Compte rendu : TP7 – MOTEUR DE JEUX

Vincent de Rousiers

Résultats:

Image de base représentée par des points

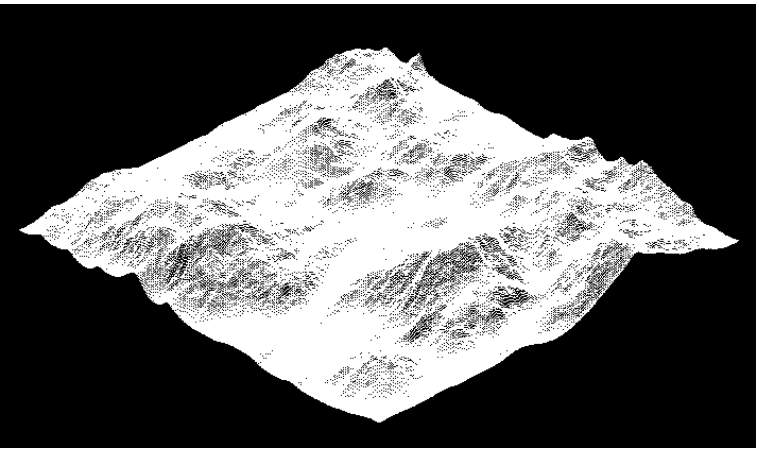


Image de base représentée par des lignes

