```
gl.Viewport( … )
```

[c] dessiner un ensemble de tores sur un plan 2D (un peu comme si les tores étaient les heures d’une montre, voir image ci-dessous **[c]**)
Paramètre fixes :

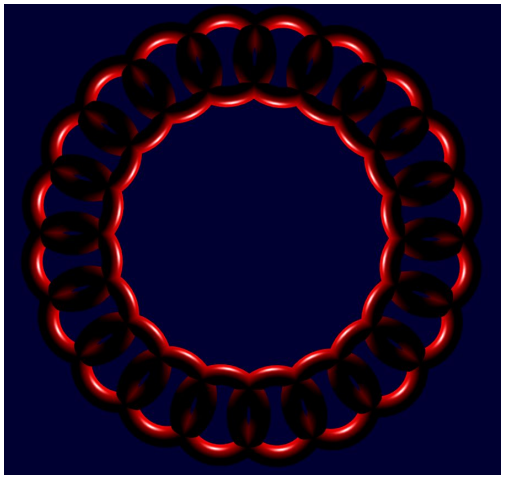
- **il y aura 20 tores**
- **le rayon du bijou est de 9 unités (on compte par rapport aux centres des tores)**
- **les tores sont répartis uniformément dans l’espace**

Note : Pour positionner les tores, utiliser l’API de Sylvain : ***Matrix.mult(…)*** => on peut concaténer plusieurs matrices, séparer par des virgules.
NOTE : si vous avez dessiné un tore dans un autre viewport, veillez à réinitialiser votre viewport global au début de votre rendu.

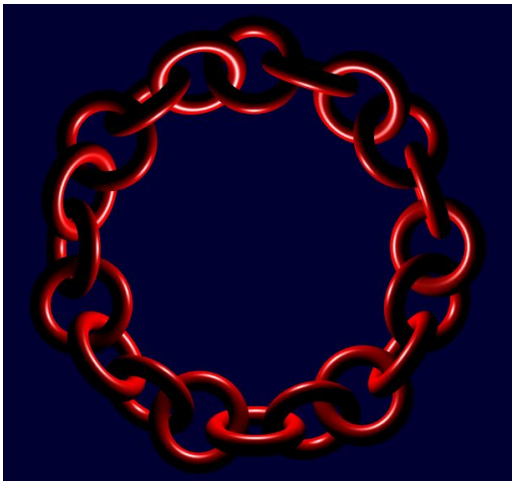
[d] Faites tourner le bijou dans son plan principal
Paramètres utilisateurs :

- booléen : animate or not

[e] ajouter une rotation finale pour tourner les tores sur eux-mêmes (voir image ci-dessous **[e]**). L’angle sera de 117 degrés entre chaque anneau..



[c] bijou à plat : plan 2d



[e] rotation finale des anneaux

[e] lighting/shading

1) ajouter les paramètres nécessaires pour faire un rendu pour que Gollum soit satisfait de son “précieux” avec un modèle de Phong classique (diffus + spéculaire).

NOTE : positionner la lumière dans l’**espace caméra**, sur l’oeil

[f] ajouter la possibilité de modifier l’espace où est défini la position de la lumière. Pour cela, ajouter un booléen, indiquant si l’on travaille en “**world space**” ou en “**view space**” pour la position de la lumière. On souhaite notamment pouvoir placer la lumière au centre de la scène (le centre du grand cercle du bijou).
NOTE : la modification doit se faire sur CPU, pas dans le shader.

[g] Ajouter une variable globale pour pouvoir modifier le type de rendu : “phong shading” ou rendu physiquement réaliste “PBR”. Vous devrez utiliser cette variable dans vos shaders pour switcher entre ces différents mode de rendu.
NOTE : je vous donne les fonctions pour réaliser ce rendu à base de microfacettes. Il prend en entrée un indice de lumières, la position et la normale du vertex courant dans l’espace caméra (view).

FIN de l’exercice #1

EXERCICE #2 - Rendu NPR: Non-Photorealistic Rendering

[5 points]

Concept art / direction artistique : The Legend of Zelda

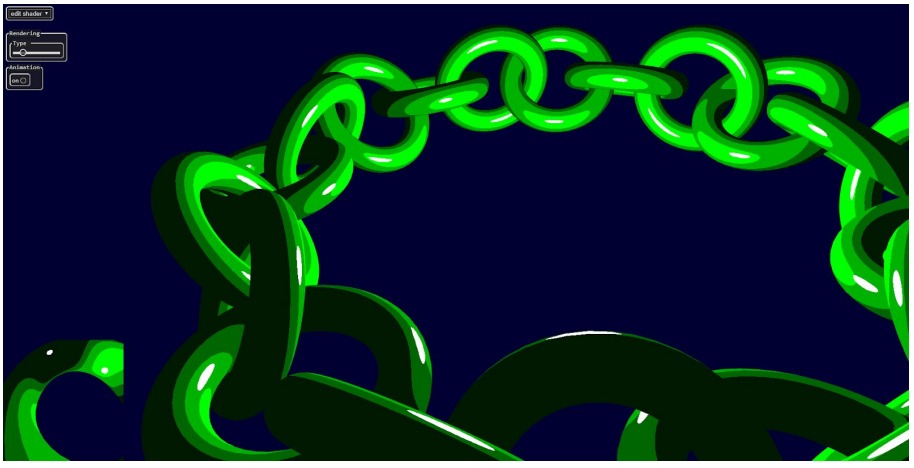
Vous êtes dans une équipe de développeurs de jeux vidéo dont le directeur artistique souhaite s'inspirer du game art/design du jeu vidéo "The Legend of Zelda - The Wind Waker". Les infographistes de votre équipe devront concevoir un bijou pour des personnages comme Zelda et Link avec la technique du "toon shading" (voir ci-dessous) avec votre aide.

