



**P1RV**

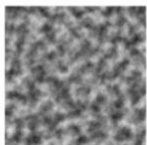
# Génération de terrain 3D à partir de heightmap

# L'idée :

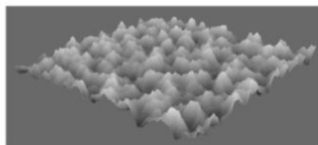
## Sujet 2 : Générateur de terrains 3D basé « heightmap »

- Implémenter en C++ OpenGL/OSG un générateur de terrain 3D basé sur l'utilisation de cartes de hauteur (*heightmaps*)

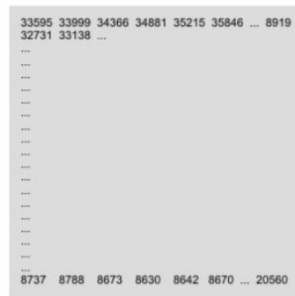
- Voir par exemple : <https://www.3d-map-generator.com/3d-map-generator-terrain/>



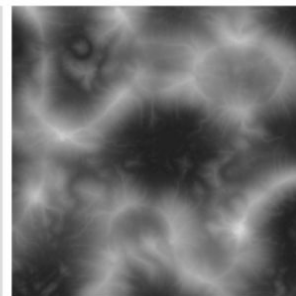
HeightMap



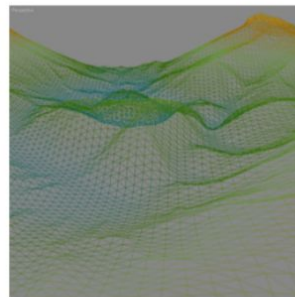
Terrain 3D



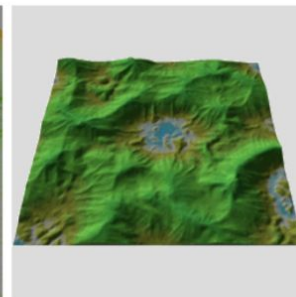
Heightmap array of altitude values



Heightmap equivalent grayscale image



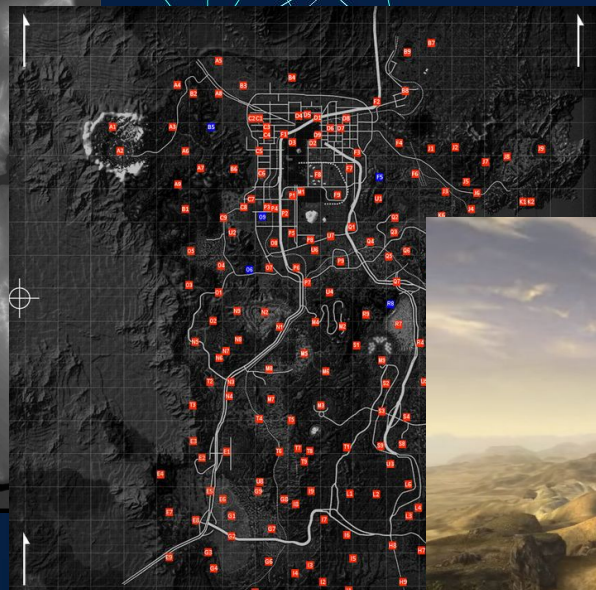
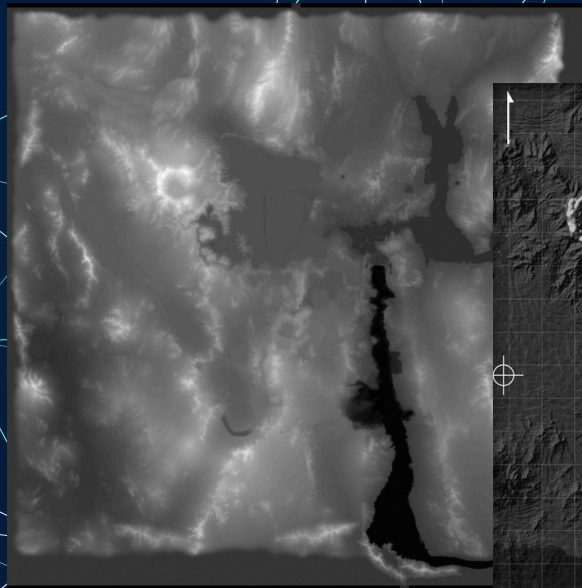
Heightmap equivalent 3D mesh



