Compte rendu tp3 et tp4

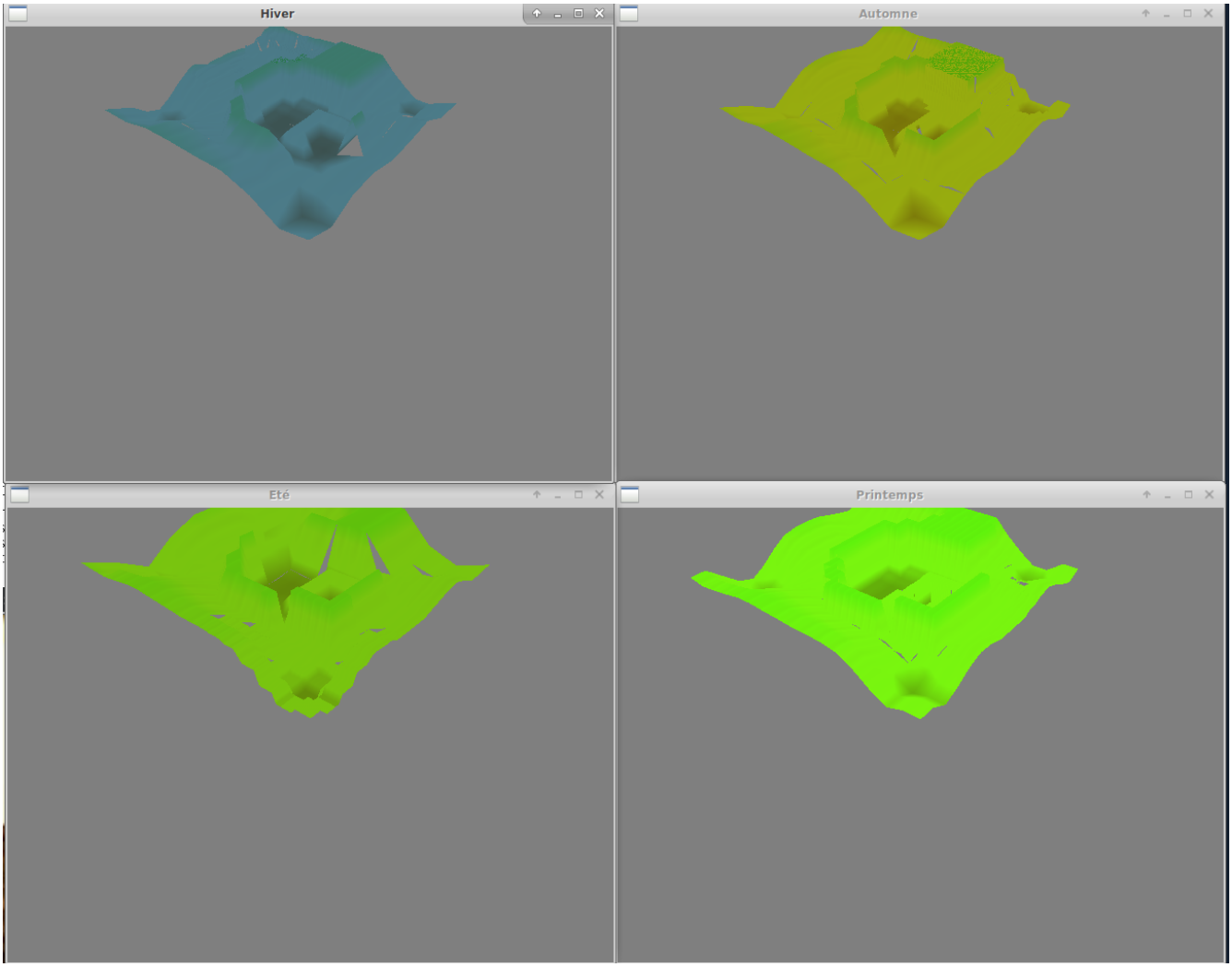
par Quentin Leroit le 15/11/2018

lien github : https://github.com/AmazingPipot/MoteurDeJeux

Le but de ce tp est de réaliser un gestionnaire de scène et un gestionnaire de niveau de détail.

Dans un premier temps, il est nécessaire d'apprendre à travailler sur plusieurs fenêtre en même temps, travaillant chacune de manière autonome tout en pouvant communiquer entre elle.

