Institut Universitaire de Technologie
Aix*Marseille Université

Imagerie Numérique
M 4102 - Synthèse d'images 2

3. Terrains 3D

DUT INFO 2ème année 2016-2017

Sébastien THON

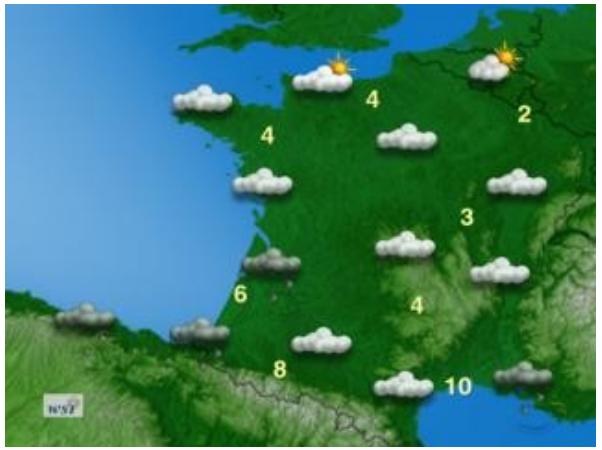
IUT d'Aix-Marseille Université, site d'Arles
Département Informatique

Définition d'un terrain 3D

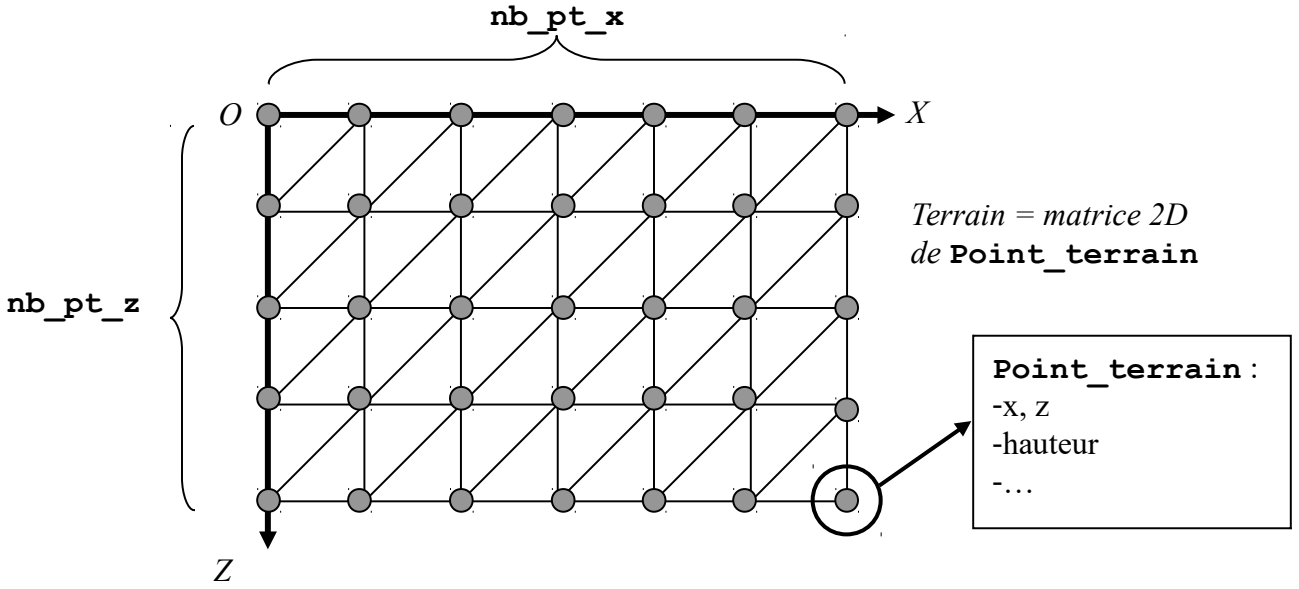
MNT : Modèle Numérique de Terrain
(*DEM : Digital Elevation Model*)