3D Mesh with colors-by-altitude

Encadrement : J.-M. Normand

# Pourquoi ce sujet ?



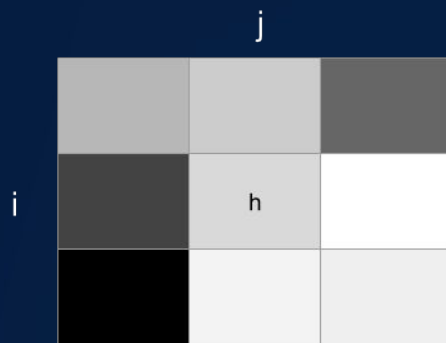
Fallout : New Vegas

# Le cahier des charges

Fonctionnalités principales	Fonctionnalités secondaires
<ul style="list-style-type: none"><li>• Lire</li><li>• Afficher</li><li>• Echelle de couleurs</li><li>• Vue</li></ul>	<ul style="list-style-type: none"><li>• Se déplacer</li><li>• Modifier</li><li>• Plusieurs échelles de couleurs</li></ul>



# L'idée de la génération du terrain en 3D

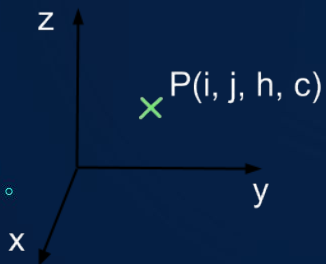


Récupération des coordonnées

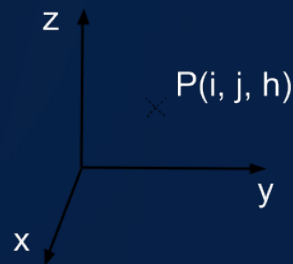


$(i, j, h)$

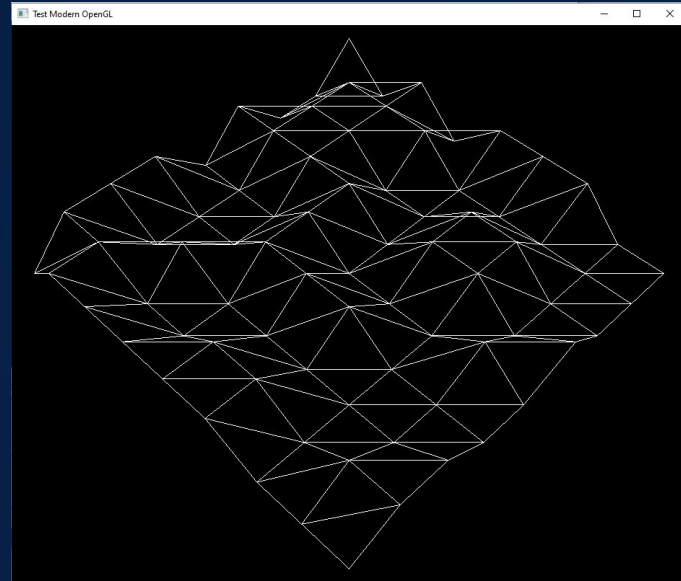
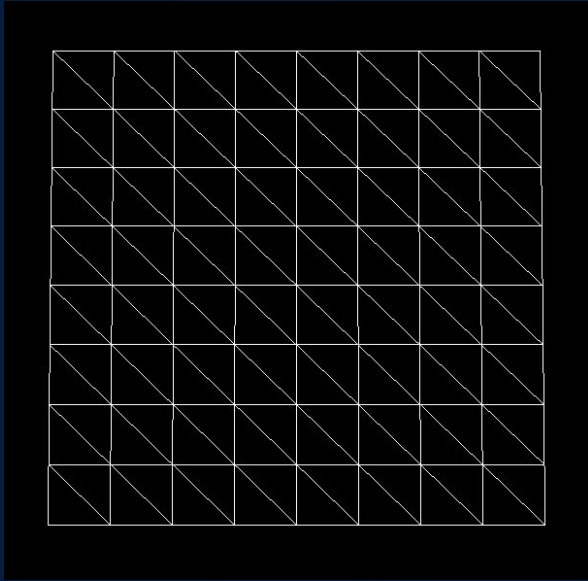
Passage en 3D



