Approximation de systèmes particulaires.

Laurent Cancé Francis
17/11/2017

I. Présentation du problème

Il existe déjà plusieurs méthode de rendu de feu, en se basant sur un système de particules en nombre important pour un rendu réaliste, ou par une méthode de dérivées en se basant sur une discrétisation de l'espace en 2D ou 3D.
Ici, nous discuterons d'une méthode hybride à la fois particulaire, et autant de synthèse différentielle.

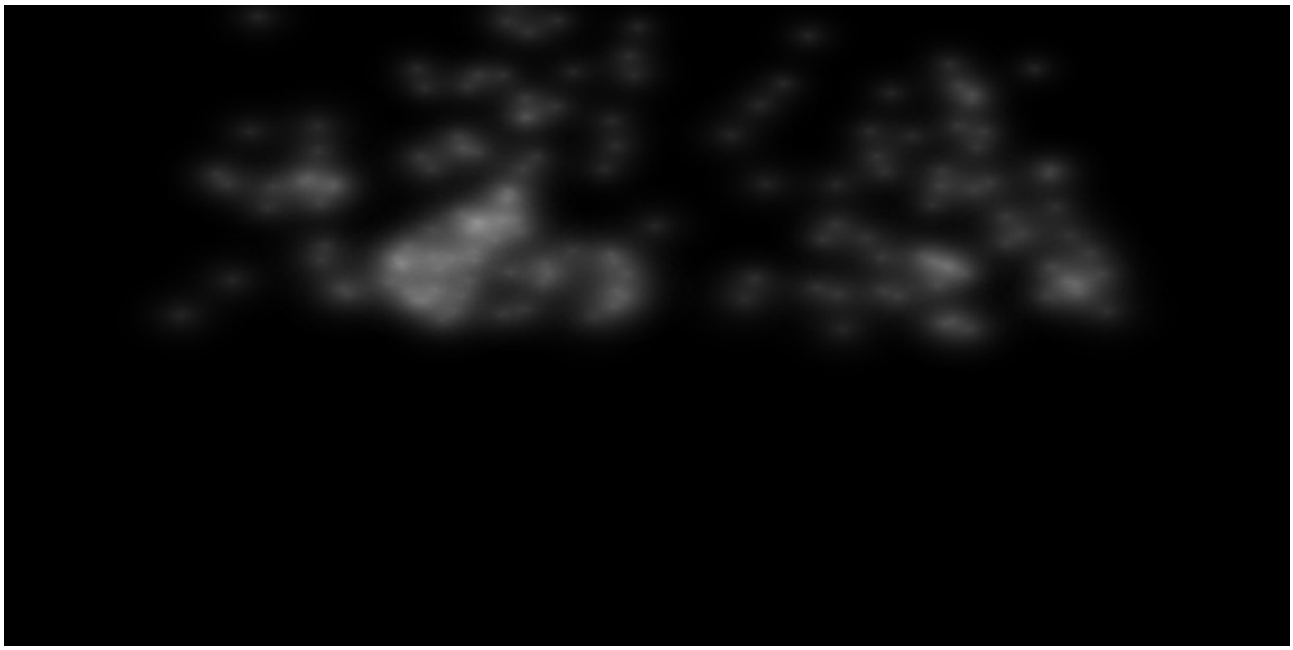
II. Approximation par les « QuickBalls »

Le rendu obtenu :

« script_fire_tutorial.ned »

Les « QuickBalls » permettent un rendu hybride de metaballs sans calcul de maillages. On définit des balles en « sprites » agencées par un ZBuffer avec intégration de l'environnement ou non dans la surface de virtualisation.

Le rendu des « QuickBalls » se fait donc en 2 passes, la première :



Par la suite, on peut définir un volume dont on obtient les normales dans la passe de rendu :

```
pos = coord(Tex1);              // position projection
val = coord(Tex2);
param = coord(Tex3);
col = sample(1,Tex0);           // coo texture
```

Le rendu du volume dans l'espace de projection est définit par la couleur (0,0,0) de la surface de virtualisation.

A partir des différentielles, on obtient les composantes tangentielles des normales :

```
Dx = texture(x+1,y) – texture(x-1,y) ;
Dx = texture(x,y+1) – texture(x,y-1) ;
```

Il suffit d'appliquer un rendu quelconque, pour des effets de fumées, feux et autres modélisations de volumes comme les métaballs.

