# Le Splatting en Francais

Maxime COSTE

December 6, 2004

## 1 Introduction

Le splatting est une méthode de texturage de terrain qui permet un texturage de très haute résolution. Le principe est relativement simple, on affiche le terrain en plusieurs passes, une pour chaque textures (par exemple une texture pour le sable, une pour l'herbe, une pour la roche...) et à chaque passage, on utilise une texture alpha pour déterminer les zones où la texture doit etre affichée. Ainsi on peut avoir du texturage haute résolution sans avoir une texture $65536 * 65536 * 4$ à plaquer sur notre terrain. L'utilisation de l'apha map permet en outre d'effectuer des transitions entre les types de terrains.

### 1.1 Assertions

Voici les assertions que je fais dans cet article :

- Votre terrain est basé sur un heightmap ou une méthode similaire;
- Votre terrain et divisé en "Tiles" carrées que vous pouvez afficher indépendamment les unes des autres.

### 1.2 Attention jeunes gens

Je présente ici ma version du splatting basique, qui est légèrement différente de la version présenté par Charles Bloom ("l'inventeur du splatting").

## 2 Le prétraitement

Avant de commencer à afficher votre terrain, il faut faire quelques petits calculs.

## 2.1 Les coordonnées de texture

Pour ma version du splatting, nous avons besoin de deux groupes de coordonnées de textures, un groupe qui étire toute la texture sur le terrain, et un autre qui répète la texture tous les $n$ vertex.

Pour ce faire, voici ma méthode : Taille est le nombre de vertex en largeur (ou hauteur), TexCoords1 sont les coordonnées de texture répétées et TexCoords2 celles de texture étirées.

```
TexCoords1 = new float[Taille*Taille*2];
TexCoords2 = new float[Taille*Taille*2];
for (unsigned x=0; x < Taille; x++)
{
    for (unsigned z = 0; z < Taille; z++)
    {
        // ici par exemple, on etire la texture
        // sur des carres de 4*4 vertex
        TexCoords1[(x*Taille+z)*2]      = x/4.f;
        TexCoords1[(x*Taille+z)*2 + 1]  = z/4.f;

        TexCoords2[(x*Taille+z)*2]      = x/(float)(Taille);
        TexCoords2[(x*Taille+z)*2 + 1]  = z/(float)(Taille);
    }
}
```

## 2.2 Déterminer les textures

Il faut aussi déterminer quel type de terrain attribuer à chaque vertex. En général le type de terrain est détermié à partir de l'altitude et de la pente. Pour ceci, je vous laisse le soin de tester ces différents paramètres (pour commencer je conseille tout de même de n'utiliser que l'altitude pour avoir une fonction la plus simple possible car il vaut mieux éviter d'avoir plusieurs sources d'erreurs en même temps).

## 2.3 Générer les Alpha Map

Pour ce faire, il y a plusieurs approches : une alpha map par texture et par tile, ou une alpha map par texture tout court. La première approche demande de créer plein de petites textures tandis que la deuxième demande autant de grosses textures que de types de terrain. J'utilise pour ma part les grosses textures (pour un terrain de 256*256 vertex, je fais une alpha map de 256*256 pixels, soit un pixel par vertex...)

Note : *pour l'autre méthode, il faut un autre groupe de coordonnées de textures qui plaque une texture par tile.*

Pour générer l'alpha map, j'utilise l'algorithme suivant

```
unsigned char* AlphaMap = new unsigned char[Taille*Taille*4];
memset (AlphaMap, 0, Taille*Taille*4);
```

```
// on va faire la moyenne des vertex a
// moins de n vertex du vertex considere
unsigned n = 3

// nTexs est le nombre de textures (nombre de types de terrain)
for (unsigned i = 0; i < nTexs; i++)
{
  for (unsigned x = 0; x < Taille; x++)
  {
    for (unsigned z = 0; z < Taille ; z++)
    {
      AlphaMap[(x+Taille*z)*4+3] = 0;

      for (int j = -n+1; j < n; j++)
      {
        for (int k = -n+1; k < n; k++)
        {
          // si la texture du vertex est celle en cours,
          // on incremente, les CLAMP servent a ne pas dépasser
          AlphaMap[(x+Taille*z)*4+3] += TexMap[CLAMP((x+j),0,Taille)*
                  Taille+CLAMP((z+k),0,Taille)] == i ? 1 : 0;
        }
      }

      // AlphaMap[(x+Taille*z)*4+3] est compris entre 0 et
      // (2*n-1)*(2*n-1), on le remet entre 0 et 255
      AlphaMap[(x+Taille*z)*4+3] *= 255/((2*n-1)*(2*n-1));
      AlphaMap[(x+Taille*z)*4+0] = 255;
      AlphaMap[(x+Taille*z)*4+1] = 255;
      AlphaMap[(x+Taille*z)*4+2] = 255;
    }
  }
}
```

Ensuite on crée une texture OpenGL et on met l'alpha map dedans.

### 2.4 La texture de base

Enfin, on calcule la texture de base. Il s'agit d'avoir une grosse texture (2048*2048 par exemple) qui sert à afficher le terrain distant. Cette texture est calculée en affichant le terrain (sans texture de base puisqu'on veut la calculer) en plaçant la vue de facon à avoir tout le terrain à l'écran, ensuite on récupère le frame buffer et on met tout ça dans une texture. Encore faut-il savoir afficher le terrain...

## 3 L'affichage

Pour afficher, on affiche en n+1 passes, n étant le nombre de textures.