Passage en 3D

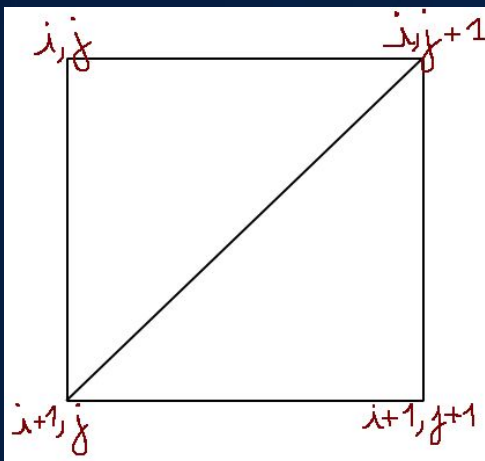


# Un premier algorithme avec OpenGL “ancien”





# Un premier algorithme avec OpenGL "ancien"




```
for (int i = 0; i < img.rows - 1; i ++)  
{  
    for (int j = 0; j < img.cols - 1; j ++)  
    {  
  
        //on dessine le triangle du haut sur une ligne  
        glBegin(GL_TRIANGLES);  
  
        height = (int)img.at<uchar>(i, j);  
        glVertex3f(i, j, height);    //Premier vertex du triangle  
        height = (int)img.at<uchar>(i, j + 1);  
        glVertex3f(i, j + 1, height); //Deuxieme vertex du triangle  
        height = (int)img.at<uchar>(i + 1, j);  
        glVertex3f(i + 1, j, height); //Troisieme vertex du triangle  
  
        glEnd();  
  
        //on dessine le triangle du bas sur une ligne  
        glBegin(GL_TRIANGLES);  
  
        height = (int)img.at<uchar>(i + 1, j);  
        glVertex3f(i + 1, j, height);    //Premier vertex du triangle  
        height = (int)img.at<uchar>(i, j + 1);  
        glVertex3f(i, j + 1, height);    //Deuxieme vertex du triangle  
        height = (int)img.at<uchar>(i + 1, j + 1);  
        glVertex3f(i + 1, j + 1, height); //Troisieme vertex du triangle  
        glEnd();  
    }  
}
```





# Mais des limitations en performances car :

- Redéfinition des vertices et envoi au GPU à chaque tour de boucle
  - volume des données : image en 1080x1080 donne plus de 2 millions de triangles !
  - De la redondance dans la définition des vertices
- 



# OpenGL moderne

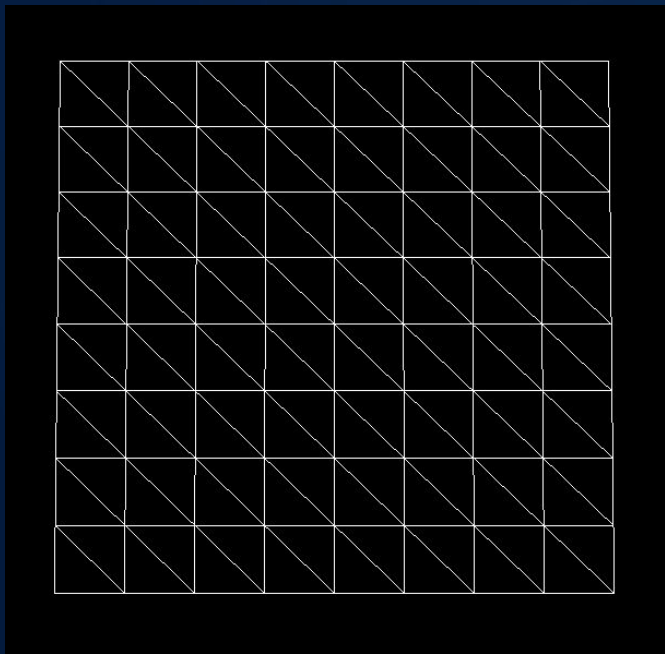
VBO et VBA :

