

Textures : un peu de maths

Nicolas Holzschuch
iMAGIS/GRAVIR-IMAG

iMAGIS is a joint project of CNRS - INPG - INRIA - UJF



Plaquage de textures

- On part d'une fonction 2D
 - la « texture »
- On veut établir une correspondance entre les coordonnées de la texture et les coordonnées de l'objet



Algorithmes de plaquage

- Forward Mapping (basé sur la texture) :
 - **pour tout** (u, v)
 - **trouver** $x(u, v) = f(u, v)$
 - $image[x][y] = texture[u][v]$
- Backward Mapping (basé sur l'écran) :
 - **pour tout** (x, y)
 - **trouver** $(u, v) = f^{-1}(x, y)$
 - $image[x][y] = texture[u][v]$



Algorithmes de plaquage

- Algorithme en deux passes :
 - **pour tout** (u, v)
 - **trouver** $x(u, v)$
 - $tmp[x][v] = texture[u][v]$
 - **pour tout** (x, v)
 - **trouver** $y(x, v)$
 - $image[x][y] = tmp[x][v]$

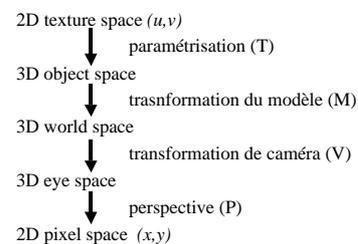


Quel algorithme ?

- basé sur la texture :
 - Peut laisser des trous à l'écran
 - Mais pas d'inversions à faire
 - Applications temps-réel
- Basé sur l'écran :
 - Il faut inverser f
 - Intégré dans les cartes graphiques
- Deux passes :
 - Toujours 1D
 - Plus de coût mémoire



Coordonnées



Paramétrisation des objets

- Conversion de la texture à l'objet
- Direct pour les triangles :
 - Et linéaire aussi : $x = au + bv + c$
- Pour les autres objets :
 - non-linéaire en général
 - On préfère subdiviser en triangles
- Cas particulier : les parallélogrammes :
 $(x, y, z, w) = M(u, v, I)$ (linéaire)



Backward mapping

- Toutes les transformations sont linéaires :
 $(x, y, w) = PVMT(u, v, I)$
- L'inverse est aussi linéaire :
 $(u, v, w) = (PVMT)^{-1}(x, y, I)$
- Linéaire, mais homogène :
 $u = (ax + by + c) / (gx + hy + i)$
 $v = (dx + ey + f) / (gx + hy + i)$



Projection Orthographique

- La matrice de projection orthographique est affine :
 - $w' = w = 1$
- L'inverse est affine aussi :
 - $w = w' = 1$
 - On peut faire du texture mapping avec seulement deux additions par pixel



Deux additions par pixel

- $u = (a/i)x + (b/i)y + (c/i)$
- $v = (d/i)x + (e/i)y + (f/i)$
- Algorithme :
for all y
 $u = (a/i)x_{min} + (b/i)y + (c/i)$
 $v = (d/i)x_{min} + (e/i)y + (f/i)$
for all x
 $u += (a/i); v += (d/i)$
 $image[x][y] = texture[u][v]$



Perspective

- Forme Générique :
 $u = (ax + by + c) / (gx + hy + i)$
 $v = (dx + ey + f) / (gx + hy + i)$
- 3 additions, 2 divisions par pixel
- Peut être optimisé pour les cas particuliers :
 - Polygones horizontaux/verticaux (Doom)



3 additions, 2 divisions

- Algorithme :
for all y
 $unum = a x_{min} + b y + c$
 $vnum = d x_{min} + e y + f$
 $den = g x_{min} + h y + i$
for all x
 $image[x][y] = texture[unum/den][vnum/den]$
 $unum += a$
 $vnum += b$
 $den += g$

