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

Conçu par : JeeGo

## Équipe :

**(JP) : Jérôme Pagé** – Secrétaire

**(SL) : Simon Lepage** – Responsable des livrables

**(JS) : Jonathan Simard** – Scrum master

**(LDM) : 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 : | **(US1)** 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 : | **(US2)** En tant qu’utilisateur, je veux être capable de manipuler la fractale à l’écran. |
| Détail ou description : | 1. Déplacer la fractale (translation) avec la souris    1. Qui et temps       1. Nom(s) : JP, SL, JS, LDM       2. Temps : 4h    2. Préconditions       1. Présence des shaders qui permettront la modification de la position (translation) de la fractale en se servant d’une matrice.       2. Présence d’un événement associé au « drag » de la souris dans le fichier FXML de la vue.    3. Règles d’affaires       1. Lors du déplacement de la souris avec le bouton gauche enfoncé, la différence des positions départ/arrivée (translation) doit être mesurée.       2. Encoder la translation mesurée dans une matrice (Matrix4f de JMonkey).       3. Envoyer la matrice aux shaders.       4. Dans les shaders, effectuer la translation sur la fractale, en se servant de la matrice reçue.    4. Règles d’affaires alternatives       1. Il devient difficile de déplacer la fractale lorsque le zoom est élevé. La translation doit donc s’adapter au zoom pour être constante peu importe ce dernier.       2. La translation s’inverse selon le signe du zoom. Elle doit être corrigée afin que le mouvement de la souris soit toujours le même, et non inversé.    5. Tests d’acceptation       1. S’assurer manuellement que la translation est naturelle et qu’elle est constante peu importe le zoom courant sur la fractale.    6. Post-conditions       1. Je peux, en utilisant la souris, déplacer (effectuer une translation) la fractale à l’écran et voir son déplacement 2. Effectuer un zoom ponctuel sur la fractale avec la souris    1. Qui et temps       1. Nom(s) : JP, SL, JS, LDM       2. Temps : 4h    2. Préconditions       1. Présence d’un événement associé au « scroll » de la molette de la souris dans le fichier FXML de la vue.    3. Règles d’affaires       1. Stocker le zoom (positif ou négatif) dans une variable vectorielle (Vector2f de JMonkey) qui sera utilisée par les shaders.       2. Dans les shaders, effectuer le zoom correspondant sur la fractale.    4. Règles d’affaires alternatives       1. Le zoom ne dépend pas de la position de la souris, mais cette fonctionnalité a été enlevée lors du déroulement du sprint.       2. Le zoom n’est pas à l’infini. Si je zoome trop, j’aperçois des morceaux pixellisés de fractale, qui sont sa finalité. Il devrait être possible de pouvoir effectuer un zoom et toujours voir la fractale.    5. Tests d’acceptation       1. Vérifier visuellement que le zoom s’effectue de manière satisfaisante : on ne voit pas de pixels et la fractale s’affiche et se zoome à l’infini.    6. Post-conditions       1. Lorsque j’utilise la molette de la souris, je peux effectuer un zoom sur la fractale affichée à l’écran. Je peux zoomer et dézoomer à l’infini. 3. Effectuer un zoom automatique selon le nombre de grossissements spécifiés    1. Qui et temps       1. Nom(s) : JP, SL       2. Temps : 2h    2. Préconditions       1. Présence d’un onglet permettant la saisie d’un nombre de zooms dans l’interface.    3. Règles d’affaires       1. Stocker le nombre de zooms voulu dans une variable qui sera utilisée par le Thread du zoom automatique. Ce nombre est entré par moi-même.       2. Effectuer le nombre de zooms demandé. Il y a utilisation d’un Thread qui permet d’effectuer plusieurs zooms ponctuels d’une même grandeur, soit le zoom automatique. Ces zooms s’effectuent au même intervalle de temps.       3. La matrice de zoom de la fractale est mise à jour à chaque zoom ponctuel.    4. Règles d’affaires alternatives       1. Il faudrait ajouter une fonctionnalité permettant d’arrêter le zoom automatique lorsque je rentre un nombre de zooms trop grand.    5. Tests d’acceptation       1. Vérifier que le nombre de zooms entré correspond au nombre de zooms ponctuels effectués.       2. Vérifier que les zooms s’effectuent au même intervalle de temps.       3. S’assurer que le zoom automatique est facile d’utilisation (calibration de la grandeur des zooms).    6. Post-conditions       1. Je suis capable d’effectuer un zoom automatique sur la fractale en entrant le nombre de zooms ponctuels que je désire faire. 4. Faire un « reset » de la fractale (valeurs de zoom et translation par défaut)    1. Qui et temps       1. Nom(s) : JP, SL       2. Temps : 2h    2. Préconditions       1. Présence d’un écouteur sur les touches du clavier.    3. Règles d’affaires       1. Lorsque j’appuie sur la touche Échap, la fractale doit s’afficher selon ses valeurs par défaut (zoom et translation de 0).       2. Écrire une méthode qui réinitialise les valeurs de zoom et de translation de la fractale. Cette méthode modifie les valeurs des matrices correspondantes.    4. Règles d’affaires alternatives       1. To update…    5. Tests d’acceptation       1. Vérifier visuellement que la fractale s’affiche comme elle s’est affichée pour la première fois lors du démarrage de l’application.    6. Post-conditions       1. Je peux réinitialiser la fractale affichée (zoom et translation) en appuyant sur la touche Échap de mon clavier. |