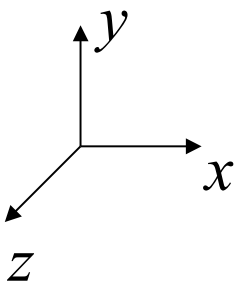
Nombreux domaines d'utilisation :

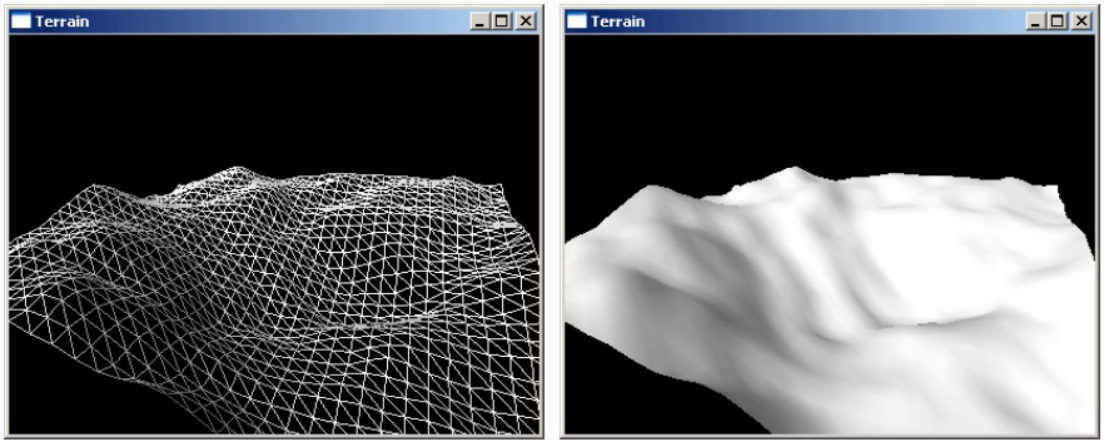
- simulateurs
- jeux vidéos
- cinéma
- visualisation scientifique
- météo
- ...



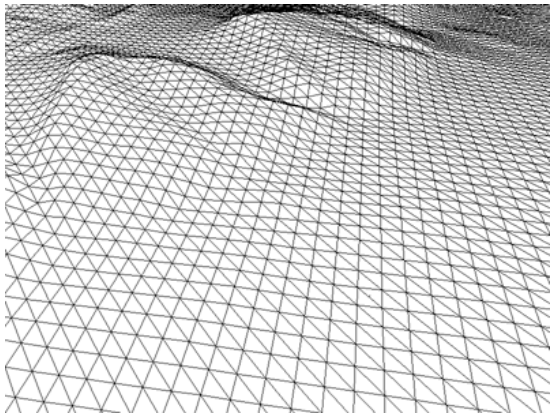
Un terrain est représenté avec une carte d'élévations. Selon le principe le plus simple, cette carte est uniforme (les points sont séparés de la même distance dans le plan xz).