Stockage des données des vertex directement dans des buffers sur le GPU

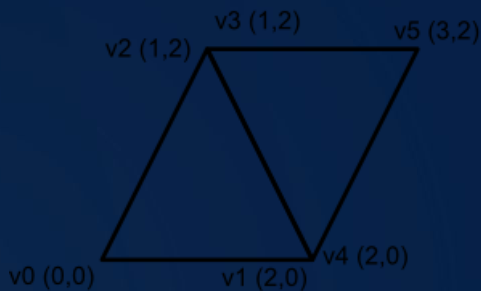
```
//buffer pour les vertex
GLuint vertex_vbo;
//génère le buffer et recupère un indice pour le designer
glGenBuffers(1, &vertex_vbo);
//on le bind pour dire que c'est ce buffer qu'on utilise à présent
glBindBuffer(GL_ARRAY_BUFFER, vertex_vbo);
//on le remplit avec notre tableau de vertex en lui fournissant un pointeur sur la premiere donnée
glBufferData(GL_ARRAY_BUFFER, tableauVertex.size() * sizeof(float), &tableauVertex[0], GL_STATIC_DRAW);
```

# OpenGL Moderne

EBO : évite la redondance de vertices !

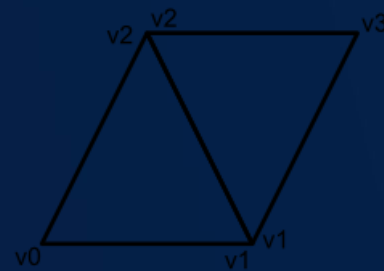


Without indexing



[0,0, 2,0, 1,2, 1,2, 2,0, 3,2]