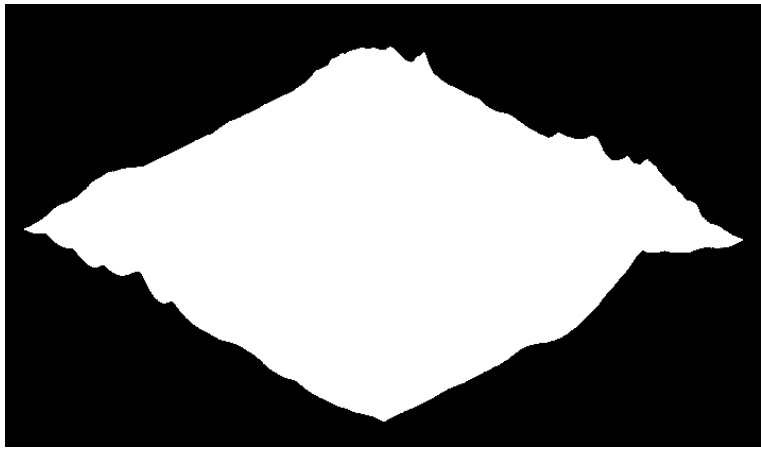
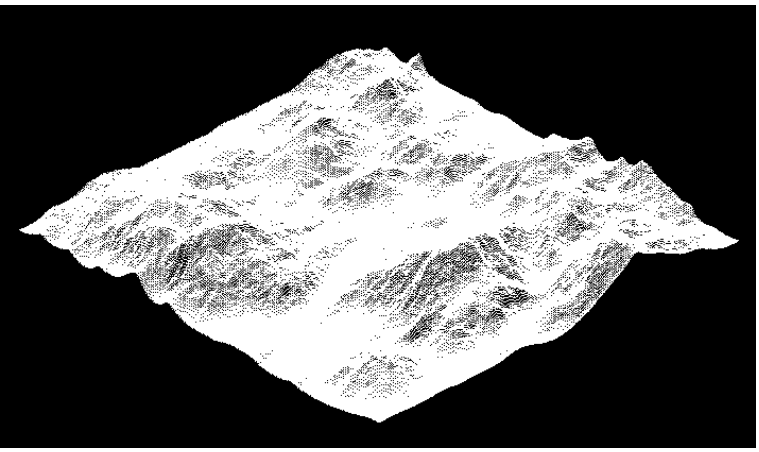
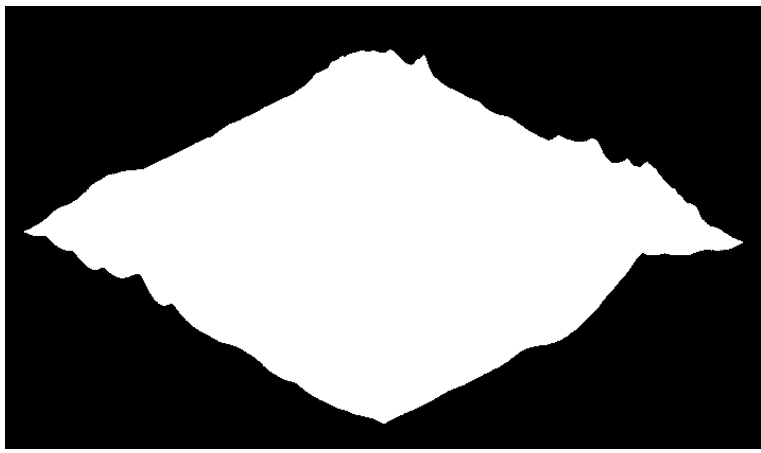
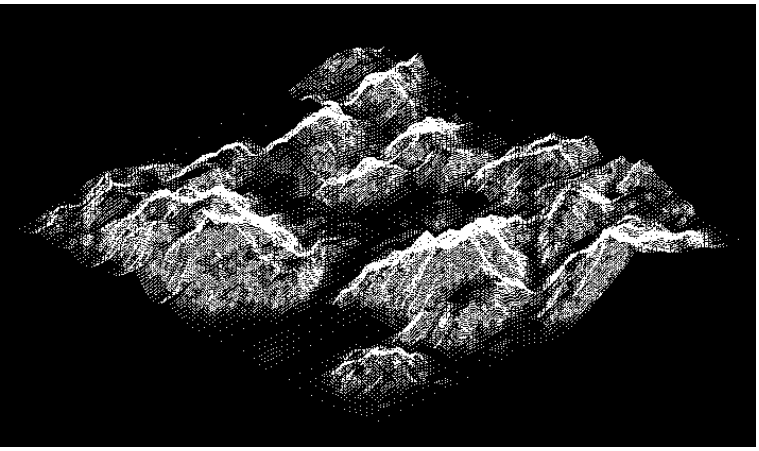


