Ludovic D’ANjou-MAdore

Simon Lepage

Jérôme Pagé

Jonathan Simard

**Projet d'intégration en Sciences informatiques et mathématiques**

201-201-RE, gr.00001

Curved\_Fractals

**Document de Conception**

Travail présenté à

M. Walid Boulabiar

Département d'Informatique

Cégep Limoilou

Le 9 février 2018

# 1. Description du projet

**1.1 Objectifs**

Un programme de visualisation de fractales dans un espace non-euclidien (donc doté de courbure) permettrait d’introduire une compréhension de concepts mathématiques abstraits et complexes (Courbure d’une variété riemannienne et l’influence d’une métrique sur celle-ci) en permettant à l’utilisateur de visualiser en temps réel la courbure qui découle d’une métrique spécifiée. Une compréhension des fractales est acquise en même temps.

**1.2 Description détaillée**

*Concepts présents dans l’applications*

L’application met en application premièrement les concepts mathématiques suivants :

* Les Variétés Riemanniennes dans leurs constructions.
  + La spécification d’une métrique comme généralisation du produit scalaire
    - , au point p
  + L’expression des Symboles de Christoffel en fonction du tenseur métrique
    - <https://fr.wikipedia.org/wiki/Symboles_de_Christoffel#Expression_%C3%A0_partir_du_tenseur_m%C3%A9trique>
  + L’influence des symboles de Christoffel sur la dérivée covariante
    - https://fr.wikipedia.org/wiki/D%C3%A9riv%C3%A9e\_covariante#Champ\_de\_vecteurs\_2
  + L’influence de la dérivée covariante sur le tenseur de courbure
    - <https://fr.wikipedia.org/wiki/Courbure#D%C3%A9finition_du_tenseur_de_courbure>
  + Calcul d’une immersion de l’espace qui contient les fractales dans l’espace écran
    - <https://en.wikipedia.org/wiki/Riemannian_manifold#The_pullback_metric>

Les fractales seront transformés point par point grâce à l’immersion déduite à partir de la métrique. La courbure à la position du curseur sera calculé ensuite.

* Des méthodes d’approximation ou des librairies de calculs symbolique pour calculer l’immersion.
  + Méthode de Newton? Apprentissage? Calcul symbolique?

**1.3 Concepteurs rôles et justifications**

*Rôles :*

*Secrétaire : Jonathan Simard*

*Scrum Master :* *Simon Lepage*

*Responsable des livrables : Jérôme Pagé*

*Directeur de produit : Ludovic D'Anjou-Madore*

*Justifications :*

Jonathan Simard en tant que secrétaire en raison de son accès à internet quasi constant.

Simon Lepage en tant que Scrum master en raison de son expérience en tant que chef d’équipe à son travail.

Jérôme Pagé en tant que responsable des livrables en raison de son expérience avec git, notre logiciel de versionnage.

Ludovic D'Anjou-Madore en tant que directeur de produit en raison de sa bonne vision d’ensemble de ce que le programme doit faire et pourrais faire.