Plaquage de texture sur un MNT



MNT

+

Orthophoto

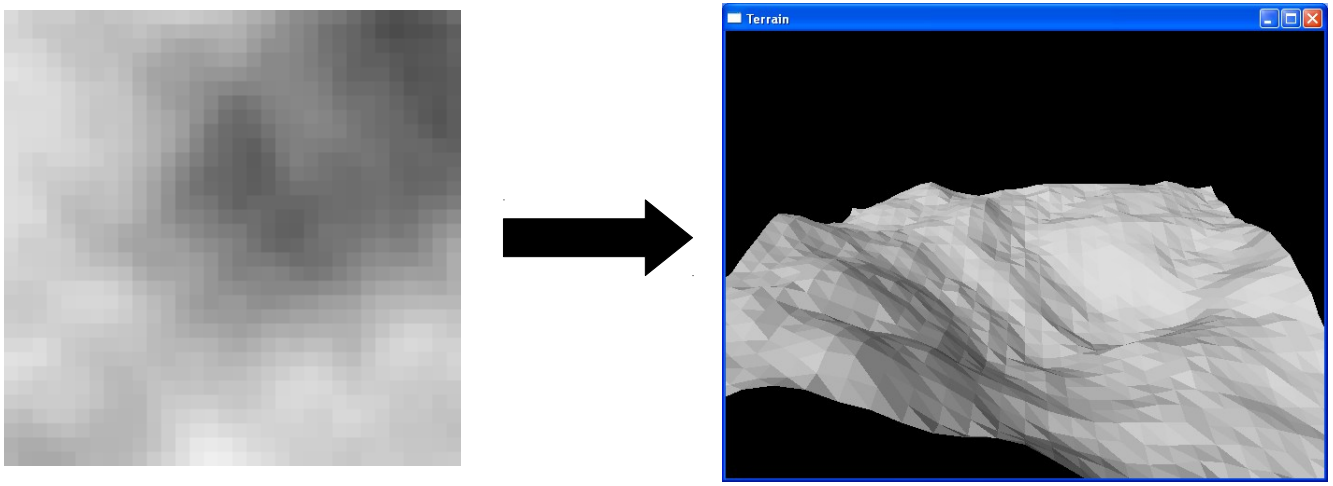
Terrain 3D texturé

Comment obtenir une carte d'élévation

- A partir de données du monde réel, obtenues par mesures (relevés topographiques au sol, LIDAR, photogrammetrie, etc.).
  Ex: **BD ALTI** de l'Institut Géographique National (IGN) ; données de la mission **SRTM** de la NASA ; etc.

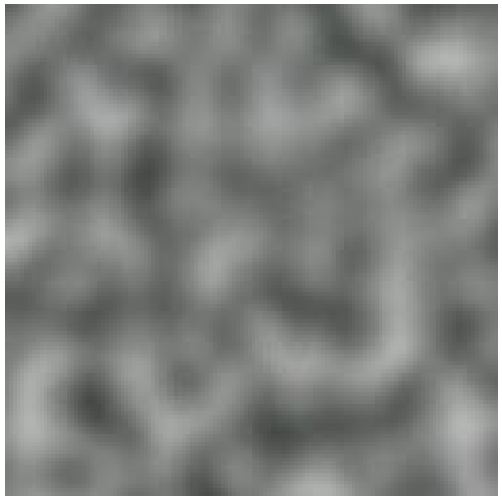
- Modélisé à la main en 3D dans un logiciel d'images de synthèse.

- Modélisé à la main en dessinant une image 2D (généralement en niveaux de gris : un pixel correspond à un point de la carte, la valeur du niveau de gris donne l'altitude du point).

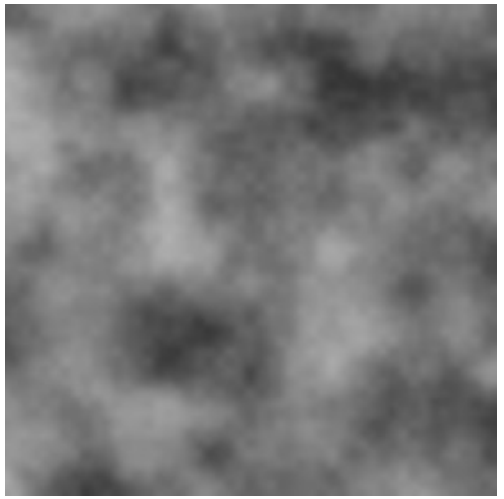


- Généré procéduralement avec un algorithme (valeurs pseudo aléatoires, bruit de Perlin, sommes de sinus, patchs de Bézier, …).

Bruit de Perlin : fonction (2D, 3D, 3D+temps, etc.) qui interpole des valeurs précalculées pour donner une valeur qui évolue pseudo aléatoirement dans l'espace (plus éventuellement le temps).



