TP Modélisation
CPE

4ETI IMI

1 Organisation du programme

1.1 Organisation et rôle des différents répertoires

L'archive contient 3 répertoires à sa racine en plus du fichier `CMakelists.txt` permettant de compiler le projet.

- **data/** Contient les données externes du programme (fichiers de maillages, textures, etc).
- **shaders/** Contient les différents shaders utilisés dans le programme.
- **src/** Contient le code source du programme.

Le répertoire `src/` est lui même organisé de la manière suivante:

- **local/** Contient les fichiers spécifique au TP en question. Ces fichiers sont potentiellement différents pour chaque TP et à adapter en fonction de vos besoins.
- **lib/** Contient les classes et fonctionnalités d'aide développées pour les TP de synthèse d'images à CPE. Ces fichiers fonctionnels permettent de développer plus rapidement vos projets. Cette bibliothèque regroupe un ensemble de classes et fonctionnalités (en format libre GPL) que vous êtes libre d'utiliser et de modifier pour vos TP à CPE ou dans vos travaux futurs. Les fichiers de cette bibliothèque seront généralement réutilisés pour plusieurs TP (bien que pouvant évoluer suivant les versions).
- **external/** Contient les bibliothèques de code externes non standards.

Le répertoire `local/` est scindé en deux sous répertoires:

- **interface/** Contient les fichiers gérant la partie GUI et fenêtres de l'application. Ces fichiers ne sont généralement pas à modifier sauf si vous souhaitez modifier l'interface.
- **scene/** Contient la description de la scène 3D. Il contient à la fois la fonction de chargement des données et la fonction d'affichage en boucle. Il s'agit généralement du fichier ou vous réaliserez la majeure partie de votre code.

Le répertoire `lib/` contient quant à lui plusieurs sous répertoires. Nous pourrons noter les suivants:

- **3d/** Contient un ensemble de classes de vecteurs et de matrices utilisant la syntaxe proche du GLSL. Ces classes sont utilisées pour manipuler des points et vecteurs du plan et de l'espace. Notez qu'il s'agit d'une version simplifié de la librairie GLM. Vous utiliserez directement ces classes et pouvez à votre guise ajouter des fonctionnalités à cette partie.
- **common/** Contient la gestion d'erreur unifiée pour l'utilisation de la bibliothèque (gestion simplifiée d'assertion et d'exception avec affichage de la trace d'exécution). Vous n'aurez généralement pas à modifier cette partie de la bibliothèque.

- **interface/** Contient la gestion des classes utilisées pour simplifier la manipulation d'une interface en Qt à la souris. Vous n'aurez généralement pas à modifier cette partie de la bibliothèque.
- **mesh/** Contient les classes de gestions de maillages (données, chargement de fichiers, etc). Vous utiliserez directement ces classes et pouvez à votre guise ajouter des fonctionnalités à cette partie.
- **opengl/** Contient les classes et fonctions directement liées à l'affichage OpenGL. Les classes `mesh` étant codé indépendamment d'OpenGL, ces classes font le lien entre les structures de données sur le CPU et les envoies sur le GPU. Vous utiliserez directement ces classes et pouvez à votre guise ajouter des fonctionnalités à cette partie.

Notez qu'il n'est pas nécessaire de connaître l'intégralité des fichiers de cette librairie. Cependant, celle-ci est codée de manière à ce que vous puissiez à tout moment être en mesure de vous référer au code et de comprendre celui-ci.