Il y a effectivement un problème de saturation de la surface de virtualisation, et la composante de profondeur peut être intégrée simplement.

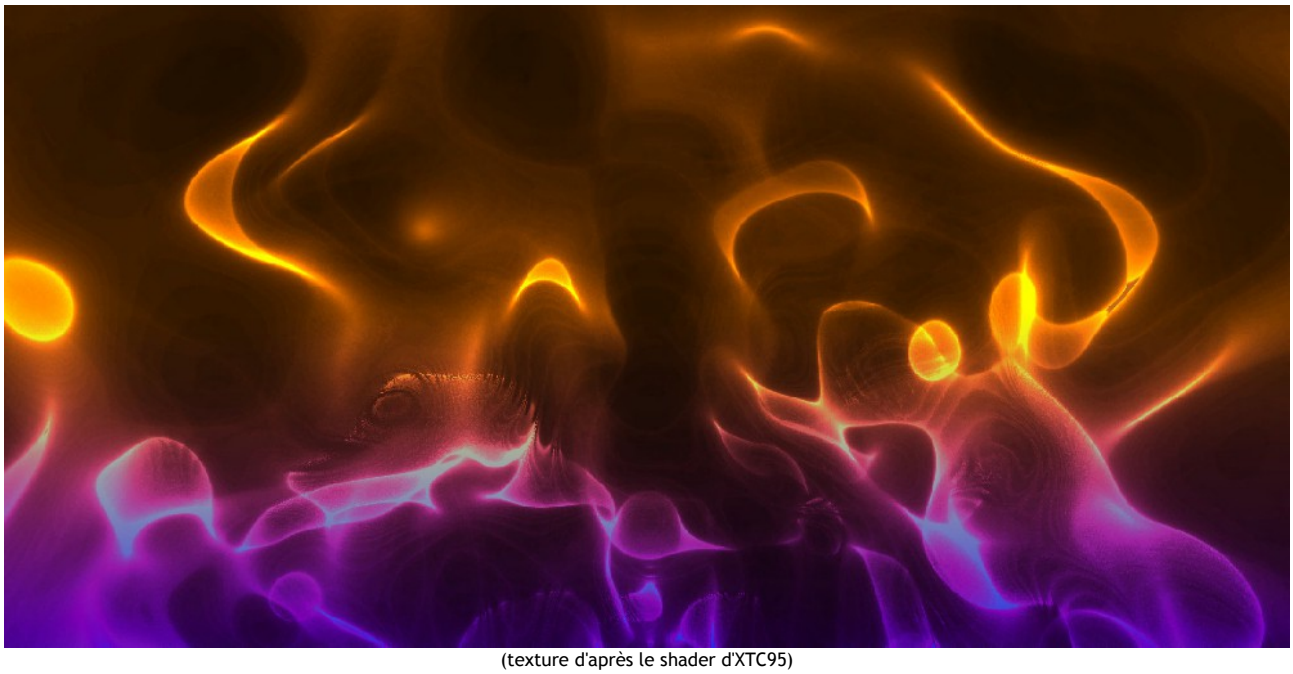
III. Approximation de l'équation d'une flamme

Comme dans une approche discrète ou particulaire, on peut animer le feu selon une « danse » de trigonométrie quelconque :

Pour chaque particules :

```
// la vitesse décroit selon la température

v.y-=0.005;

// pseudo intégration du mouvement

p.x += v.x + R * 0.25 * sin(p.y * PI * 4) * sin(t * PI * 4);
p.y += v.y + 0.01 * abs(sin(p.y * PI));
p.z += v.z + R * 0.25 * cos(p.y * PI * 4);

// la topologie de réduction

R-=0.005*abs(sin(t*PI));

// contraintes

if ((p.y<HAUTEUR_SIMULATION)||(R<0.005)||(rand(1000)>990))
        replacewithnewparticulesfromemitters(n) ;
```

Dans le cas d'une simulation de feu, on peut moduler le rendu par des sinusoïdes :



(texture d'après le shader d'XTC95)

Le rendu des particules modulées devient simplement une approximation de réflexions volumiques à partir des différentielles !

IV. Applications

« *SkullTown Adventures* »