| 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 : | **(US3)** 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 chaînes de caractères de l’utilisateur représentant les composantes du champ tensoriel métrique.    1. Qui et temps       1. Nom(s) : JP, SL       2. Temps : 2h    2. Préconditions       1. Présence d’un onglet dans l’interface permettant la saisie des composantes du champ tensoriel métrique.    3. Règles d’affaires       1. Stocker les chaînes entrées par l’utilisateur dans les TextField correspondants dans des variables String.    4. Règles d’affaires alternatives       1. Pas de bugs ou autre. Rien à corriger.    5. Tests d’acceptation       1. Vérifier que la saisie des composantes s’effectue correctement et qu’elles sont facilement accessibles par le shader.    6. Post-conditions       1. Je suis capable de spécifier les composantes à utiliser pour le champ tensoriel métrique. 2. Valider la structure et la syntaxe des composantes du tenseur métrique et les interpréter pour construire le champ tensoriel métrique en Java.    1. Qui et temps       1. Nom(s) : JS, LDM       2. Temps : 2h    2. Préconditions       1. Méthode permettant de construire le champ tensoriel métrique en Java dans le modèle de l’application (dans une nouvelle classe qui manipule et représente le tenseur métrique).    3. Règles d’affaires       1. Calculer la nouvelle longueur du vecteur normal selon la surface affichée (fractale).       2. Déformer la fractale selon le nouveau vecteur normal (transforme la longueur des segments).    4. Règles d’affaires alternatives       1. Pas de bugs ou autre. Rien à corriger.    5. Tests d’acceptation       1. Vérifier que le champ tensoriel métrique construit est valide : le vecteur normal est changé et on peut voir sur la fractale que les longueurs des segments ont été changées.    6. Post-conditions       1. L’application peut construire le champ tensoriel métrique en Java selon les composantes que j’ai spécifiées. 3. ~~Construire une texture 3D~~     1. ~~Qui et temps~~       1. ~~Nom(s) : JS, LDM~~       2. ~~Temps : 4h~~    2. ~~Préconditions~~       1. ~~Être capable d’accéder à un champ tensoriel métrique valide, stocké dans une variable.~~    3. ~~Règles d’affaires~~       1. ~~Stocker la texture 3D dans une variable qui sera envoyée au shader.~~       2. ~~Utiliser des 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. ~~Règles d’affaires alternatives~~       1. ~~To update…~~    5. ~~Tests d’acceptation~~       1. ~~Vérifier qu’une texture 3D peut être construite et que ses attributs correspondent à ceux attendus selon les composantes du champ tensoriel.~~    6. ~~Post-conditions~~       1. ~~L’application est capable de construire une texture 3D selon le champ tensoriel métrique.~~ 4. Envoyer l’équation de la courbure au shader et calculer la déformation pour chaque pixel grâce à JMonkey.    1. Qui et temps       1. Nom(s) : JS, LDM       2. Temps : 4h    2. Préconditions       1. ~~Être capable de construire une texture 3D.~~    3. Règles d’affaires       1. Recevoir l’équation écrite par l’utilisateur via l’interface graphique (TextField)       2. L’équation est envoyée au shader et ce dernier calcule la déformation pour chaque pixel de la fractale.    4. Règles d’affaires alternatives       1. Pas de bugs ou autre. Rien à corriger.    5. Tests d’acceptation       1. Vérifier visuellement que la fractale se déforme selon la structure de données reçue.       2. Vérifier dans les shaders temporaires (runtime) si l’équation de la courbure est interprétée et le calcul de déformation se fait.    6. Post-conditions       1. Je suis capable de voir la déformation causée par la texture 3D sur la fractale affichée à l’écran. Cette déformation change selon les composantes du champ tensoriel. |
| Tests d’acceptation : | 1. Afficher l’ensemble de Mandelbrot et confirmation visuelle de la déformation.    1. Vérifier que les composantes du champ tensoriel métrique entrées par l’utilisateur construisent un tenseur métrique qui transforme la fractale affichée (transforme les longueurs selon le tenseur).    2. Vérifier que l’équation pour la paramétrisation sphérique entrée par l’utilisateur déforme la fractale en la courbant (vérifier que la courbure agit comme un globe terrestre vu du dessus). |
| 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 se 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) |