1.2 Lancement du projet

Question 1 *Assurez que que le programme compile et s'exécute correctement. Notez que pour s'exécuter, le programme doit être lancé depuis le répertoire racine (le chemin vers les fichiers de données et shaders étant données sous la forme* `data/nom` *et* `+shaders/nom`*).*

2 Construction d'une surface paramétrique discrète

Placez vous dans le fichier `local/scene/scene.cpp` et observez à la fois la fonction de chargement des maillages (chargement à partir d'un fichier, ou construction directe d'un maillage par ajout de sommets). Observez également les méthodes d'affichage. Notez la construction suivante:

1. Création d'un maillage sous la forme d'une classe `mesh` (Observez cette classe, que contient-elle, quels sont ces méthodes).
2. Envoie des données sur le GPU par le biais de la classe `mesh_openl` puis affichage dans la méthode `scene::draw_scene` précédé de la phase de mise en place de données des shaders.

Question 2 *Créez un nouveau maillage de type* `mesh`*. À l'aide des méthode* `add_vertex()` *et* `add_triangle_index` *créez un maillage modélisant le plan suivant:* $(x, y, z) = (u, v, 0)$ *pour* $(u, v) \in [0, 1]^2$*. On considérera dans un premier temps un échantillonnage de 5x5 sommets (dans les directions u et v).*

Remarque. On évitera l'utilisation de variables flottantes dans les boucles for (précision limité), préférer utiliser uniquement des variables entières. ex.

```
int Nu = ...
int Nv = ...
for(int ku=0 ; ku<Nu ; ++ku)
{
  for(int kv=0 ; kv<Nv ; ++kv)
  {
     ...
```

Réfléchissez à l'organisation de vos triangles liants les sommets, faites un schéma au préalable. Vous pouvez vous inspirer de la figure 1 illustrant un terrain (l'application suivante).

Question 3 *Factorisez votre code dans une fonction prenant en paramètre* $(x_{\min}, x_{\max}, y_{\min}, y_{\max}, N_u, N_v)$ *qui généralise la création d'une surface plane telle que* $x \in [x_{\min}, x_{\max}]$, $y \in [y_{\min}, y_{\max}]$, *et échantillonnée suivant* $N_u \times N_v$ *valeurs.*

3 Construction d'un champ de hauteurs

Un *champ de hauteurs* est une surface analysée comme une fonction $z = f(x, y)$.

Question 4 *Faites en sorte que la coordonnées* z *de votre surface varie en fonction des coordonnées* (u, v) *par exemple à l'aide de fonction trigonométrique. Prenez soin de faire varier* (u, v) *sur des coordonnées normalisées (ex. sur* $[0, 1]$ *) et non pas comme des coordonnées entières. Notez qu'il n'est pas trivial de donner une apparence de terrain accidenté à l'aide d'une somme de fonctions standards.*

Question 5 *À la place de votre fonction* $z(u, v)$*, appliquez désormais un bruit de Perlin sur la coordonnée* $z = perlin(s, t)$ *(utilisez la classe* `perlin`*). De manière similaire, prenez soin de faire varier les paramètres* (s, t) *sur des coordonnées normalisées (par exemple sur* $[0, 3]$*). Modifiez les paramètres du bruit de Perlin afin d'obtenir l'apparence d'un terrain.*

Question 6 *Adaptez la texture et/ou la couleur de votre terrain à votre guise. N'oubliez pas qu'il est possible d'appliquer une couleur en chaque sommet comme une fonction de la position ou de l'altitude.*

