**« Backlog » de produit : Curved Fractals**

Conçu par : JeeGo

## Équipe :

Jérôme Pagé – Responsable des livrables

Simon Lepage – Secrétaire

Jonathan Simard – Scrum master

Ludovic D’Anjou-Madore – Directeur de produit

## Légende :

* Vert, indique que ces scénarios sont réalisés.
* Jaune, indique que ces scénarios font partie du « backlog de sprint » courant.
* Rouge, problème ou questionnement important qui demande une rencontre d’équipe.
* Aucune couleur, indique que ces scénarios ne sont pas encore faits, toujours en réflexion et modifiables.

## « Backlog » de produit

|  |  |
| --- | --- |
| **1** | |
| Acteur ou rôle : | Utilisateur |
| Scénario ou story : | User Story 1 |
| Détail et description des **items** à faire : | **Spécification de l’équation décrivant la fractale**  Faire l’interface.  Faire le backbone du MVC.  Recevoir l’équation de l’utilisateur.  Vérifier la syntaxe de l’équation (s’assurer que l’équation contient les variables « z » et « c », et une syntaxe admissible par GLSL).  Configurer et produire une infrastructure d'affichage avec Jmonkey.  Faire un shader générique sur lequel on ajoutera l'équation.  Concevoir un systeme de gestion des Materials (produire un Material et pour chaque équation envoyée, la fusionner avec une copie du shader generique et le charger dans le Material de base et réafficher).  Afficher un Material.  **Spécification de la couleur**  Configurer le menu de la couleur.  Traduction en vecteur rgb  Envoie des vecteurs aux shaders |
| Tests d’acceptation : | Confirmation visuelle du fonctionnement en affichant l’ensemble de Mandelbrot |
| Complexité : | 5 |
| Effort : | 2j/personne **ou** 3 |
| Commentaires : |  |

|  |  |
| --- | --- |
| **2** | |
| 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, effectuer 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 : | La fonctionnalité d’enregistrer la position de la souris a été enlevée de ce sprint, et elle ne sera peut-être pas implémentée du tout. La translation et le zoom sont fonctionnels, mais ils devront être peaufinés plus tard (au sprint 4 ou 5). |

|  |  |
| --- | --- |
| **3** | |
| 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) |