With indexing

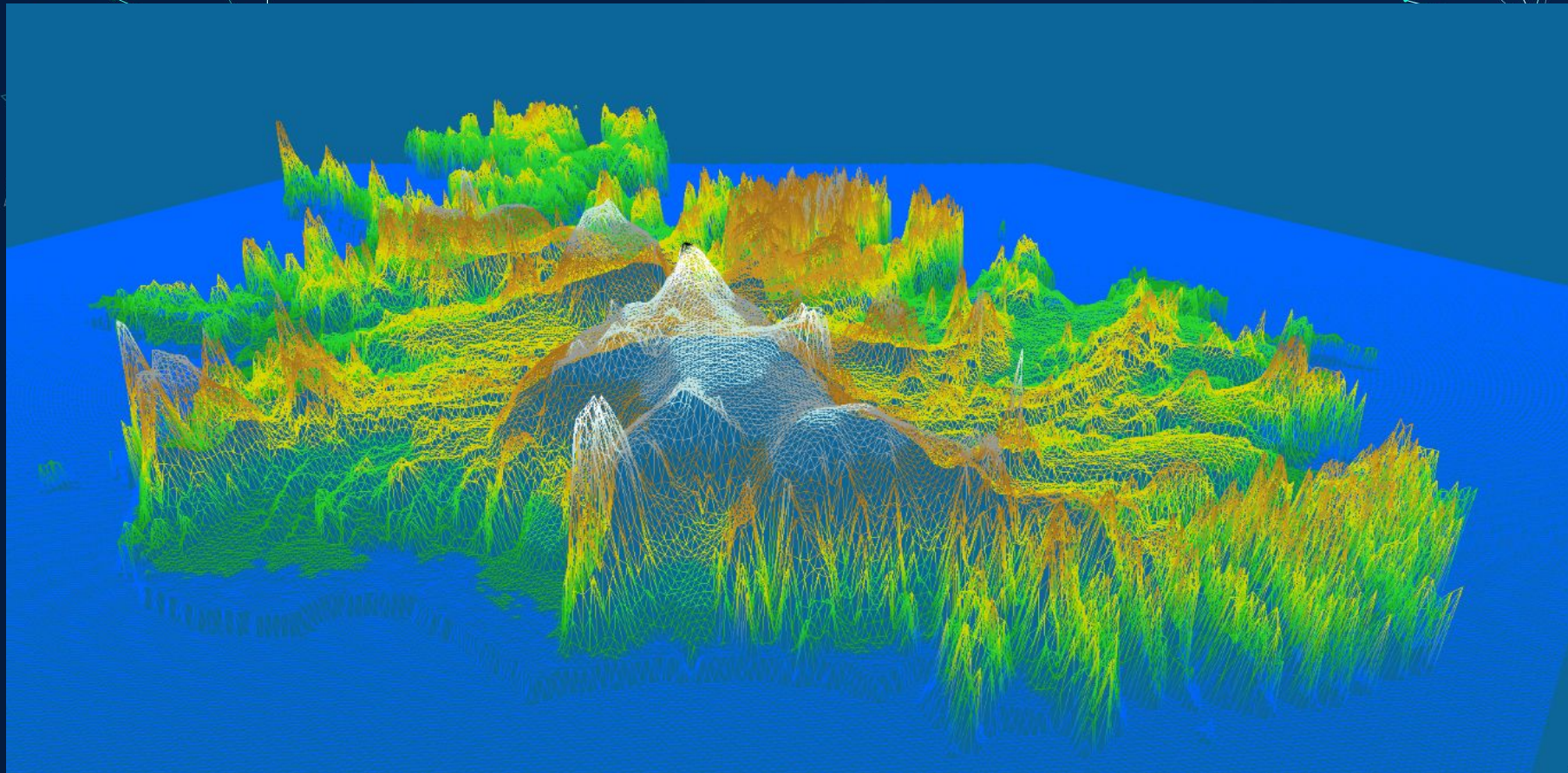


