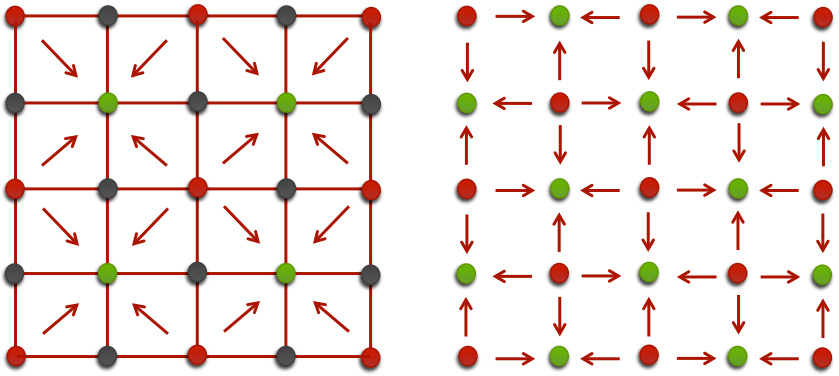
Synthèse d'image
Génération d'un paysage virtuel

Dans ce TP de 2 séances, vous allez expérimenter sous Matlab des méthodes de synthèse d'images. Le moteur de lancer de rayons Mitsuba sera utilisé pour le rendu. Une évaluation par binôme sera organisée lors de la dernière séance de TP.

1 Subdivision du terrain

L'objectif est ici de générer un paysage réaliste à partir d'une description de départ très grossière d'un terrain (fichier `terrain.m`). Ce terrain sera subdivisé n fois grâce à l'algorithme vu en cours de *diamond-square*. Cet algorithme procède en deux étapes que nous rappelons ici :



A chaque fois qu'une nouvelle valeur est calculée, on la perturbe d'une valeur aléatoire dont l'amplitude devra diminuer à chaque itérations. Avant subdivision la carte du terrain fait une taille de $h \times l$ (dans la figure ci-dessus 3×3), après subdivision cette taille devient $H \times L$ où $H = 2h - 1$ et $L = 2l - 1$ (dans la figure ci-dessus 5×5). Cela signifie que la taille est quasiment doublée à chaque itération il faudra veiller à ne pas trop itérer.

TODO Implémenter une fonction `Subdivise(terrain, alpha)` qui prend en argument le terrain non subdivisé `terrain` et une amplitude de perturbation `alpha` et renvoie le terrain subdivisé.

TODO Implémenter une fonction `GenererTerrain(initterrain, nsubdiv, alpha, lambda)` qui prend en argument le terrain initial `initterrain`, le nombre de subdivisions `nsubdiv`, une amplitude de perturbation initiale `alpha` et un facteur d'aténuation `lambda` et qui renvoie le terrain subdivisé.

2 Visualisation

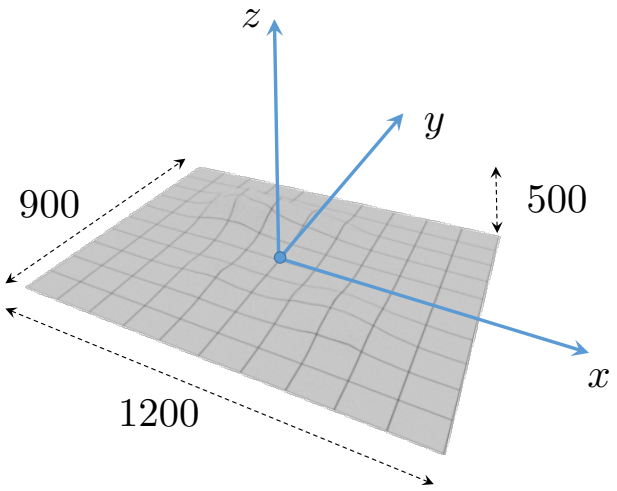
Afin de visualiser ce terrain pour vérifier que l'algorithme de subdivision fonctionne, vous pouvez utiliser la commande `surf` de Matlab. Pour un rendu plus joli, nous allons utiliser Mitsuba. Mitsuba [1] est un moteur de rendu expérimental qui possède de nombreuses fonctionnalités. Pour décrire la scène que le moteur doit rendre, un fichier XML doit être fourni. Dans les fichiers distribués, un exemple de fichier XML permettant de rendre un terrain représenté sous forme d'image de hauteur est donné (fichier `texture-lookat.xml`).

