Projet 3D

Rush 1

Le *convertisseur* de terrains

But : convertir un terrain de type «heightmap» en fichier utilisable par nos applications 3D

```
int main (...)
{
        LireFichierHeightmap();
        ConstruireTerrain(float echelleXY, float echelleZ    );
        CalculerNormales();
        ConstruireIndex();
        EnregistrerTout();

        return 0;
}
```

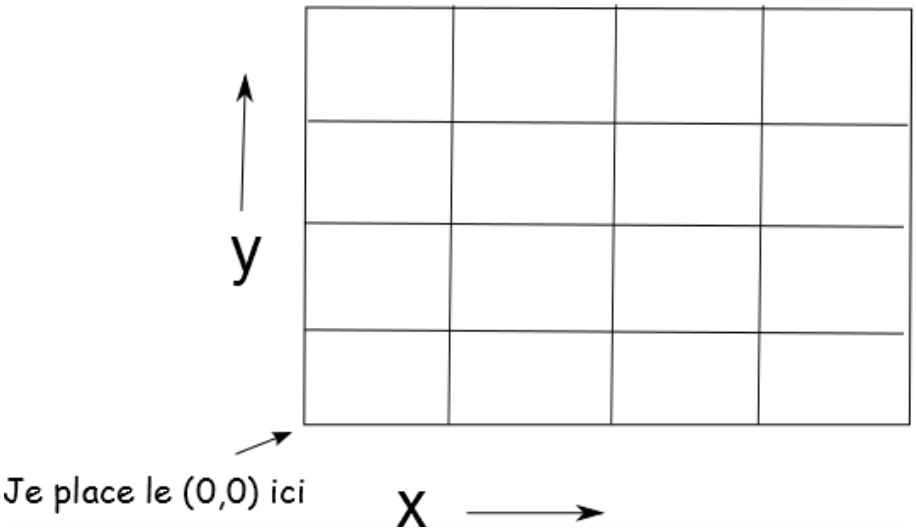
PLUS: un petit programme permettant de lire le fichier créé pour tester la classe et/ou fonction de lecture

La «patch» de terrain

Éléments de solution - La classe CTerrain

1. Comment définir une «patch»

Prenons par exemple un terrain de 5 sommets X 5 sommets



Les sommets peuvent être définis horizontalement ou verticalement, à vous de choisir mais rappelez vous-en!

Combien de polygones aurons nous ?

2. Comment définir les sommets?

a. VARIABLES UTILES : dx, dy ➔ dimension en X et dimension en Y
sommets ➔ un tableau de (dx X dy) sommets
nbSommets
nbPolygones
CSommetTerrain ➔ une classe pour les sommets

b. Pour le moment, définir une normale simple, par exemple (0, 0, 1)

c. Créer dynamiquement le tableau. **Pourquoi?**

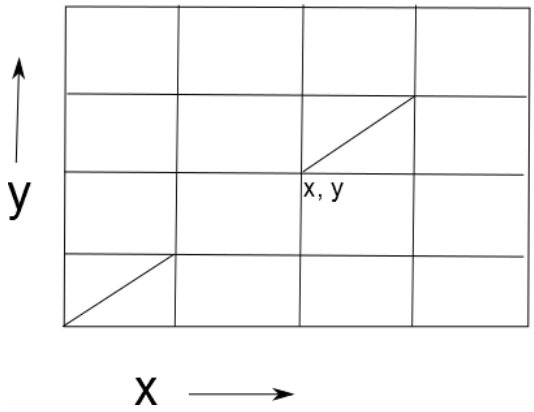
d. Si possible, définir le tableau à une seule dimension (M X M). Et utiliser x et y pour calculer la position (l'indice).

e. Pour la classe de sommets, utiliser des **floats**. **Pourquoi?**
S'assurer que le format est compatible avec ce que nous ferons plus tard dans notre application DirectX. **Pourquoi et comment?**

f. Sauvegarder les sommets et **quoi d'autre?**

g. (création des index et sauvegarde des index – voir en 3.)

3. Comment définir les index?



Prenons, par exemple les polygones au dessus de x,y

Voici MA FONCTION Init+Index

```
void CTerrain::InitIndex ()
{

pIndices = new unsigned int[nombrePolygones*3]; // déclarer unsigned int* pIndices; dans
la classe CTerrain

          int k = 0;

          for ( int y = 0 ; y < M-1 ; ++y )
          {
               for ( int x = 0 ; x < M-1 ; ++x )
               {
                      // L'important ici est d'utiliser la même formule pour identifier
                      // les sommets qu'au moment de leur création
                      pIndices[k++] = y*M + x ;
                      pIndices[k++] = (y+1) * M  + (x+1);
                      pIndices[k++] = y*M  + (x+1);
                      pIndices[k++] = y*M  + x;
                      pIndices[k++] = (y+1)* M  + x;
                      pIndices[k++] = (y+1)* M  + (x+1);

               }

          }

          (sauvegarder…)

}
```