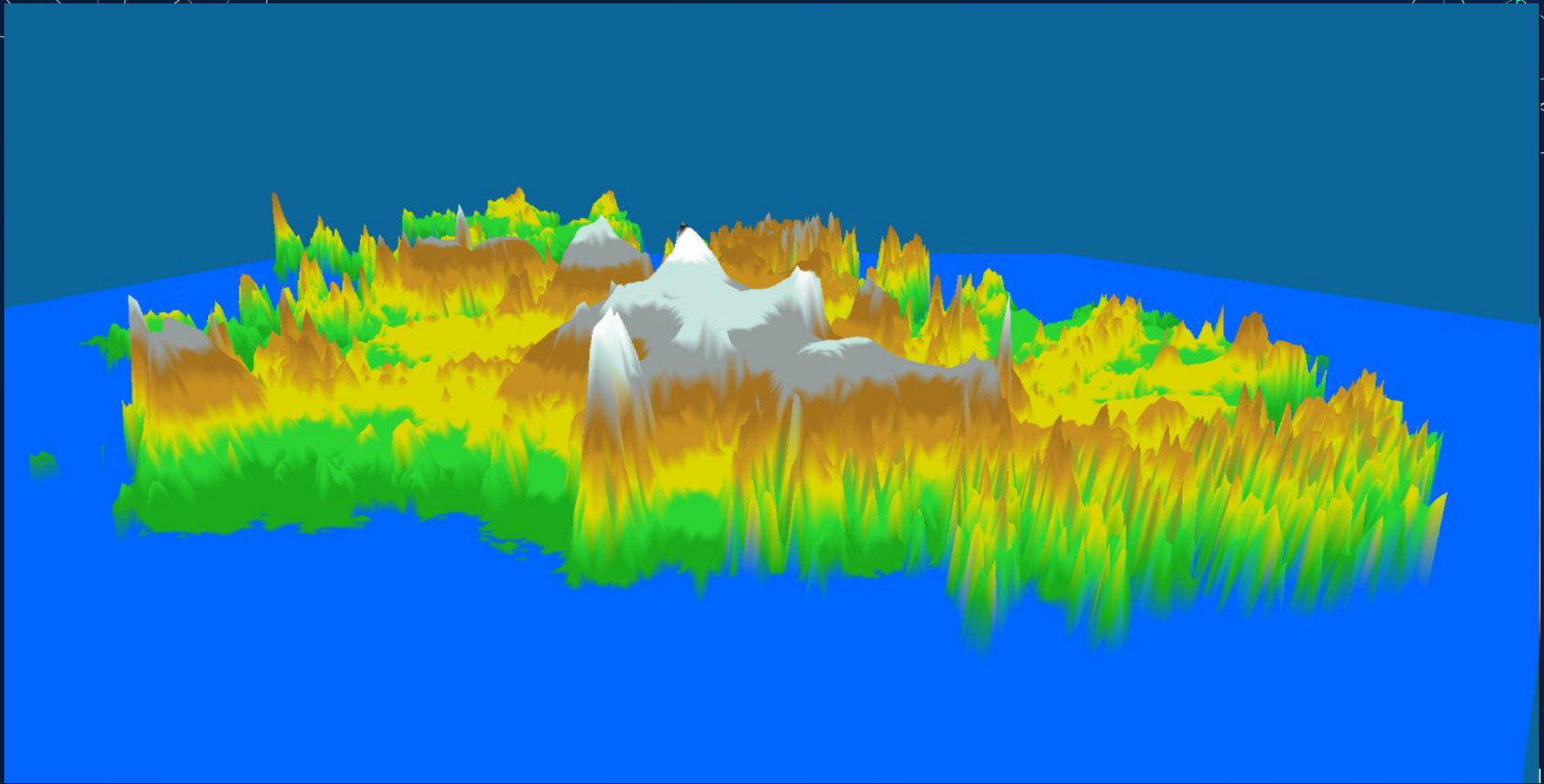
[0,1,2, 2,1,3]  
[0,0, 2,0, 1,2, 3,2]