Ce fichier contient des arguments qui peuvent être donnés au moment de l'appel à Mitsuba. Une fonction `RendreTerrain(terrain, texture, outfile, origin, target)` vous est aussi fournie. Elle permet de passer les arguments et exécuter la commande Mitsuba. Voici la signification des arguments :

1. `https://www.mitsuba-renderer.org`

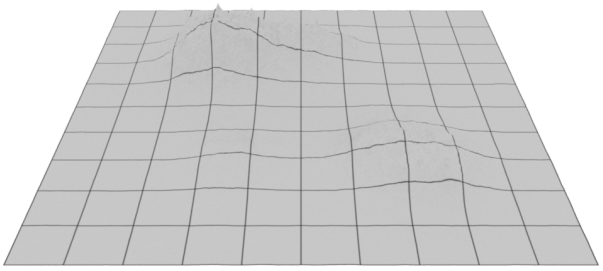
— `terrain` : le terrain précédemment calculé (sous forme de nom de fichier)
— `texture` : la texture à plaquer sur ce terrain
— `outfile` : le nom du fichier dans lequel doit être mis l'image du rendu
— `origin` : un vecteur 3D indiquant la position de l'observateur
— `target` : un vecteur 3D indiquant le point visé par la caméra

Afin de vous aider à positionner votre caméra, voici le gabarit de la scène telle qu'elle a été paramétrée dans le fichier `texture-lookat.xml` :



TODO Exporter la matrice de terrain en un format reconnu par Mitsuba et qui encode la composante niveau de gris sur 16 bits (par exemple en png 16 bits). Il y aura peut-être des changements d'échelle à effectuer ainsi que des changements de type. Vous pouvez ensuite lancer la fonction `RendreTerrain` qui vous est donnée. Vous utiliserez la texture fournie `grille.png`.

Voici le genre de résultat que vous pouvez obtenir :



Nous voyons la forme du terrain mais sans ombrage, il est difficile de voir le relief. C'est dû au fait que nous avons utilisé pour le rendu un éclairage de type *envmap* constante. Cela signifie que l'environnement émet des rayons dans toutes les directions depuis tous les points de l'espace. Nous allons pour l'instant rester avec ce type de rendu mais en essayant de mieux faire voir le relief.

3 Fabrication d'une texture

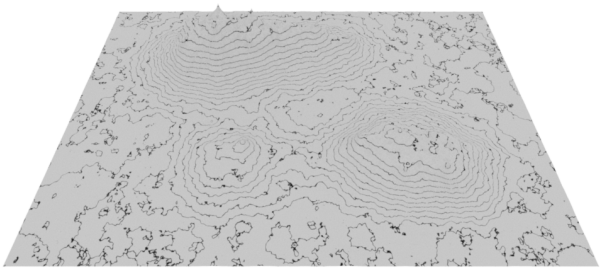
La texture aura la même taille que l'image du terrain mais avec une dimension de plus dont la taille sera 3 (une valeur par composante RGB).

3.1 Courbes de niveaux

La première texture que nous allons contruire va uniquement dépendre de l'altitude et va consister à faire apparaître des lignes de niveaux. On peut choisir par exemple de mettre en valeur les altitudes tous les 25 mètres. Cela signifie que lorsqu'un point est proche de cette valeur nous allons le colorier en noir et sinon en blanc. L'intensité du noir peut être fonction de la proximité avec la valeur (avec un dégrapé de gris) pour éviter les effets d'aliasing.

TODO Implémenter la fonction Matlab qui construit la texture de courbe de niveau et dont le prototype est `CourbeNiveau(terrain,inter).terrain` est la matrice de terrain, `inter` est l'intervalle de visualisation des lignes de niveaux. La fonction renvoie une texture de la même taille que le terrain donné en entrée mais qui possède trois composantes RVB.

Vous devriez obtenir ce genre de rendu :

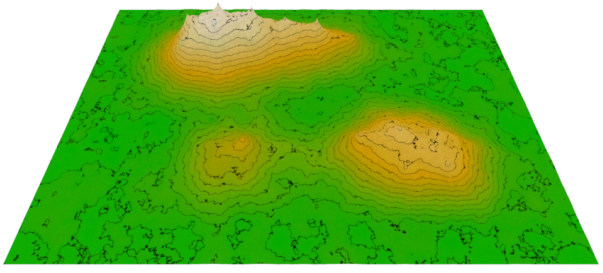


3.2 Rampe de couleur

Afin de peaufiner le style « carte Michelin » nous allons ajouter une rampe de couleur en fonction de l'altitude. Nous allons associer à chaque altitude une couleur par l'intermédiaire d'une image de hauteur 1 pixel (fichier `colorramp.png`). Pour cela il suffit de faire correspondre l'altitude avec la coordonnée dans l'image de la rampe et d'aller piocher la couleur correspondante.

