Fonctions avancées de rendu 3D

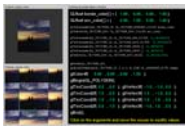
Gilles Debunne
DEA IVR
2004



Plan du cours

- Utilisation des textures
  - Principe
  - Matrice de texture
  - Mipmapping
  - Textures d'environnement
- Transparence
- Tests additionnels sur les fragments
- Techniques avancées
- Optimisations et problèmes
- Evolution des cartes graphiques

Textures



- Initialisation de la texture

```
glEnable(GL_TEXTURE_2D);
glGenTextures(1,&texID);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGBA,…, bits);
```

- Choix de la texture courante

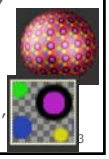
```
glBindTexture(GL_TEXTURE_2D, texID);
```

- Paramètres (répétition, interpolation…)

```
glTexParameteri(GL_TEXTURE_2D, … , …)
```

- Mélange avec les couleurs des faces

```
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,
          GL_MODULATE/BLEND/REPLACE/DECAL);
```



Coordonnées de textures : u,v,s,t

- Spécifiées pour chaque sommet

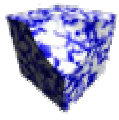
```
glTexCoord2f(u,v);
glVertex3f(x,y,z);
```

- Points à coordonnées multiples
  Nécéssaire à la frontière entre deux textures
- Génération automatique possible
  Texture d'environnement `(SPHERE_MAP)`
- Filtrage par la matrice de texture

Exemples : textures 1D, 2D, 3D

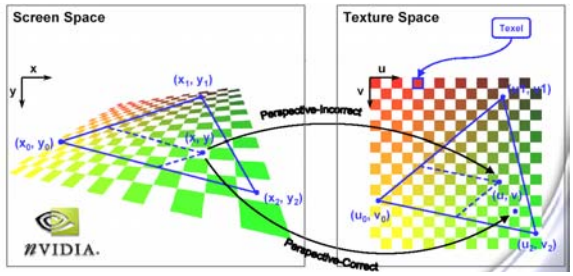
- 1D : u est un potentiel sur une surface
  Intérêt : pas de mélange de couleurs
  Applications : courbes de niveau, isovaleurs
- 2D : Ajout de détails

- 3D : Marbre, bois,…

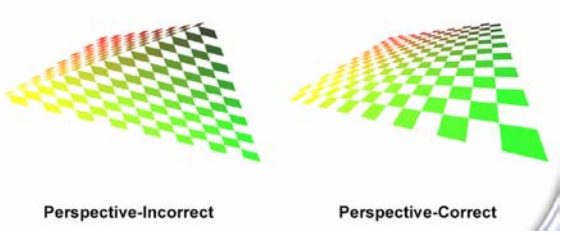


Interpolation des coordonnées

- Prise en compte du Z



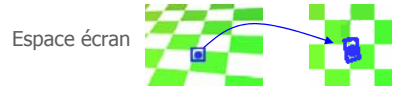
Intêret de la correction



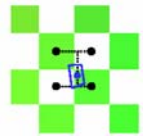
- Non bijection entre texels et pixels

Pixel plus petit que le texel

- Paramètre GL_TEXTURE_MAG_FILTER



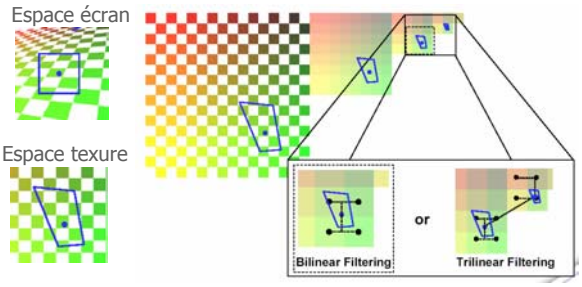
Couleur du texel le plus proche
`GL_NEAREST`



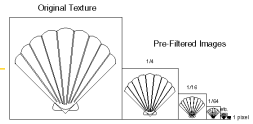
Filtrage bi-linéaire
`GL_LINEAR`

Plusieurs texels dans le même pixel

- Paramètre GL_TEXTURE_MIN_FILTER



Mipmapping



- Hiérarchie de textures
- Evite le clignotement (aliassage)
- Préfiltrage des couleurs
- Génération automatique ou manuelle

```
gluBuild2DMipmaps(…)
```

⚠ Il faut spécifier tous les niveaux !