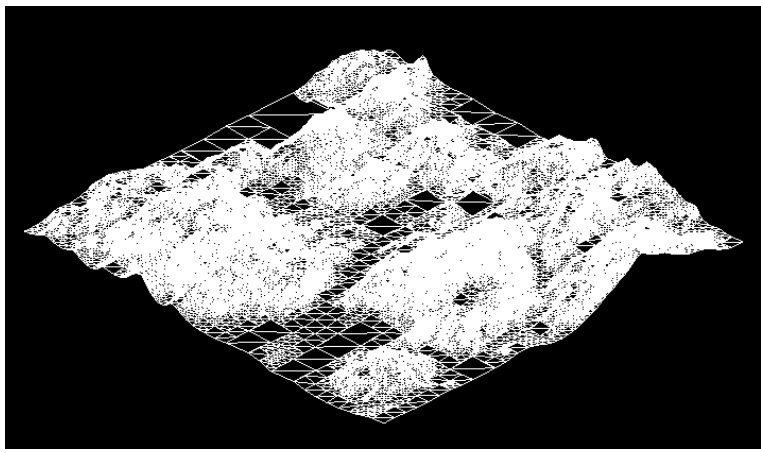
Image obtenue via « QuadTree » avec une tolérance nulle (0,0)



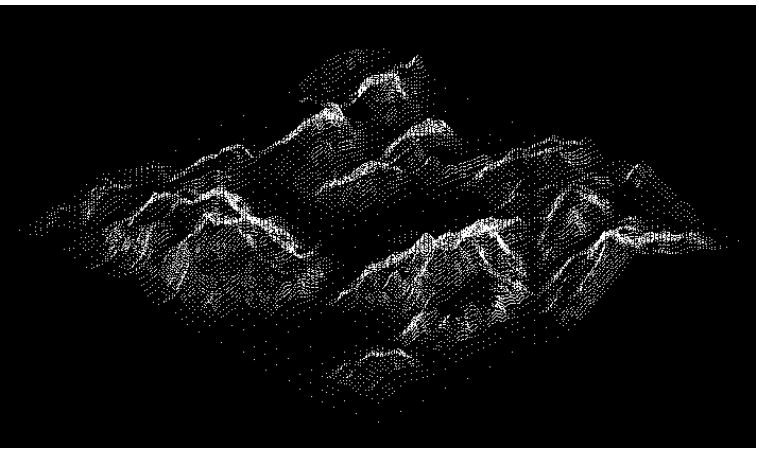


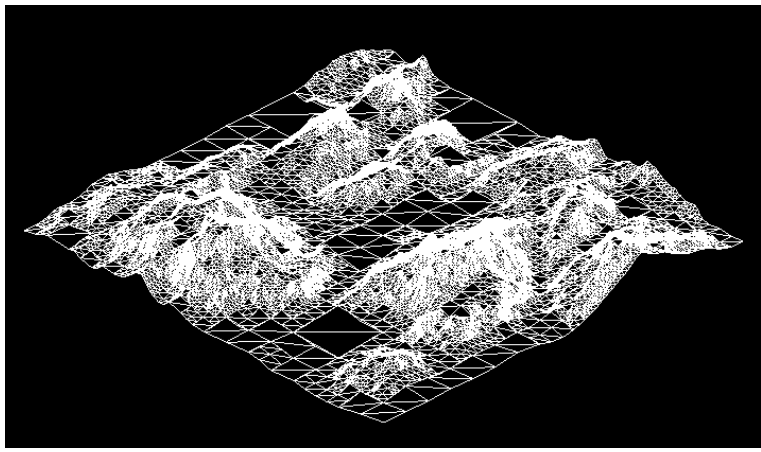
avec une tolérance assez faible (10,0)



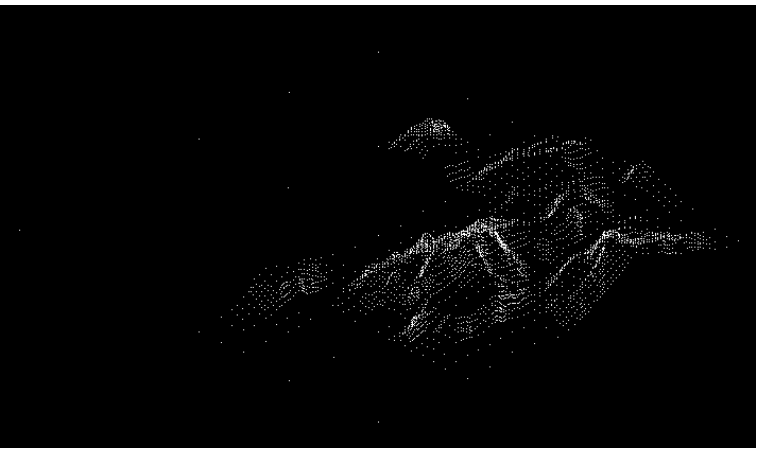


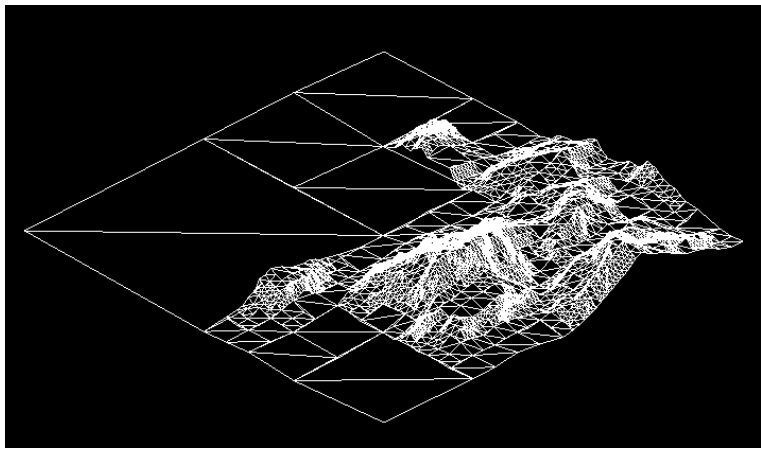
avec une tolérance faible (20,0)