TODO Implémenter une fonction `CourbeNiveauRamp( terrain, inter, rampfile )` assez proche de la précédente mais qui intègre la rampe de couleur.

Résultat attendu :



3.3 Ombrage

Nous allons nous mêmes calculer un ombrage plutôt que de laisser Mitsuba le faire (ce serait trop simple !). Pour cela il va falloir donner une position à la source de lumière et prendre conscience du positionnement du

terrain dans l'espace. Le dernier élément qui sera nécessaire à construire un ombrage est la normale au terrain en chaque point du terrain.

TODO Implémenter une fonction qui permet de transformer notre matrice d'altitude en une matrice de points 3D. Cette fonction s'appellera `ConstruitPoints3D (terrain, xmin, xmax, ymin, ymax )` et renverra une matrice de la même taille que le terrain mais avec une troisième dimension de taille 3 (pour les 3 composantes x, y et z).

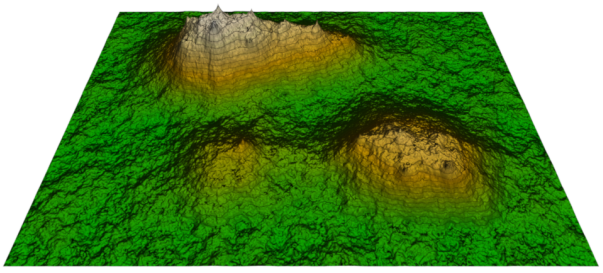
La fonction qui permet de calculer les normales au terrain est fournie car une implémentation sous forme de boucle serait catastrophique en Matlab. Elle s'appelle `ConstruitNormales(points)` et prend en argument le résultat de la fonction précédente. Elle renvoie les normales au terrain.

TODO Implémenter la fonction :

```
Ombrage( terrain, inter, rampfile, normales, points, lumiere)
```

qui va compléter le calcul de la texture par un ombrage basique diffus (composante diffuse de l'ombrage de Phong, cf. polycopié de cours).

Résultat attendu :



4 Fabrication d'une matrice de transformation

Vous avez jusqu'à présent utilisé la fonctionnalité de Mitsuba qui permet de fabriquer automatiquement la matrice de transformation correspondant à une position de l'observateur et d'un point visé. Nous allons maintenant utiliser une autre fonctionnalité de Mitsuba qui permet de donner directement cette matrice 4×4.

Notons o l'origine (position de la caméra) et c la cible regardée. Cette matrice doit correpondre au changement de repère suivant :

— La troisième direction w du repère est celle reliant o à c
— La première direction u du repère correspond à la direction opposée aux abscisses dans le repère image
— La deuxième direction v du repère correspond à la direction des ordonnées dans le repère image
— La translation t correspond à o

Vous pouvez vous servir de la fonction `cross` de Matlab qui permet de calculer un produit vectoriel entre deux vecteurs donnés en argument.

Au final la matrice sera construite de la manière suivante :

$$\begin{pmatrix} u_x & v_x & w_x & t_x \\ u_y & v_y & w_y & t_y \\ u_z & v_z & w_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

L'argument matrix dans Mitsuba doit être donné par lignes.

TODO Implémenter la fonction :

```
RendreTerrainMatrix(terrain, texture, outfile, origin, target)
```

qui prend exactement les mêmes arguments que la fonction `RendreTerrain` mais qui va utiliser le fichier `texture-matrix.xml` comme fichier de rendu. Vous devez donc fournir à Mitsuba la matrice de transformation telle que présentée précédemment.

5 Bonus

Si toutes les questions précédentes ont été traitées, vous pouvez maintenant implémenter d'autres fonctionnalités, comme par exemple :

- Un ombrage plus complet (composante spéculaire, attention nécessite la position de l'observateur)
- Un lac (ou une mer suivant la nature de la scène) à une altitude donnée qui va aussi influencer la couleur de la texture (cf image ci-dessous)
- L'ajout d'objets 3D dans la scène (Mitsuba permet d'importer des fichiers au format OBJ)
- Un script permettant de faire plusieurs rendus de la même scène sous plusieurs points de vue : une animation
- *etc.* (la seule limite étant que l'on ne peut pas vous mettre plus de 20 comme note)