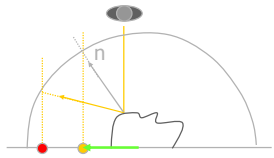
- Amélioration par filtrage anisotrope :
  Plusieurs échantillons par pixel

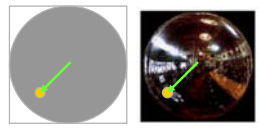
Matrice de texture

- Après `GL_MODELVIEW` et `GL_PROJECTION`
  `glMatrixMode(GL_TEXTURE)`
- S'applique sur u,v,s,t
- Nombreuses applications possibles
  Ex : Projection inverse d'une diapositive
  u,v,s,t = x,y,z,w
  Matrice = matrice de la projection
  Résultat : u,v correspondant
  Déformation de la perspective, …

Textures d'environnement

- Calcul des coordonnées du rayon réfléchi





```
glEnable(GL_TEXTURE_GEN_X)
glTexGeni(GL_X,GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP)
```

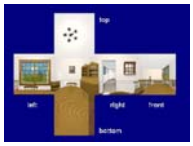
Avec X = S et T

- Calcul faux (cos), la texture doit compenser

Cartes cubiques

- Extension des textures d'environnement
- 6 textures, sur un cube autour de la scène
- Permet de tourner autour de l'objet
- Aussi : normalisation de vecteurs

  v = (x,y,z) → (r,g,b) = (x,y,z) / ||v||



Plan du cours

- Utilisation des textures
- Transparence
- Tests additionnels sur les fragments
- Techniques avancées
- Optimisations et problèmes
- Evolution des cartes graphiques

Transparence α=opacité

- Combinaison des couleurs

  Fragment dessiné $c_1=(r_1,g_1,b_1,\alpha_1)$

  Couleur actuelle du pixel $c_2=(r_2,g_2,b_2,\alpha_2)$

  $$c = \alpha_1 c_1 + (1-\alpha_1) c_2$$

  1
- Différentes fonctions de mélange

```
glBlendFunc(source, dest);
glEnable(GL_BLEND);
```

- Buffer RGBA non nécessaire (en back to front)

Transparence : difficultés

- Limitations
  - Approximation de la physique
  - Rendu de l'arrière vers l'avant
    - Sauf si une seule épaisseur est transparente
- Mauvaise fonction de mélange par défaut

  Utiliser `glBlendFunc(`α_{src}`, 1-`α_{src}`);`

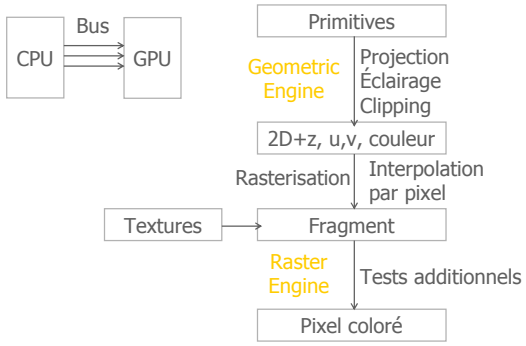
  ou mieux (si toutes les couleurs sont *premultiplied alpha*)

```
glBlendFunc(1, 1);
```


Plan du cours

- Utilisation des textures
- Transparence
- Tests additionnels sur les fragments
  - Principe
  - Alpha
  - Stencil
- Techniques avancées
- Optimisations et problèmes
- Evolution des cartes graphiques

Le pipeline graphique

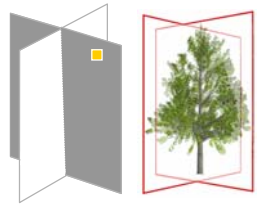


Tests additionnels des fragments

- Choix du test `glBufferFunc(…)`
  Condition d'élimination d'un fragment
- Résultat du test `glBufferOp(…)`
  Que se passe-t-il pour les fragments qui passent ?
  Mise à jour du buffer, de la couleur…
- Les *Buffer* concernés (dans l'ordre)
  Alpha Buffer `glEnable(GL_ALPHA_TEST)`
  Stencil Buffer `glEnable(GL_STENCIL_TEST)`
  Z Buffer `glEnable(GL_DEPTH_TEST)`

Exemple : Alpha Test

- `glAlphaFunc(GL_GREATER, 0.0);`
- Opacité entre 0 et 255
  Valeur du Z modifiée ssi $\alpha>0$
  Fragment rejeté sinon



Exemple : Stencil Buffer

- Mise à jour d'un pochoir lors du rendu
- Mise à jour *aussi* quand Z ou stencil *échoue*
- Valeurs vues comme des entiers ou des bits
  Un automate à états finis par pixels
- Applications
  Mirroirs, Ombres
  Masque pour des filtrages d'images (souris…)
  Nombre de polygones sous chaque pixel

Plan du cours

- Utilisation des textures
- Transparence
- Tests additionnels sur les fragments
- Techniques avancées
  Color matrix
  Multi-passes, cartes de lumières
- Optimisations et problèmes
- Evolution des cartes graphiques

Modification de couleur

- Filtre appliqué lors du rendu
  A la couleur du fragment et/ou de la texture
- Deux types de filtres
  Matrice de couleur (4x4) appliquée à r,g,b,a
  `glColorTable` $r'=f_r(r)$, $g'=f_g(g)$, $b'=f_b(b)$
- Application
  Modification de teinte
  Couleur selon la distance
  Cartes de densités (muscles, vaisseaux, …)

Rendu multipasses

- Combinaison de couleurs de pixels
  `glBlendFunc(source, dest);`
  `glBlendEquation(+, *, min, max,…);`
- Zone éventuellement limitée par le stencil buffer
- Surcout lié au nombre de passes
  Supprimer les options inutiles des passes >1
  `(LIGHTING,SMOOTH_SHADING, Z Test,…)`
- Applications : multi-texture, reflets, ombres..