Bruit de Perlin



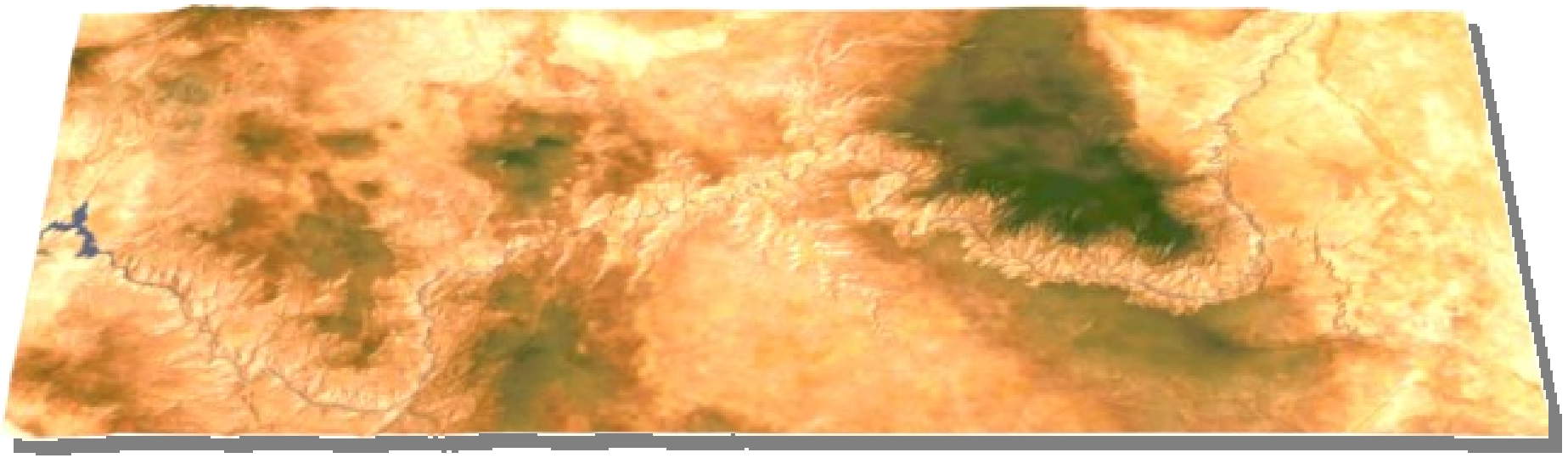
Turbulence de Perlin, obtenue en additionnant plusieurs bruits à plusieurs échelles (octaves)

http://www.noisemachine.com/talk1/

Problème

Un terrain peut parfois représenter des millions de faces.

→ Impossible d'afficher en temps réel une zone de plusieurs km^2 (nécessaire pour des simulateurs, surtout de vol)



Grand Canyon
4,097 x 2,049 sommets ~ 16.7 millions de triangles

Il existe de nombreuses techniques de représentation de terrains ; c'est un domaine de recherche scientifique très actif, dont les travaux sont ensuite utilisés dans de nombreux domaines (simulation, réalité virtuelle, jeu vidéo, …)

Vous trouverez une liste assez exhaustive des techniques de représentation de terrain 3D sur :

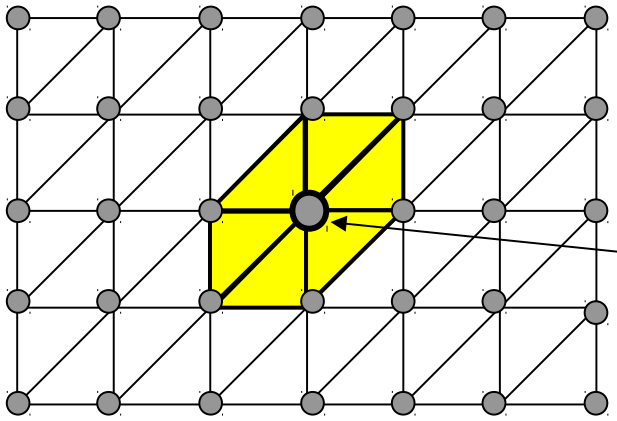
http://www.vterrain.org/LOD/Papers/

Affichage

Quelle que soit la méthode de représentation utilisée pour le terrain, on doit au final afficher un ensemble de triangles.
Plusieurs possibilités, plus ou moins efficaces :

1. Affichage de triangles individuels (`GL_TRIANGLES`)

Simple à utiliser mais inefficace car un même sommet sera transmis plusieurs fois au pipeline OpenGL (et donc transformé géométriquement) → calculs importants et inutiles.