**1.4 Type D'application**

Windows

**1.5 Technologies impliquées**

Eclipse, GPU avec support pour openGL 3.2 et plus

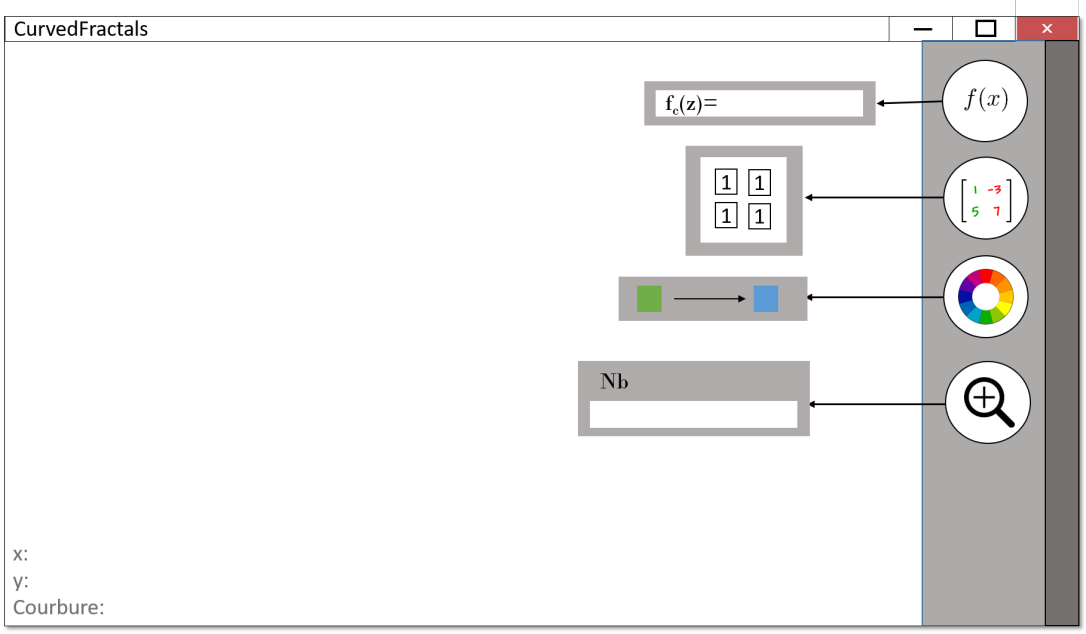
Librairies : Jmonkey (openGL), JavaFX

**1.6 Références et documentation**

<http://graphics.cs.ucdavis.edu/~joy/NSF-IIS-0916289/Papers/Obermaier2012b.pdf>

# 2. Prototypes de l’application

**2.1 Prototype de l’interface**



# 3. User stories

|  |  |
| --- | --- |
| Acteur ou rôle : | Utilisateur |
| Scénario ou story : | En tant qu’utilisateur, je voudrais je veux être capable de spécifier l’équation d’une fractale, sa distribution de couleurs et la voir affichée à l’écran. |
| Détail ou description : | 1. Spécification de l’équation décrivant la fractale    1. Reçoit l’équation de l’utilisateur    2. Vérifier la syntaxe    3. On écrit l’équation dans le shaders    4. Recharge JMonkey    5. Affichage du résultat 2. Spécification de la couleur    1. Reçoit les extremums de couleurs de l’utilisateur avec un colorpicker    2. Traduction en vecteur rgb    3. Envoi des deux vecteurs aux shaders |
| Tests d’acceptation : | Confirmation visuelle du fonctionnement en affichant l’ensemble de Mandelbrot |
| Complexité : | 5 |
| Effort : | 2j/personne **ou** 3 |
| Commentaires : |  |

|  |  |
| --- | --- |
| Acteur ou rôle : | Utilisateur |
| Scénario ou story : | En tant qu’utilisateur, je veux être capable de manipuler la fractale à l’écran. |
| Détail ou description : | * 1. Lors du déplacement de la souris avec le bouton gauche enfoncé, mesurer la translation   2. Encoder la translation dans une matrice.   3. Envoyer la matrice au shaders   4. Dans les shaders, effectué la translation sur la matrice   5. Lorsque du roulement de la mollette de la souris, enregistrer le zoom demandé   6. Effectué le zoom demandé  1. 3.1. Recevoir le zoom de l’utilisateur et le délai entrer chaque zoom   3.2. Enregistrer la position de la souris  3.3 Effectuer une répétition de zoom le nombre de fois demandé par l’utilisateur   1. 4.1. Lorsque l’utilisateur pèse sur la touche escape, affiché la fractale dans sa position originale (zoom de 0 et translation de 0) |
| Tests d’acceptation : | Afficher l’ensemble de Mandelbrot et confirmation visuelle du fonctionnement du zoom et de la translation avec la souris ainsi que du « reset » avec la barre espace |
| Complexité : | 5 |
| Effort : | 2j/personne **ou** 3 |
| Commentaires : |  |

|  |  |
| --- | --- |
| Acteur ou rôle : | Utilisateur |
| Scénario ou story : | En tant qu’utilisateur, je veux pouvoir spécifier les composantes du champ tensoriel métrique et voir les effets de la courbure de celui-ci sur une fractale |
| Détail ou description : | 1. Recevoir les chaines de caractères de l’utilisateur représentant les composantes du champ tensoriel métrique. 2. Valider la structure et la syntaxe des composantes et les interpréter pour construire le champ tensoriel métrique en Java. 3. Construire une texture 3D (2D Array Textures ou un tableau 2x2 de matrices 2x2 sur OGL 4.3 et +) en évaluant le tenseur à chaque pixel (les composantes x et y de la texture représentent la position de chaque pixel et le niveau (pour le 2DTextures Array) représente la nième composante du tenseur a un point (pour le tableau 2x2, en fixant la position, on obtient une matrice, la représentation locale du tenseur métrique). 4. Envoyer la structure de données choisie sur le shader et calculer la déformation pour chaque pixel. |
| Tests d’acceptation : | Afficher l’ensemble de Mandelbrot et confirmation visuelle de la déformation. |
| Complexité : | 7 |
| Effort : | 3j/personne **ou** 4 |
| Commentaires : | -Lors du changement de tenseur spécifié par l’utilisateur, on recalcule une texture et on met à jour la référence dans le shader  -lors d’un zoom, on doit « zoomer » aussi les coordonnés d’accès à la représentation en mémoire du tenseur. Donc, si on utilise une texture 3d, l’interpolation ce fait automatique pour chaque niveau. Si on utilise un tableau de matrice, dans chaque direction, on doit interpoler entre les composantes. (par exemple : voir l’interpolation linéaire dans la description des concepts) |