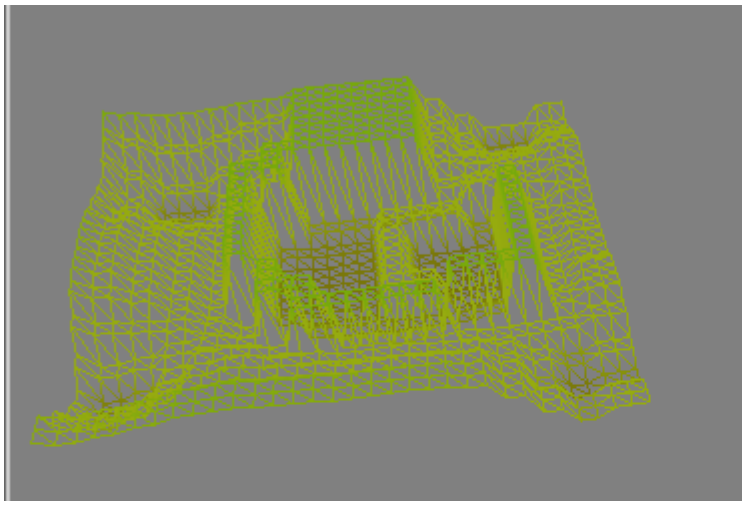
Pour cela, nous allons simuler le changement de saison. Il est donc nécessaire de créer 4 fenêtres, auquelles nous associerons un objet *connect()*. Celui-ci, permet de transmettre pour chaque fenêtre, un timer et un script à la fin de ce premier. Le script *SaisonSuivante()* dans la classe *mainWidget* va changer la saison puis définir une nouvelle couleur d'affichage dans le shader via la fonction *basiqueSaison()*.



Remarque : la qualité de l'image est différente selon les écrans, en effet le lod est en cours de fonctionnement de manière aléatoire, sur chacune.

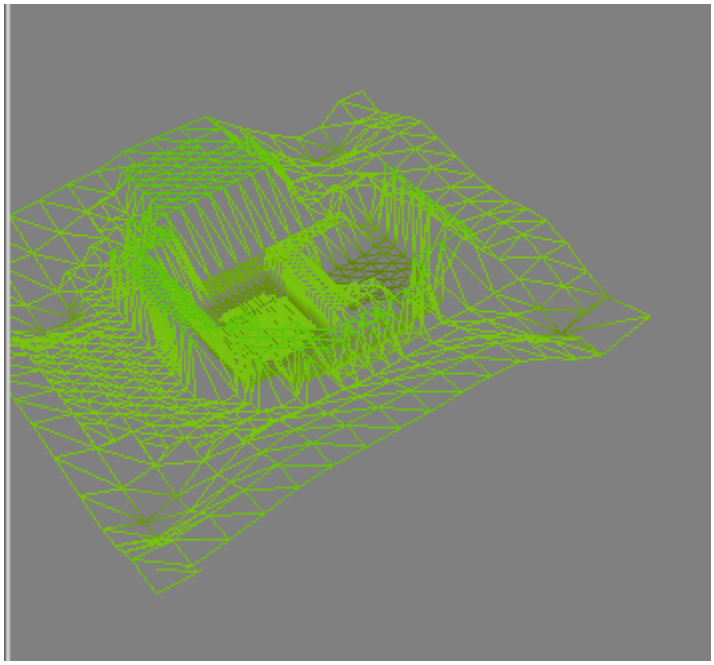
Dans un second temps, nous allons implémenter un quadtree. Celui doit être capable d'ajouter de l'information en rajoutant des points de manière régulière. Pour cela la classe quadtree est créée. Elle se compose des fonctions *initFils()* qui crée les enfants du quadtree et *feuille()* qui retourne le nombre de quadtree.

A partir de là, dans la classe *geometryEngine* la fonction *initMapQuadtree()* crée le quadtree de départ puis va indexer les vertices construit depuis la fonction *constructionVertex()*.



Remarque : pour afficher les arrêtes, il suffit d'appuyer sur la touche espace. Actuellement la profondeur du quadtree est de 4.

La seconde partie du tp consiste à creer un LOD, pour cela il faut rajouter un nouveau constructeur dans la classe *quadtree* celui-ci prend en compte la position pour appliquer le LOD ainsi que des paramêtres de distance indiquant quand la récursion s'arrête et ou elle commence. De plus ce constructeur est associé à la fonction *lod()* celle-ci va déterminer dans quel inteval le quadtree va être recalculé.



Remarque : pour changer le constructeur, il suffit de choisir le constructeur désiré dans la fonction *initMapQuadtree* de la classe *geometryEngine.*

Cependant, on constate des erreurs de raccordement dans la texture, du fait que des points sont rajoutés sur des arrêtes d'un côté mais pas relié aux autres de l'autre côté.

Questions Bonus :

Pour réaliser des volumes à partir des particules, on peut créer 2 heigmaps une originale et une modifiable. Le contact de la particule avec la surface va faire varier la hauteur du point. En fonction de cette variation on modifie la texture. Ainsi, l'accumulation de neige fait augmenter l'altitude et fait blanchir la texture, tandis que l'eau la creuse et la rend bleu.

Une méthode de simplification 3D simple peut consister à supprimer régulièrement des points mais risque de faire disparaître de l'information utile.

Pour afficher une scène infinie, un moyen est de créer une équation de plan (l'équation permettant de calculer une infinité de points mais de ne pas perdre les valeurs précédentes) auquel on lui associe un bruit contrôlé pour créer de la variation. Ensuite on se place au centre de la zone d'affichage puis lorsqu'on se “déplace”, on ne se déplace pas rééllement mais on recalcule les coordonnées selon l'équation auquel on ajoute le bruit. De cette manière on peut revenir en arrière et repasser par les même paysage de manière “infinie”.

L'optimisation peut passer par un LOD moins grand ou moins précis.