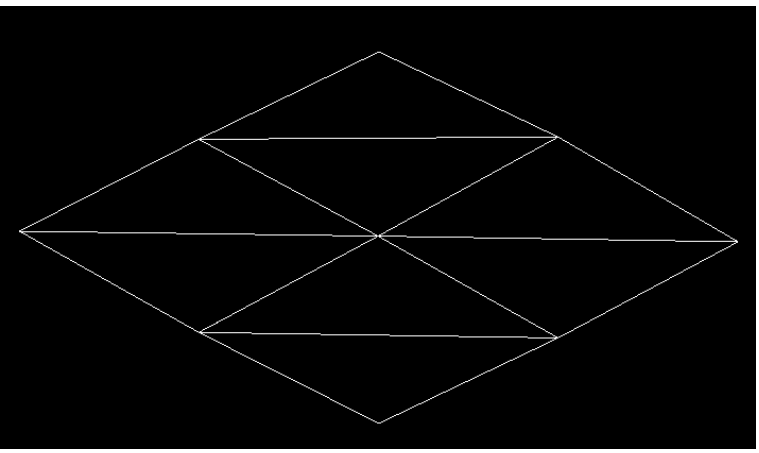


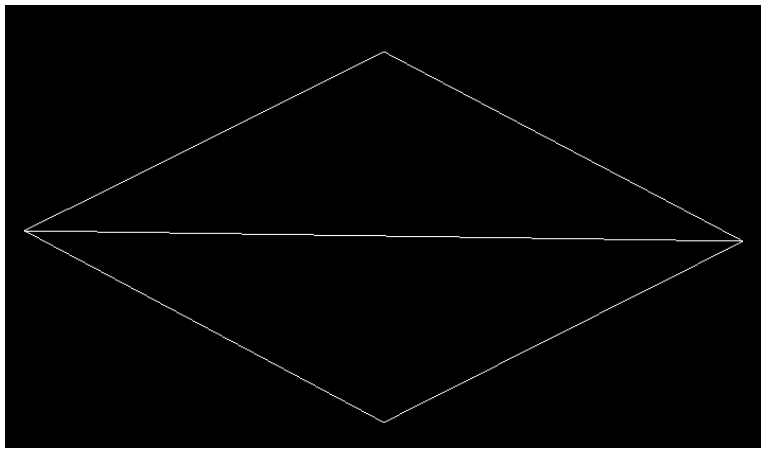
avec une tolérance moyenne (30,0)





avec une tolérance haute (40,0) et très haute (50,0)





Le gain de performance est bien entendu très important en augmentant la tolérance de l'algorithme et il n'y a pas de perte de points ou de faces avec une tolérance nulle.

QuadTree :

La méthode de QuadTree utilisée pour réaliser ce TP est la suivante :

- On démarre l'algorithme avec en paramètre les limites en x et y du quad actuel (donc au début les limites de l'image de base)

```
void GameWindow::quadTree(QImage m_image, float tolerance, int widthMin, int
widthMax, int heightMin, int heightMax)
```

- On utilise le tableau de points déjà créé lors d'une première lecture de l'image et qui permet de ne pas retraiter la lecture afin de récupérer la position des points qui nous intéressent via leurs indices

```
int i = widthMin;
int j = heightMin;
int id1 = i*m_image.width() +j;

i = widthMax;
j = heightMin;
int id2 = i*m_image.width() +j;

i = widthMin;
j = heightMax;
int id3 = i*m_image.width() +j;

i = widthMax;
j = heightMax;
int id4 = i*m_image.width() +j;
```

- On vérifie ensuite que les points ne seront pas identiques lors de la prochaine itération (que les limites ne soient pas égales)

```
if(widthMin < widthMax -1 && heightMin < heightMax -1)
```

  - Si c'est le cas, on est arrivé à une « feuille », on crée donc les relations qui permettront de créer le QuadTree. `relationsQT` étant un tableau d'int qui « noue » les points via leurs indices dans le tableau de points. Ce nouveau tableau associe donc à un point d'indice id1 les deux points qui vont former avec lui une face (id2 et id3). Pareil pour id4.

```
relationsQT[id1][cptQT[id1]++] = id2;
relationsQT[id1][cptQT[id1]++] = id3;
relationsQT[id1][cptQT[id1]++] = id4;
relationsQT[id4][cptQT[id4]++] = id2;
relationsQT[id4][cptQT[id4]++] = id3;
```

  - Sinon on teste la tolérance. On regarde si la différence entre le point le plus haut et le point le plus bas du quad actuel est bien inférieure à la tolérance.

```
float max1 = max(p[id1].z, p[id2].z);
float max2 = max(max1, p[id3].z);
float maxF = max(max2, p[id4].z);

float min1 = min(p[id1].z, p[id2].z);
float min2 = min(min1, p[id3].z);
float minF = min(min2, p[id4].z);

if(maxF-minF < 0.001f*tolerance)
```

    - Si la tolérance n'est pas respectée alors on arrive là aussi à une branche et on peut nouer les points à nouveau

- Sinon, on peut continuer de descendre dans l'arbre

```
quadTree(m_image, tolerance, widthMin, (widthMax+widthMin)/2, heightMin, (heightMax+heightMin)/2);
quadTree(m_image, tolerance, (widthMax+widthMin)/2, widthMax, heightMin, (heightMax+heightMin)/2);
quadTree(m_image, tolerance, widthMin, (widthMax+widthMin)/2, (heightMax+heightMin)/2, heightMax);
quadTree(m_image, tolerance, (widthMax+widthMin)/2, widthMax, (heightMax+heightMin)/2, heightMax);
```