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

**Rapport final**

Travail présenté à

M. Walid Boulabiar

Département d'Informatique

Cégep Limoilou

Le 18 mai 2018

# **Description du projet**

* 1. **Concepteurs : rôles, justifications et tâches**
     1. **Rôles :**

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

Scrum Master : *Jonathan Simard*

Secrétaire : *Simon Lepage*

Responsable des livrables : *Jérôme Pagé*

* + 1. **Justifications :**

*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 pourrait faire. Est également celui qui en connaît le plus sur la géométrie riemannienne et les autres concepts mathématiques utilisés dans l’application.

*Jonathan Simard* en tant que Scrum master en raison de son expérience comme chef d’équipe à son travail.

*Simon Lepage* en tant que secrétaire en raison de ses bonnes capacités de rédaction.

*Jérôme Pagé* en tant que responsable des livrables en raison de son accessibilité et de son accès à Internet quasiment 24 heures sur 24.

* + 1. **Tâches (en plus de celles liées au rôle) :**

*Ludovic D’Anjou-Madore*: Travaille sur l’ensemble de l’application, surtout sur le modèle. Est responsable de la conception de l’algorithmie mathématique et de son développement (vérificateur de syntaxe pour les formules, calculs pour déformer la fractale, la déplacer et la zoomer, etc.). Travaille aussi sur les shaders openGL.

*Jonathan Simard* : Travaille sur l’intégration des différentes composantes nécessaires au bon fonctionnement de l’application (librairies, structure MVC). Contribue au code des shaders openGL.

*Simon Lepage*: Travaille sur l’ensemble de la documentation nécessaire pour le projet (backlogs de produit et de sprints, rapports) et sur l’interface de l’application (FXML, style CSS, intégration au code).

*Jérôme Pagé*: Travaille sur l’interface de l’application (FXML, style CSS, intégration au code, ajout de fonctionnalités) et sur l’utilisation faite par l’utilisateur de l’application (événements de la souris, interaction avec l’interface, …).

* 1. **Objectifs**

Un programme de visualisation de fractales dans un espace non-euclidien (doté d’une certaine 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, et l’application permet d’appliquer les principes de quelques concepts mathématiques (géométrie riemannienne, champ tensoriel métrique).

* Conception et réalisation d’une application programmée principalement en Java permettant l’affichage et la manipulation de fractales.
* Intégration de connaissances provenant de domaines connexes abordées par le programme d’études (ici, les mathématiques) à l’informatique.
* Utiliser les méthodes Agile et Scrum pour organiser et gérer quotidiennement la progression du projet.

* 1. **Description détaillée**

CurvedFractals est un logiciel libre et ouvert, développé dans le cadre du projet d’intégration du DEC en Sciences Informatiques et Mathématiques.  
Ce logiciel, ultime expérience esthétique, prétend humblement permettre de visualiser l’interaction entre deux objets mathématiques, soit les fractales et les variétés riemanniennes (au sens très faible du terme: la responsabilité de doter la variété d’une métrique **riemannienne** est laissé à l’utilisateur, ainsi que celle de s’assurer que la paramétrisation qu’il fournit donne effectivement une variété différentielle, mais le résultat s’affiche même si rien ne respecte les propriétés).

Effectivement, en plus d’afficher une fractale d’équation arbitraire spécifiée par l’utilisateur, celui-ci peut spécifier la structure (tenseur métrique et paramétrisation) de l’espace dans lequel cette fractale sera affichée. Évidemment, l’utilisateur peut aussi se déplacer et zoomer sur la fractale (de manière automatique ou manuelle). Les esthètes confirmés seront ravis de pouvoir pigmenter la fractale de la façon qu’ils le souhaitent, tandis que les technologues jouiront de l’affichage (presque) en temps réel grâce à l’utilisation active (OpenGL + shaders) de leur GPU.

* + 1. **Concepts présents dans l’application**

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](https://fr.wikipedia.org/wiki/Symboles_de_Christoffel#Expression_à_partir_du_tenseur_métrique)
  + 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](https://fr.wikipedia.org/wiki/Courbure#Définition_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 en fonction de la métrique pour obtenir l'effet de courbure.

* + La transformation se fait par l'endomorphisme de courbure construit à partir du tenseur de courbure (tenseur métrique).
  + Interpolation linéaire pour passer d’une description discrète à une description linéaire du champ tensoriel métrique :
    - pour 0<= i <=1

On remarque bien, à la complexité des équations en jeu, l'utilisation de concepts complexes, comme la dérivation covariante jusqu'au nombre effarent de terme, une fois la notation d'Einstein enlevée. La transmission de ceux-ci au GPU (carte graphique) sera un problème technique en soi. Sans cette transmission, la seule alternative est de les calculer sur le CPU (processeur), perdant beaucoup des avantages d'une carte graphique...

Ensuite, les concepts informatiques utilisés dans l'application sont principalement ceux reliés à l'imagerie par ordinateur. En effet, pour bénéficier du plein potentiel des cartes graphiques modernes, la majorité des calculs concernant le calcul de la fractale et la transformation de celle-ci en fonction de la courbure seront exécutés sur le GPU, par l'entremise des *shaders* (openGL), des petits programmes exécutés pour chaque sommet d'un modèle et pour chaque pixels de l’écran, parallélisés, qui permettent d'effectuer les transformations (ici appliquer la courbure sur la fractale) et effets (ici l'affichage de la fractale en soi) voulus de manière extrêmement efficace. De plus, envoyer une partie substantielle des calculs sur la carte graphique a des effets bénéfiques d'un point de vue technique par exemple, en permettant d'alléger grandement la structure du modèle MVC.