Toon/Cel Shading

Le toon shading ou cel shading est une technique de rendu non-photoréaliste qui reproduit l'aspect des dessins animés sur celluloides (feuilles d'animation). Le nombre de couleurs est alors restreint à quelques valeurs : les variations d'intensités continues sont remplacées par une fonction créant des plateaux constants (elle est "quantifiée"), sous forme de bandes (voir les images ci-dessous). On dit que les données sont "quantifiées".

 "cel shading" (gauche) vs "rendu PBR" (droite)	 Le toon shading donne un "aspect" de dessin animé.

BIJOU - exemple de rendu attendu :



TODO
Reprenez l'exercice précédent et continuez sur le même code avec le même shader.
***NOTE** : si vous n'avez pas réussi à faire l'exercice précédent ou si vous commencez par celui-ci, faites l'exercice sur un tore unique et créer d'abord un lighting/shading de Phong (diffus + spéculaire). Pour créer un tore, regarder la partie **[a]** de l'exercice 1.*

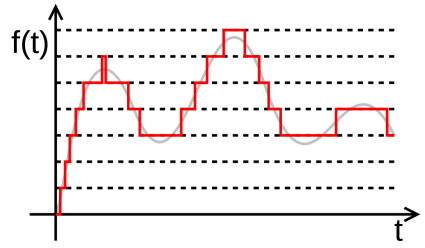
NOTE : GLSL fournit des fonctions comme qui pourraient vous être utile dans cet exercice : ***round***(), ***floor***(), ***fract***(), ***step***(), ***smoothstep***()...

[a] Si vous ne l'avez pas fait avant, ajouter une variable globale pour pouvoir modifier le type de rendu : "phong shading", "PBR" ou "toon shading". Vous devrez utiliser cette variable dans vos shaders pour switcher entre ces différents mode de rendu.

Paramètres utilisateurs :
- type de rendu (phong, npr)
- nombre de bandes du toon shading

NOTE : le choix du type de rendu doit intervenir dans votre shader.

[b] Créer/modifier la composante diffuse du modèle de Phong
Il faut "quantifier" votre rendu **diffus** en fonction du nombre de bandes souhaités pour le cel shading (voir image ci-dessous).



Quantification (en rouge) d'un signal continu (courbe grise)

Une fois votre intensité diffuse quantifiée, vous pouvez générer votre couleur diffuse avec :
vec3(0.0, value, 0.0) pour avoir un rendu vert en cel-shading

[c] Créer/modifier la composante spéculaire du modèle de Phong
On procède ici différemment. Vous pouvez "binariser" la composante spéculaire pour ne pas avoir de bandes sur la tache spéculaire : c'est du tout ou rien. Il s'agit de seuiller la valeur. Vous pouvez mettre une valeur en dur dans votre shader.
Si vous voulez une tache spéculaire blanche, vous pouvez faire quelque chose comme :
vec3(value)

FIN de l'exercice #2

EXERCICE #3 - Textures procédurales

[5 points]

TODO

Reprenez l'exercice précédent et continuez sur le même code avec le même shader.

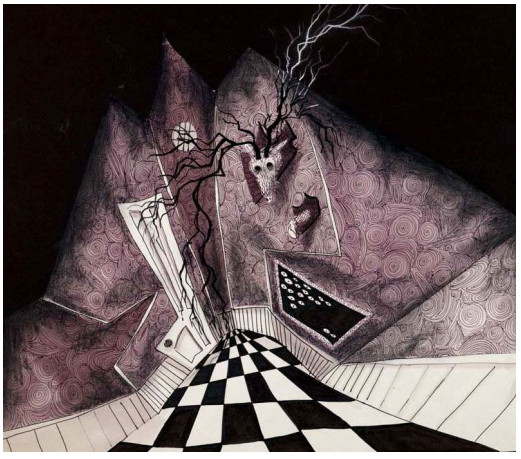
***NOTE** : si vous n'avez pas réussi à faire l'exercice précédent ou si vous commencez par celui-ci, faites l'exercice sur un tore unique et créer d'abord un lighting/shading de Phong (diffus + spéculaire). Pour créer un tore, regarder la partie **[a]** de l'exercice 1.*

NOTE : GLSL fournit des fonctions comme qui pourraient vous être utile dans cet exercice : ***round***(), ***floor***(), ***fract***(), ***step***(), ***smoothstep***()...

