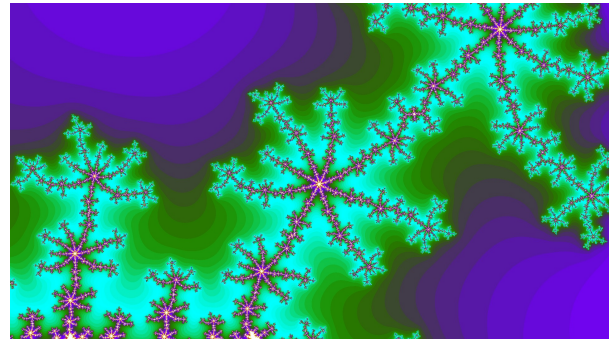
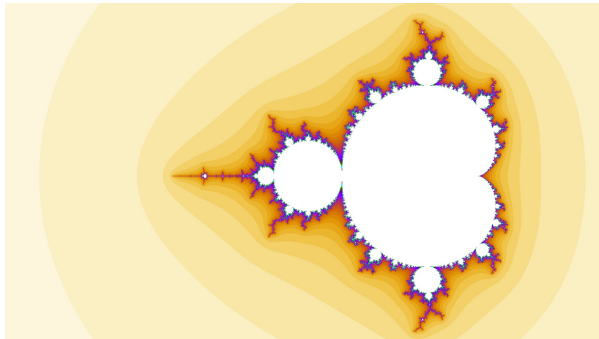


TP5 : Utilisation du Fragment shader



Introduction

L'objectif du TP est d'apprendre à utiliser le fragment shader pour calculer des images complexes.

Pour la récupérer la base de code, allez à l'adresse suivante :

<https://team.inria.fr/imagine/modelisation-synthese-dimage-ricm4-polytech-2015-2016/>.

Pour l'utiliser, suivez la démarche habituelle :

- extrayez l'archive du TP dans le dossier de votre répertoire personnel dédié aux TP de 3D (`unzip TP5.zip -d ~/TP3D/`)
- accédez au dossier du TP (`cd ~/TP3D/TP5/`)
- créez un lien symbolique vers le dossier `external` (`ln -s ../external/`)
- créez un dossier pour la compilation (`mkdir build`)
- accédez à ce dossier (`cd build`)
- lancez `cmake` (`cmake ..`)
- lancez la compilation (`make`)
- exécutez (`./polytech_ricm4_tp5`)

Vous devriez avoir la hiérarchie de fichiers ci-contre. Notez que ce TP ne requiert pas l'utilisation du dossier `models`

```

TP3D
├── external
│   └── ...
├── models
│   └── ...
├── TP1
│   └── ...
├── TP2
│   └── ...
├── TP3
│   └── ...
├── TP3 bis
│   └── ...
├── TP4
│   └── ...
├── TP5
│   ├── ...
│   ├── external [-> ../external/]
│   ├── build
│   │   └── ...
│   ├── shader
│   │   └── ...
│   ├── src
│   │   └── ...
│   └── CMakeLists.txt

```

Introduction

Prenez la peine de regarder le code source ainsi que les shaders du programme qui vous est donné. Notez l'absence de vertex buffer. Que visualise-t-on dans la fenêtre ? Comment sont transmises les positions des sommets ? À quoi correspond la couleur affichée à l'écran ?

Fractale de Mandelbrot

Une fractale est une image qui se répète en elle-même: une sous-partie de l'image contient l'image toute entière. De manière concrète, une image fractale peut se calculer grâce à une procédure itérative. Une telle image n'a pas d'utilité pratique, mais est relativement couteuse à calculer. Les cartes graphiques permettent d'effectuer de nombreux calculs en parallèle, et le calcul d'image fractale en temps réel constitue un bon moyen de le constater. Par ailleurs, la mise en place du programme de calcul et d'affichage va nous permettre de prendre en main la partie "fragment shader" d'OpenGL.

Procédure. Une des fractales les plus emblématiques est connue sous le nom d'ensemble de Mandelbrot. Cet ensemble est défini comme suit : un nombre complexe c fait partie de l'ensemble si la suite complexe (z_n) définie récursivement ainsi :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

ne diverge pas lorsque n tend vers l'infini.

En pratique, pour calculer l'image, il faut affecter une couleur à chaque pixel en fonction d'un critère déterminant la convergence de z_n . Pour ce faire, on affecte un pixel p à un nombre complexe c_p à partir duquel on calcule un nombre fixe N d'itération de la suite z . On considère une valeur seuil S . Si pour un $n < N$ donné, $\|z_n\| > S$, on affecte directement à p une couleur dépendant de n . Sinon on affecte une couleur par défaut.

L'algorithme est comme suit :

```
z = 0
pour n allant de 0 à N:
    z = z * z + c
    si |z| > S:
        stop boucle
couleur(p) = colormap(n)
```

Travail à faire. Pour calculer une image fractale il vous faut :

- créer une fonction calculant le carré d'un nombre complexe (Note : un nombre complexe sera représenté par un `vec2`)
- créer une fonction de calcul de couleur à partir d'un nombre entier (`colormap`)
- mettre en place l'algorithme ci dessus

Tout ceci dans le fragment shader.

Variantes. Différentes variantes de l'ensemble de Mandelbrot existent. Si vous voulez vous amuser, vous pouvez tester:

- de changer la formule de z : $z_{n+1} = z_n^k + c$ (ensemble de Mandelbrot d'ordre k)
- de calculer z_0 en fonction de p , avec c constant (ensemble de Julia)
- d'évaluer la divergence de la suite avec différents critères ($\|Re(z)\| + \|Im(z)\| > S$ par exemple)
- ces variantes et bien d'autres sont consultables sur le site www.shadertoy.com

Paramètre uniformes

Les paramètres uniformes sont des variables du programme principal qui sont transmises aux shader. Contrairement aux variables lues depuis un VBO, les valeurs des paramètres uniformes sont les mêmes pour toutes les instances d'un shader.

Contrôle de caméra Afin de pouvoir vous déplacer dans l'ensemble de Mandelbrot, vous pouvez calculer le nombre associé à p de manière à prendre en compte un déplacement, voire un zoom. Mettez en place un système de contrôle de la vue avec les flèches en vous inspirant de l'interface du TP2.

Pour ceci, vous pouvez par exemple déclarer dans le C++ deux variables globales représentant les positions des coins supérieur gauche (p_0) et inférieur droit (p_1). En passant ceux-ci aux fragment shader par l'intermédiaire de variables uniformes, vous pouvez calculer la position de p relativement au rectangle défini par p_0 et p_1 . Ensuite, il ne vous reste qu'à relier l'appui sur des touches du clavier (les flèches par exemple) à la modification de p_0 et p_1 : les translater pour déplacer la caméra, ou bien les rapprocher l'un de l'autre pour zoomer.

Animation En passant une variable uniforme représentant le temps, il est facile de créer une animation dans le shader. Il faut pour ceci faire varier un paramètre de l'algorithme (comme par exemple z_0 , ou le mapping de couleur) en fonction de la variable représentant le temps.

Anti aliasing

Si vous observez les zones à très grand contraste, vous observerez que celle-ci sont très pixelisée : c'est l'aliasing. Ce défaut intervient lorsque la fréquence d'un signal est égale à plus de la moitié de la fréquence d'échantillonnage. Pour y pallier, il faut augmenter la fréquence d'échantillonnage, autrement dit augmenter le nombre d'échantillons par pixel. Pour ce faire, il suffit de calculer plusieurs valeurs à différents endroits du pixel et de les moyenner. Mettez en place un système de sur-échantillonnage.