Exemple : cartes de lumière

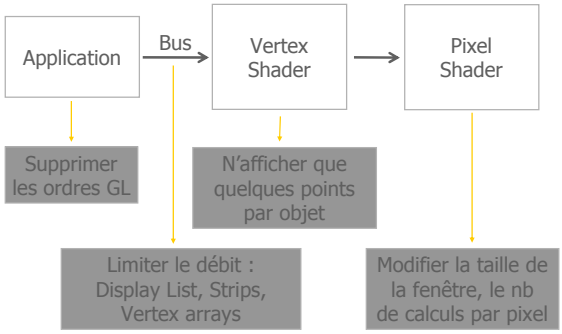
- Eclairage précalculé

```
glTexEnvf(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_MODULATE);
```


Plan du cours

- Utilisation des textures
- Transparence
- Tests additionnels sur les fragments
- Techniques avancées
- Optimisations et problèmes
  - Accélerer le rendu
  - Problèmes classiques
- Evolution des cartes graphiques

Les goulots d'étranglement



Optimisations

- `GL_CULL_FACES` (50% du Raster)
- `TriangleStrip, TriangleFan`
- Display Lists, Vertex Buffer Objects
- `glColorMaterial(GL_FRONT_AND_BACK, …)`
- Préférer `glMaterial` si peu de modifications
- Lumières à l'infini, éviter les spots
- Options inutiles du multi-passes

Ça ne marche pas !

- Caméra mal placée par les transformations
- Pas de couleur ou de normale définie
- Normales transformées par un `glScale`

```
glEnable(GL_RESCALE_NORMAL); (gl >= 1.2)
```

- Géométrie à l'envers et `CULL_FACE`
- Dernier u,v à 0,0 ou normale vers le fond
- Oubli de `glLoadIdentity()`
- `GL_COLOR_MATERIAL` activé (ou non)

Ça ne marche toujours pas !

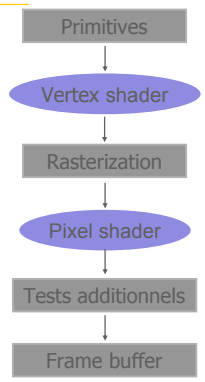
- Les lumières subissent la `GL_MODELVIEW`
- `glBlendFunc` est mal choisie par défaut
- Le z lu dans le frame buffer n'est pas linéaire
- Les polygones sont triangulés → interpolation
- Changement de `MatrixMode` entre temps
- Taille de texture non puissance de 2
- Manque un niveau de mipmap
- Transfos dans `GL_PROJECTION` et brouillard

Plan du cours

- Utilisation des textures
- Transparence
- Tests additionnels sur les fragments
- Techniques avancées
- Optimisations et problèmes
- Evolution des cartes graphiques
  - Vertex & Pixel shaders
  - Applications et futur

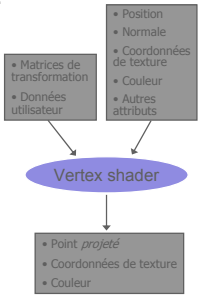
Pipeline graphique programmable

- GPU de plus en plus puissant : SIMD
- 10G pixels / sec
- 800M points / sec
- Les opérations deviennent programmables
- nVidia vs ATI



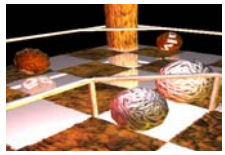
Vertex shader

- Transformation géométrique
  - Interpolation, morphing, …
- Souplesse d'utilisation
  - Ex : couleur = normale



Pixel shader

- Indirections dans les textures
- Calculs sur ces valeurs
- Applications
  - Performances, souplesse
  - Matériaux complexes (Phong, bump)



Les cartes programmables

- Evolution très rapide
  - ATI Radeon 9800, nVidia GeForce 6
- Shader par langages haut niveau
  - HLSL (DirectX), GLSL (openGL), cg, …
- Limitations
  - Types de calculs, indirections dans les textures
  - Souplesse des programmes : SIMD → synchro
  - Nombre d'instructions : mémoire et vitesse

Le futur

- Plus de : mémoire, vitesse, bande passante
- Des langages unifiés de haut niveau
- De la 3D partout (PDA, voitures, maison...)