Concept art / direction artistique : Tim Burton (The Nightmare before Christmas)

Vous êtes dans une équipe de développeurs dans le cinéma dont le directeur artistique souhaite s'inspirer du concept art des films de Tim Burton : Beetlejuice, The nightmare before Christmas… Un point clef est la déformation des perspectives et donc des lignes (voir images ci-dessous).

Les infographistes de votre équipe devront concevoir un bijou pour les personnages comme Beetlejuice et Lydia Deetz en générant des formes procédurales déformées (voir ci-dessous) avec votre aide.

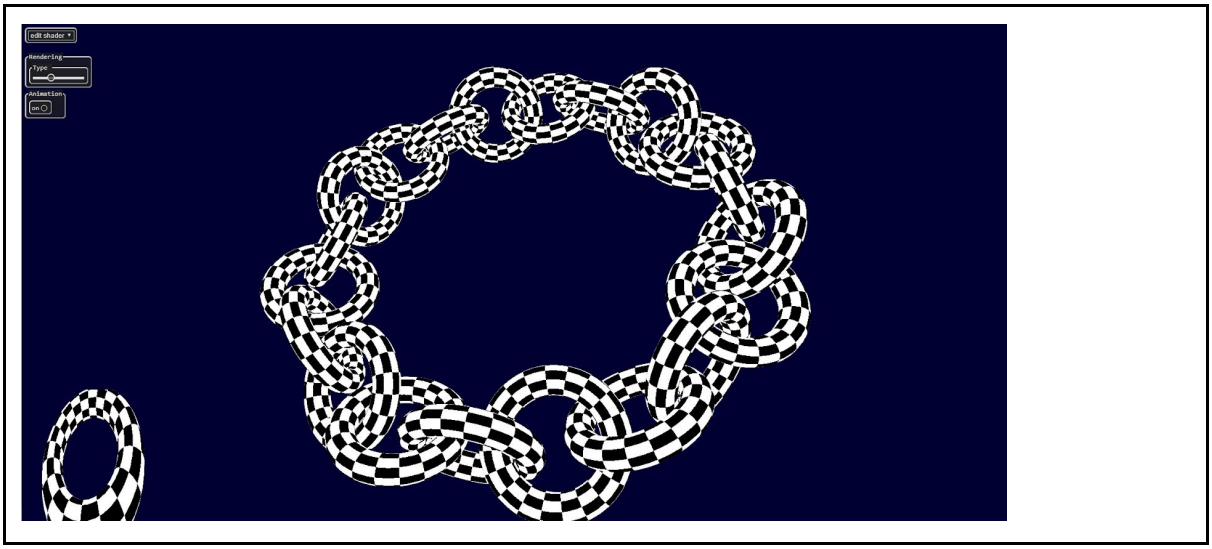


Procédez par étape :

Il est ici question de génération procédurale de texture à la sauce “shadertoy”.
***NOTE** : Il n’y a pas de FBO (framebuffer object), tout est calculé à la volée dans votre fragment shader.*

[a] Créer une grille noire et blanche

BIJOU - exemple de rendu attendu :



Paramètres utilisateurs :

- type de rendu (phong, npr, texture procédurale)
- nombre de bandes subdivisions (pour une grille : en x et en y)

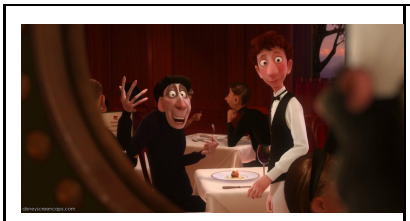
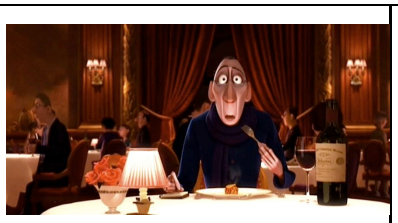
Si vous avez le temps, ajouter des sliders dans l’interface graphique pour tuiler le tore : nombre de bandes horizontales et verticales.

[b] Tim Burton effect

Appliquer des déformations spatiales.
Pour cela, il faut d’abord obtenir une information de distance grâce à des “**signed distance field**” (voir ci-dessous). Ces types de fonctions prennent en entrée des coordonnées spatiales et retourne une distance par rapport à ces bords.
Voici un catalogue de fonctions de distances 2D :
https://www.iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm
Pour faire des déformations de votre texture procédurale, vous pouvez plugger vos valeurs de coordonnées de textures, puis appliquer divers transformation de l’espace sur cette distance : pow, cos, sin, etc…

BONUS - “Surprenez moi !”

Ratatouille
Réalisez un effet à votre “sauce” !!


NOTE : je vous donne une fonction **noise**() dans votre code si vous voulez jouer avec.

NOTE : vous pouvez composer/créer/mixer de nouvelles formes de type “signed distance field” avec la fonction **min**() qui va prendre le forme procédurale la plus proche entre plusieurs. En appliquant des translations sur les “uv” en entrée des fonctions, on peut positionner les formes dans l’espace. S’il y en a plusieurs, on fait un min().