NOTEZ QU'IL N'Y A PAS D'ALGORITHME FACILE POUR DÉFINIR NOS POLYGONES

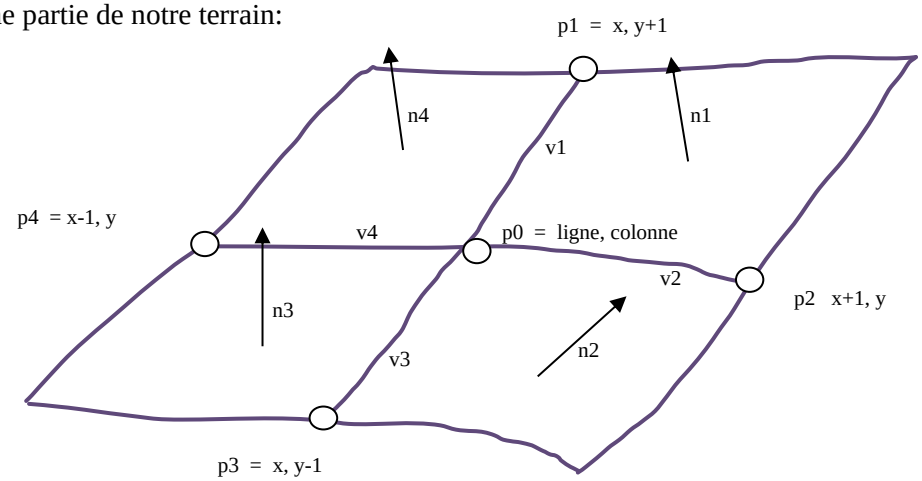
LE "TOURNE" MES POLYGONES DANS LE SENS HORAIRE (CLOCKWISE)

LES FACES TOURNANT EN SENS ANTI-HORAIRE SERONT RETIRÉES ([redacted]_CCW)

Le *convertisseur* de terrains

Le calcul des normales

Soit une partie de notre terrain:



1. On se rappelle ☺ que le **produit vectoriel** (*cross product*) de deux vecteurs nous donne comme résultat un vecteur perpendiculaire au plan contenant les deux vecteurs. On pourrait le normaliser (le rendre de longueur 1) pour avoir réellement la normale du plan mais pour nos besoins, ce n'est pas nécessaire.
2. On calcule donc v1 à v4, par exemple v1 = p1 – p0 (si p1 existe...)
3. On calcule ensuite n1 à n4 avec le produit vectoriel.
4. On additionne v1 à v4, on normalise le résultat et voilà !

Le code (à titre d'exemple) Attention, je suis en main droite, le z en haut.

```
XMFLOAT3 CalculNormale(int x, int y)
{
    XMVECTOR n1;
    XMVECTOR n2;
    XMVECTOR n3;
    XMVECTOR n4;

    XMVECTOR v1;
    XMVECTOR v2;
    XMVECTOR v3;
    XMVECTOR v4;

    n1 = n2 = n3 = n4 = XMVectorSet(0,0,1,0);  // Le Z est le haut

    // v1 = p1 – p0, etc...
    if (y < M-1)  v1 = ObtenirPosition( x, y+1 ) - ObtenirPosition( x, y );
    if (x < M-1)  v2 = ObtenirPosition( x+1, y ) - ObtenirPosition( x, y);
    if (y > 0)    v3 = ObtenirPosition( x, y-1 ) - ObtenirPosition( x, y);
    if (x > 0)v4 = ObtenirPosition(x-1, y ) - ObtenirPosition( x, y);

    // les produits vectoriels
    if (y < M-1 && x < M-1 ) n1 = XMVector3Cross(v2, v1);
    if (y > 0   && x < M-1 ) n2 = XMVector3Cross(v3, v2);
    if (y > 0   && x > 0   ) n3 = XMVector3Cross(v4, v3);
    if (y < M-1 && x > 0   ) n4 = XMVector3Cross(v1, v4);

    n1 = n1 + n2 + n3 +n4;

    n1 = XMVector3Normalize(n1);
    XMFLOAT3 resultat;

    XMStoreFloat3(&resultat, n1);

    return resultat;
}
```

Autres détails

1. J'ai utilisé DirectXMATH comme bibliothèque mathématique

2. Si vous ne voulez pas (ou ne pouvez pas) utiliser DirectXMATH, il vous faut une fonction de produit vectoriel et une fonction de normalisation du style

```
montypedevecteur ProduitVectoriel(  montypedevecteur  v1, montypedevecteur v2 )
{
   montypedevecteur resultat;

   resultat.x = v1.y * v2.z – v1.z – v2.y;
   resultat.y = v1.z * v2.x – v1.x – v2.z;
   resultat.z = v1.x * v2.y – v1.y – v2.x;

   return resultat;
}

montypedevecteur Normalise( montypedevecteur v )
{
   return v / longueur(v);
}

float Longueur(montypedevecteur v)
{
   return sqrt( v.x * v.x + v.y * v.y + v.z * v.z );
}
```