Vertices  
reused  
twice

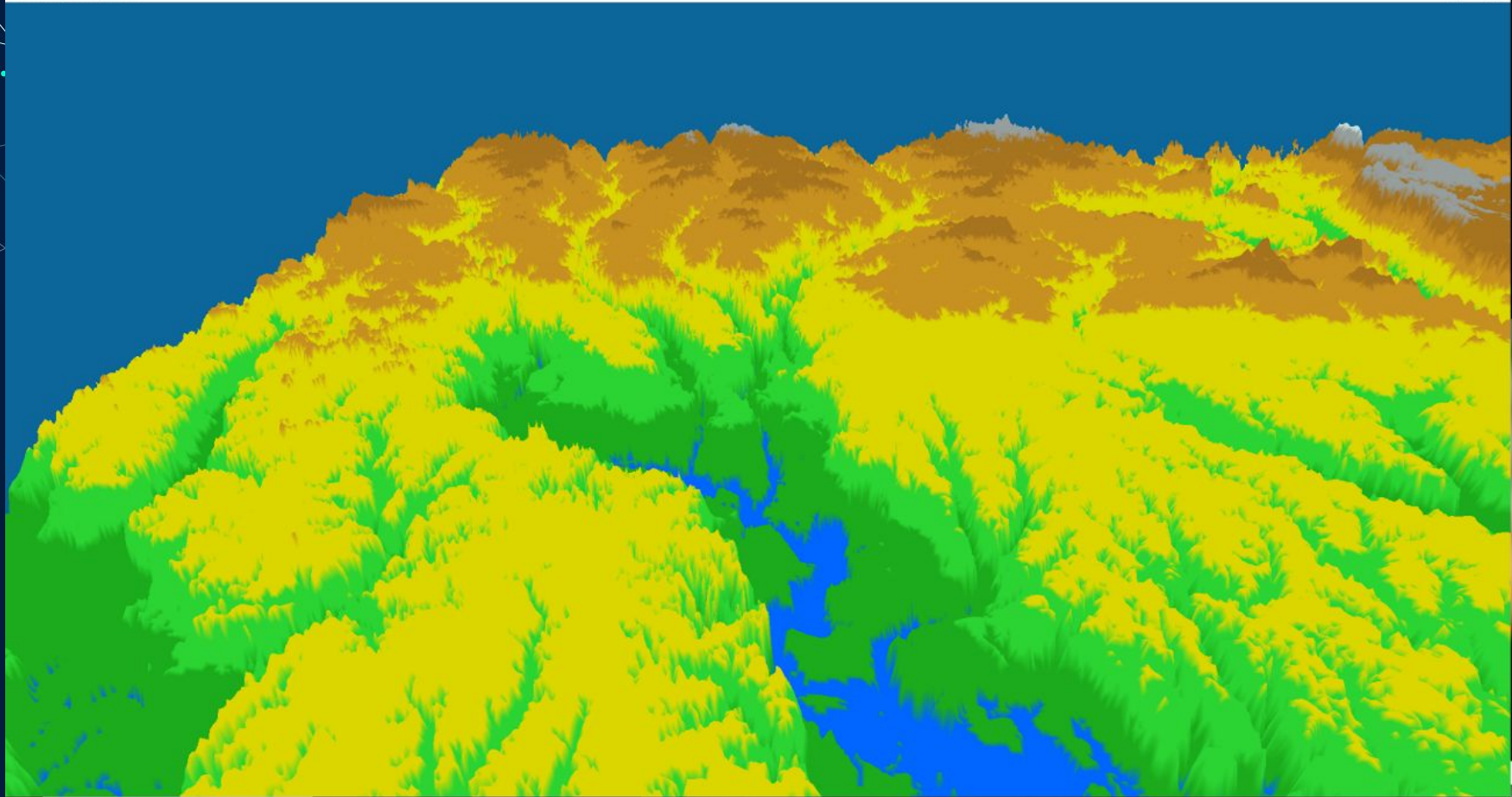


# Résultats :











# Edition de heightmap : OpenCV

Ce que l'on voudrait faire :

- pinceau à la souris façon Paint, avec choix de la teinte, de la taille
- possibilité d'appliquer un flou par endroit, pour adoucir le relief

Les outils d'OpenCV :

- callback souris 
- trackbars (sliders) 
- tracé de formes géométriques (pleines ou contours) 



# Redéfinition des pinceaux

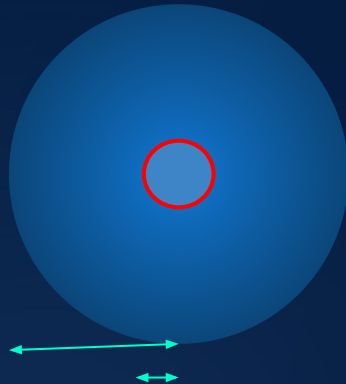
Une première classe : CircleBrush

Attributs :

- int m\_color
- int m\_radius\_int
- int m\_radius\_ext

Une méthode pour peindre :

- void paint(int x, int y, cv::Mat& image) const



# Redéfinition des pinceaux

Une première classe : CircleBrush

Attributs :

- int m\_color
- int m\_radius\_int
- int m\_radius\_ext

Une méthode pour peindre :

- void paint(int x, int y, cv::Mat& image) const



# Redéfinition des pinceaux

Une autre classe : BlurBrush

Attribut:

- int m\_radius

Une méthode pour peindre :

- void BlurPaint(int x, int y, cv::Mat& image) const



## Redéfinition des pinceaux

Une autre classe : BlurBrush

Attribut:

- int m\_radius

Une méthode pour peindre :

- void BlurPaint(int x, int y, cv::Mat& image) const

# Limites et amélioration

Le rendu 3D manque encore de réalisme -> Shader ? Autre échelle de couleurs, texture ?  
Ombres ? Manque de temps mais intéressant à approfondir !

Le calcul des zones de flou est lent dès que la taille du pinceau augmente -> Utiliser des ROI ?  
Pour aller plus loin : plusieurs niveaux de flou



**Bilan**

**OpenGL  
“ancien”**

**OpenCV**

**Optimisation**

**GLFW**

**OpenGL  
“moderne”**