La première passe consiste à afficher le terrain avec la texture de base. Pourquoi ? Parce que lors du premier passage avec les textures haute

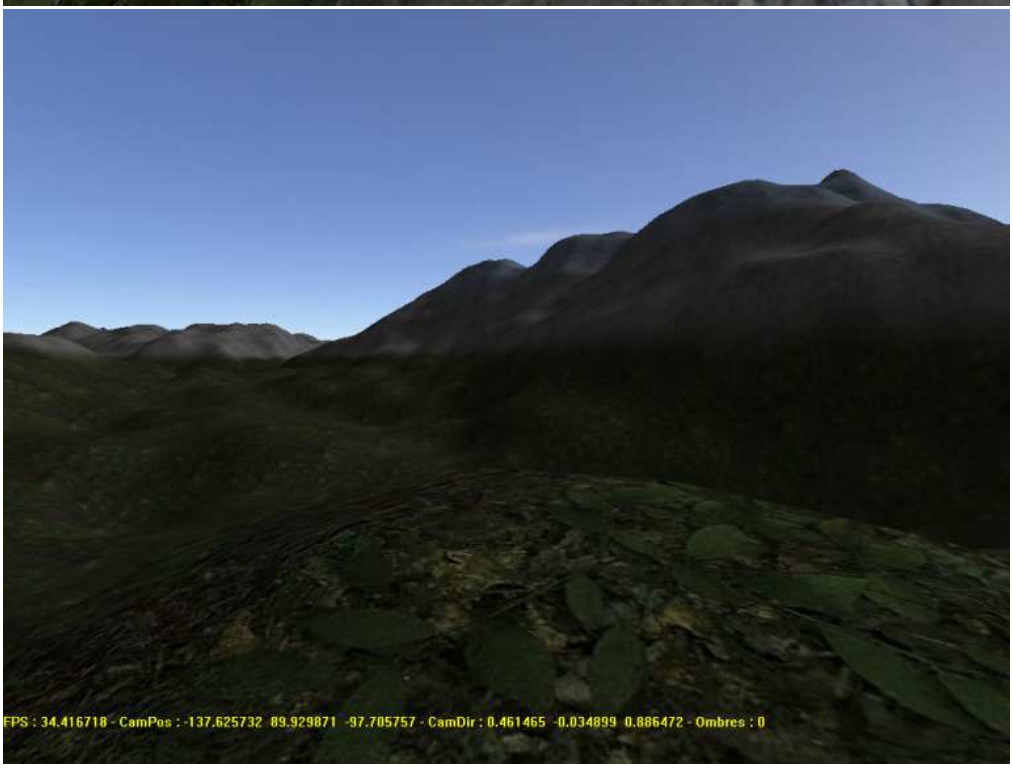
définition, il faut qu'il y ait quelque chose sur lequel "blender", et aussi parce que la plupart du temps, on n'affiche les texture détaillées que pour le terrain proche, et on reste avec la texture de base pour le terrain lointain.

Les autres passes se font en affichant chaque texture dans les parties qui lui sont attribuées via son alpha map.

Note : *on peut améliorer la vitesse en utilisant le multitexturing pour faire deux passes en une mais dans mon cas, avec 4 unitées de textures dont une réservée pour le shadow mapping, je n'avais pas trop le choix.*

```
// premiere passe : on affiche la texture de base
// il faut ici utiliser les coordonnees de texture qui etirent
// la texture sur tout le terrain
glBindTexture(GL_TEXTURE_2D, IdTextureBase);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

for (unsigned c = 0; c < nTiles; c+=1)
    AfficherTile(&(pTiles[c]));

// utilisation de l'extention opengl GL_ARB_texture_env_combine

// on va blender les differents rendus, un avec chaque texture
glEnable(GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE_ARB);
glTexEnvi(GL_TEXTURE_ENV, GL_SOURCE0_RGB_ARB, GL_TEXTURE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB_ARB, GL_REPLACE);

// dans GL_TEXTURE1, on met l'alpha map et on dit a opengl
// d'utiliser l'alpha de GL_TEXTURE1 comme alpha tout court
glActiveTextureARB (GL_TEXTURE1_ARB);
glTexEnvi(GL_TEXTURE_ENV, GL_SOURCE0_RGB_ARB, GL_PREVIOUS);
glTexEnvi(GL_TEXTURE_ENV, GL_SOURCE0_ALPHA_ARB, GL_TEXTURE);
glTexEnvi(GL_TEXTURE_ENV, GL_SOURCE1_ALPHA_ARB, GL_PRIMARY_COLOR_ARB);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB_ARB, GL_REPLACE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_ALPHA_ARB, GL_MODULATE);

glActiveTextureARB (GL_TEXTURE0_ARB);

// utiliser ici les coordonnees de textures qui se repetent le
// long du terrain
for (unsigned i = 0; i < nTexs; i++)
{
    glBindTexture(GL_TEXTURE_2D, pTexsId[i]);

    glActiveTextureARB (GL_TEXTURE1_ARB);
    glBindTexture(GL_TEXTURE_2D, pAlphaMapsId[i]);
    glActiveTextureARB (GL_TEXTURE0_ARB);

    for (unsigned c = 0; c < nTiles; c+=1)
        AfficherTile(&(pTiles[c]));
}
```

Quelques images pour montrer le résultat :

# 4 Informations annexes

Le splatting par Charles Bloom : http://www.cbloom.com/3d/techdocs/splatting.txt
Contactez-moi à cette adresse e-mail : frrrwww-splatting@yahoo.fr
Merci a allergy pour la correction des fautes d'orthographes.