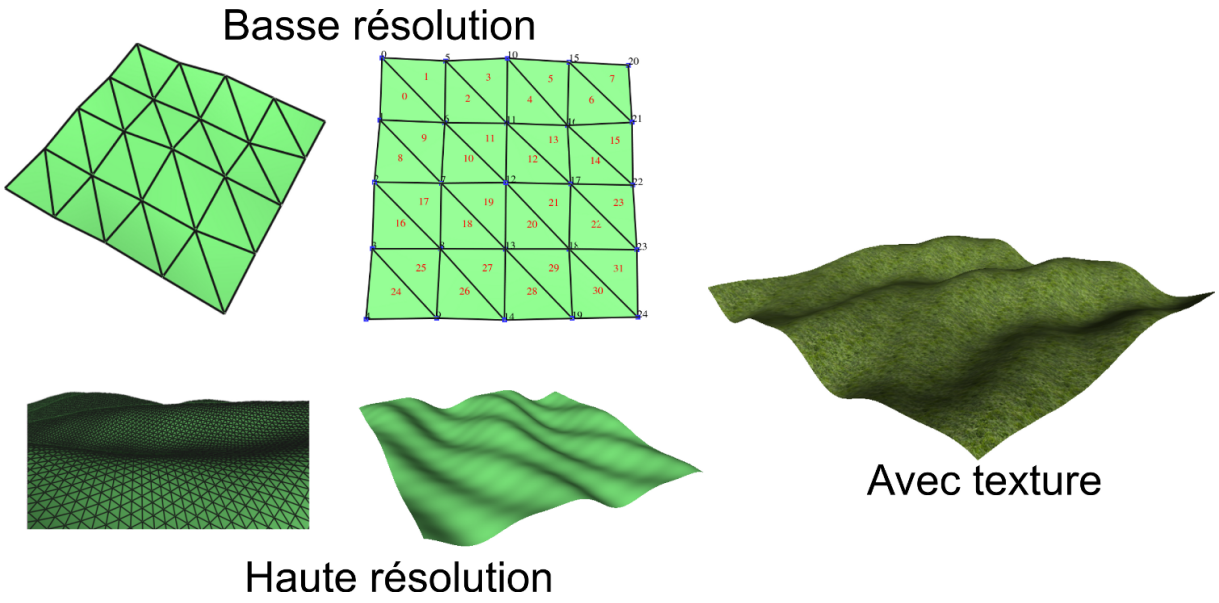

Figure 1: Exemple d'un terrain et ordonnancement des sommets et triangles.

4 Deformation locale d'une surface

Question 7 *Déformez localement votre terrain pour y placer une piste d'atterrissage d'avion.*

Vous devez donc aplatir votre terrain localement pour y placer la piste.

Pour cela, vous pourrez procéder de cette manière:

- Définir la trajectoire du centre de la piste d'atterrissage dans les coordonnées du plan (x,y).
- Aplatir localement les sommets du terrain en fonction de la distance à cette trajectoire.
- Ajouter un (ou plusieurs) maillage modélisant votre piste.

Un exemple d'une telle scène est illustré en fig. 2.

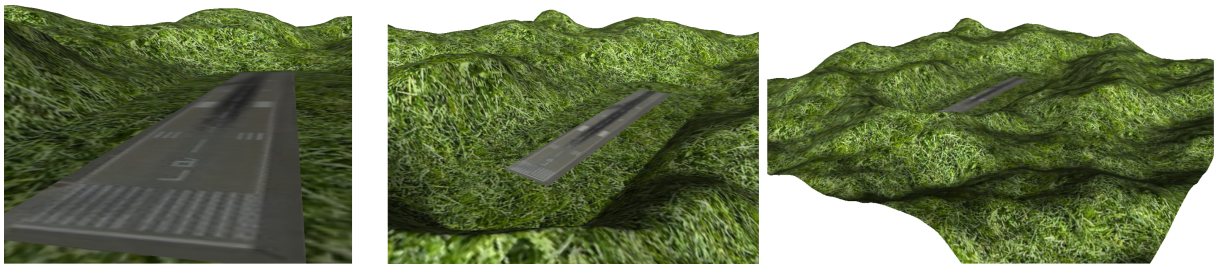

Figure 2: Exemple d'un terrain et d'une piste d'atterissage.

5 Surface paramétrique

La tour de contrôle de la piste est modélisée par un cylindre.

Question 8 *Mettez en place une tour de contrôle.*

Pour cela, vous implémenterez une fonction qui génère un cylindre vertical texturé de hauteur h et de rayon r.

Les données h et r devront être des paramètres de votre fonction. On passera également en paramètre le nombre N d'échantillons à prendre en compte pour discrétiser la partie circulaire du cylindre. Notez que l'on pourra laisser le cylindre creux.