* + 1. **Fonctionnement**
       1. **Mécanismes internes**

Pour afficher une fractale en haute définition, il faut, sur chaque pixel de l’écran, itérer l’équation de la fractale et évaluer si cette suite tend à diverger (après environ 600 itérations, on peut approximativement considérer que si elle ne diverge pas maintenant, elle ne le fera pas). Donc, sur un écran d’une résolution de 1080p (le standard actuel), ça commence à être lourd, étant donné qu’il faut itérer sur 2 073 600 pixels… Heureusement, le progrès technologique étant ce qu’il est, une bonne partie des ordinateurs relativement modernes sont dotés d’une carte graphique accélérant beaucoup ce genre de calculs.

Toutefois, la seule manière de fournir les équations nécessaires aux shaders est de les écrire directement dans leur fichier. Sur le CPU, cela aurait été évité, mais cela aurait engendré la perte du temps réel lors de l’affichage. Notre approche nous rajoute un délai de compilation des shaders (les shaders étant assez légers, l’effet est négligeable), mais nous garde en temps réel ailleurs.

Donc, il est **impératif** que les fichiers genericShaderFrag.glsl, genericMat.j3md, initialFrag.glsl et initialMat.j3md restent dans **/src/vue**. De plus, les fichiers j3md doivent rester inchangés, tandis que les fichiers .glsl doivent garder les commentaires qui indiquent la position des équations à inscrire, en plus de ne pas déranger le code des méthodes de lesquels ils sont contenus. Tout le reste peut être modifié sans altérer le fonctionnement de l’affichage. (Il est recommandé de ne pas changer les systèmes de coordonnées dans les shaders, c-à-d les normalisations des variables d’entrées, car cela rend la translation et le zoom inconfortables d’utilisation).

* + - 1. **Fonctionnement mathématique**

TODO : CTRL C+V du README.md sur GitHub, section Fonctionnement mathématique

* 1. **Comparatif avec l’existant et l’idée de départ**

L’application existante finale correspond à l’idée de départ du projet. Les fonctionnalités, l’interface et l’algorithmie conçues au début se trouvent dans l’application. Seulement une fonctionnalité a été enlevée du développement et ne fait pas partie de l’existant, soit l’utilisation d’une texture 3D. Cette fonctionnalité a d’ailleurs été remplacée par une fonctionnalité similaire permettant d’atteindre le même résultat, qui était d’utiliser le tenseur métrique afin de déformer les longueurs de la fractale affichée. Somme toute, nous avons bien évalué le temps nécessaire à la réalisation de l’ensemble de l’application. Peu de sacrifices ont été faits, et le fonctionnement global de l’application est satisfaisant.

* 1. **Technologies utilisées**
* **Outils et langages :** Eclipse, SceneBuilder, Notepad++, Visual Paradigm, Microsoft Office (Word, PowerPoint et Project), GPU avec support pour openGL 3.2+, Java, FXML, GLSL, CSS3
* **Librairies :** [jMonkey](https://github.com/jMonkeyEngine/jmonkeyengine) (openGL), [JavaFX](https://docs.oracle.com/javafx/2/api/index.html), [JME3-JFX](https://github.com/empirephoenix/JME3-JFX), [exp4j](https://github.com/fasseg/exp4j)

# **Développement**

* 1. **Méthodes de développement**
     1. **Vue d’ensemble du fonctionnement du cadre de développement**

Le fonctionnement de notre cadre de développement est simple. Nous utilisons Trello pour organiser les quelques Sprints qui ont lieu et les tâches à effectuer lors de chaque Sprint. Ces tâches sont déterminées selon les User Stories conçues au début du projet. TODO – À CONTINUER

* + 1. **Déroulement des travaux**
    2. **Difficultés rencontrées**
    3. **Outils de développement**
  1. **Organisation du travail**
     1. **Fonctionnement de l’équipe**
     2. **Difficultés rencontrées et découvertes**
     3. **Forces, faiblesses et acquis**
  2. **Validation de l’application**

# **Contribution personnelle**

* 1. **Ludovic D’Anjou-Madore**
     1. **Apprentissages pour résoudre les problèmes que j’ai dû rencontrés**
     2. **Évaluation de ma contribution à l’équipe**
     3. **Évaluation de notre** ☭) **équipe et de mon expérience en équipe**
  2. **Jonathan Simard**
     1. **Apprentissages pour résoudre les problèmes que j’ai dû rencontrés**
     2. **Évaluation de ma contribution à l’équipe**
     3. **Évaluation de notre** ☭) **équipe et de mon expérience en équipe**
  3. **Simon Lepage**
     1. **Apprentissages pour résoudre les problèmes que j’ai dû rencontrés**
     2. **Évaluation de ma contribution à l’équipe**
     3. **Évaluation de notre** ☭) **équipe et de mon expérience en équipe**
  4. **Jérôme Pagé**
     1. **Apprentissages pour résoudre les problèmes que j’ai dû rencontrés**
     2. **Évaluation de ma contribution à l’équipe**
     3. **Évaluation de notre** ☭) **équipe et de mon expérience en équipe**

# **Conclusion et perspectives du projet**

* 1. **Évaluation de la production**
  2. **Évaluation des acquis**
  3. **Avenir du projet**

# **Annexe**