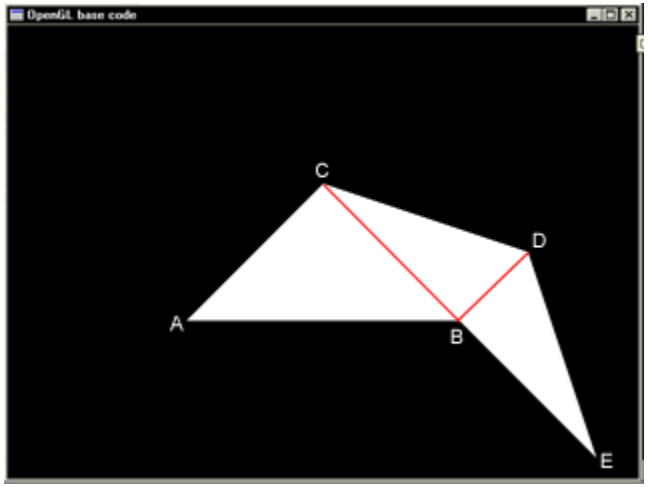
Ce sommet, partagé par 6 triangles, sera transmis 6 fois au pipeline d'OpenGL

2. Triangle strips (`GL_TRIANGLE_STRIP`)

Plus efficace, car on ne transmet qu'une seule fois un sommet donné. Avec n sommets, on définit $(n\text{-}2)$ triangles.

```
glBegin(GL_TRIANGLE_STRIP);
  glVertex3f(-1.0f, -0.5f, -4.0f);    // A
  glVertex3f( 1.0f, -0.5f, -4.0f);    // C

  glVertex3f( 0.0f,  0.5f, -4.0f);    // B
  glVertex3f(1.5f,  0.0f, -4.0f);     // D
  glVertex3f(2.0f, -1.5f, -4.0f);     // E
glEnd();
```



3. Vertex arrays

Très rapide. On fourni 2 listes au pipeline OpenGL :

- la liste des sommets ;
- une liste d'indices dans le tableau de sommets précédent pour les relier sous la forme de triangles.

```
// Définition d'un point du terrain
typedef struct
{
   GLfloat     s, t;                    // Coordonnées de texture
   GLfloat     xn, yn, zn;              // Normale
   GLfloat     x, hauteur, z;           // Coordonnées du sommet
} Point_terrain;

Point_terrain  liste_points[NB_POINTS];
GLuint         liste_indices[NB_INDICES];
```

On peut avoir des vertex arrays définissant des triangles individuels, des triangle strips, etc :

1) vertex arrays définissant des **triangles individuels** :

```
// Affichage
glInterleavedArrays( GL_T2F_N3F_V3F, 0, liste_points );

glDrawElements( GL_TRIANGLES, NB_INDICES,
                GL_UNSIGNED_INT, liste_indices );
```

Remarque : la constante **`GL_T2F_N3F_V3F`** précise l'ordre et la nature des données stockées dans le tableau **`liste_points`**, dans l'ordre :

- **`T2F`** : Coordonnées de texture (2 float)
- **`N3F`** : cooronnées de normale (3 float)
- **`V3F`** : coordonnées de sommet (3 float)

Il existe d'autres constantes, voir la doc d'OpenGL :
`GL_V3F`, **`GL_C3F_V3F`**, **`GL_T2F_C3F_V3F`**, etc.

2) vertex arrays définissant des **triangles strips** :

```
// Affichage
glInterleavedArrays( GL_T2F_N3F_V3F, 0, liste_points );

glDrawElements( GL_TRIANGLE_STRIP, NB_INDICES,
                GL_UNSIGNED_INT, liste_indices );
```