Un exemple de structure en cylindre est illustré en fig. 3.

6 Maillage

La tour de contrôle est attaquée par un dinosaure.

Question 9 *Placez un maillage modélisant ce type d'attaque.*

Le dinosaure est mutant, sa tête est animée.

Question 10 *Animez la tête de celui-ci tel que celle-ci gonfle et se dégonfle localement.*

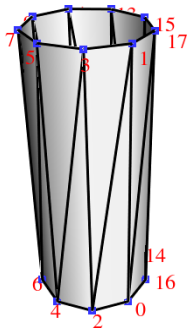

Figure 3: Exemple d'une structure cylindrique avec $N = 9$.

Pour cela, on viendra translater les sommets le long de leur normales respectives.

Soit $\mathbf{n}_i$ la normale en un sommet i donné. Soit $\mathbf{x}_i^0$ les coordonnées du sommet i avant déformation, et $\mathbf{x}_i$ les coordonnées du même sommet après déformation. On a alors

$$\mathbf{x}_i(t) = \mathbf{x}_i^0 + \mu(t)\,\mathbf{n}_i\ ,$$

pour tous i appartenant à la tête de l'animal. μ étant un scalaire dépendant du temps à déterminer.

Un exemple de telle déformation est illustré en fig. 4.

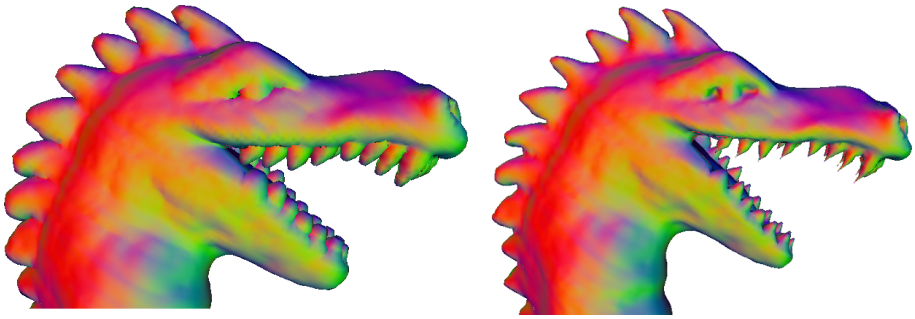

Figure 4: Exemple d'un gonflement/degonflement locale par translation des sommets le long de la normale.

7 Lissage

On souhaite désormais lisser le maillage du dinosaure afin d'estomper les détails de sa surface.

Question 11 *Lisser celui-ci en utilisant l'approche du lissage Laplacien.*

Pour cela, considérons un sommet i de coordonnées $\mathbf{x}_i$. Soit $\mathcal{V}_i$ les sommets voisins de i (on parle de 1-voisinage). Lisser le maillage revient à appliquer itérativement la déformation suivante:

$$\forall i\ ,\quad \mathbf{x}_i = \mathbf{x}_i + \frac{\lambda}{\mathrm{card}(\mathcal{V}_i)} \sum_{j \in \mathcal{V}_i} (\mathbf{x}_i - \mathbf{x}_j)\ ,$$

avec $\lambda \in [0, 1]$, et $\mathrm{card}(\mathcal{V}_i)$ le nombre de voisins de i.

Pour vous aider, dans un premier temps, construisez une structure de donnée permettant de stocker l'indice des voisins d'un sommet i donné.

Notez que l'on parle de lissage Laplacien, car l'itération suivante revient à résoudre l'équation d'évolution

$$\frac{\partial f}{\partial t} = \lambda \triangle f \ ,$$

avec f vallant ici les coordonnées, t étant discrétisé par le nombres d'itérations, et l'opérateur $\triangle$ étant discrétisé spatialement par la différence issue du 1 voisinage.