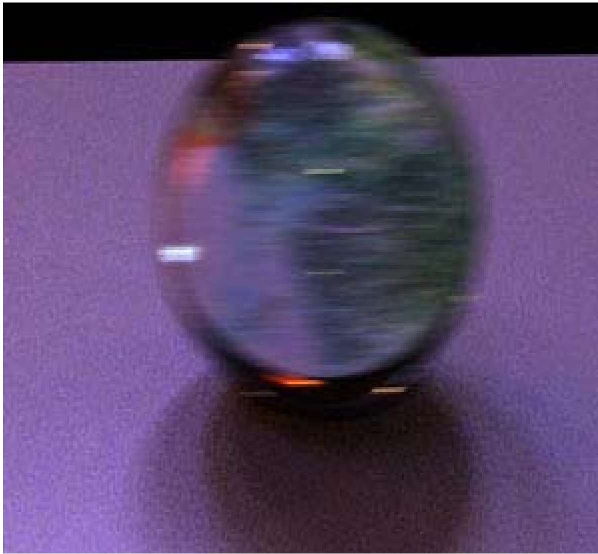
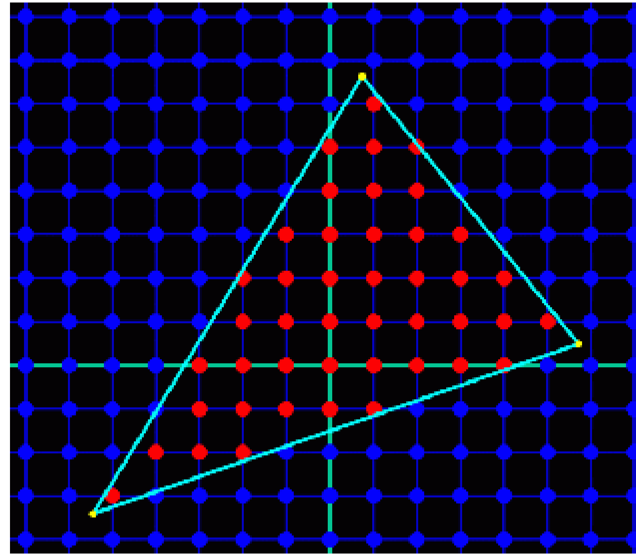
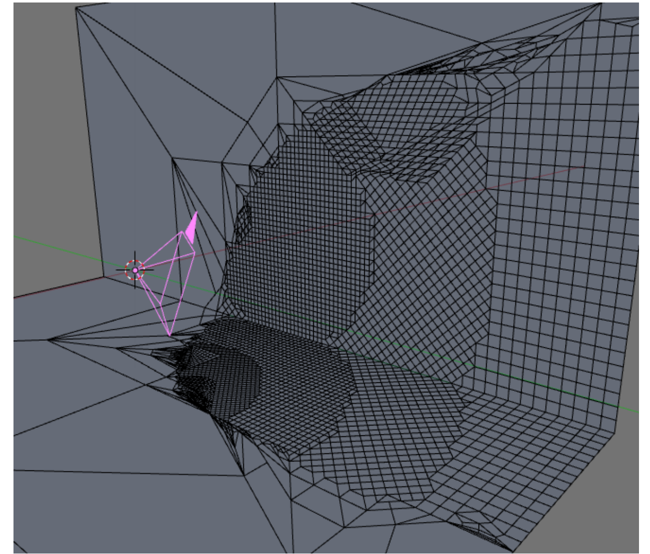




IFT3355

INFOGRAPHIE

SECTION 13: SHADERS

Derek Nowrouzezahrai et Pierre Poulin

Département d'informatique et de recherche opérationnelle

Université de Montréal

Université de Montréal

Motivation

Université de Montréal

Motivation

Université de Montréal

Motivation

Université de Montréal

Motivation

Université de Montréal

Motivation

Université de Montréal

Survol

- Le pipeline OpenGL
  - Transformation des sommets
  - Rastérisation des formes
  - Affichage des pixels

- Shaders: *vertex* et *fragment* (pixel)
  - Introduction à la programabilité du pipeline de OpenGL v1.5

Le pipeline OpenGL

```
// Dessiner un triangle avec: coordonnées texture, normales, et couleur
glBegin(GL_TRIANGLES);
   glNormal3f(0, 0, 1); // normale par triangle
   glTexCoord2i(0, 0);
   glColor3f(1, 0, 0);
   glVertex3f(0, 0, 0);

   glTexCoord2i(0, 1);
   glColor3f(0, 1, 0);
   glVertex3f(0, 1, 0);

   glTexoord2i(1, 0);
   glColor3f(0, 0, 1);
   glVertex3f(1, 0, 0);
glEnd();
```

Université de Montréal

Le pipeline OpenGL

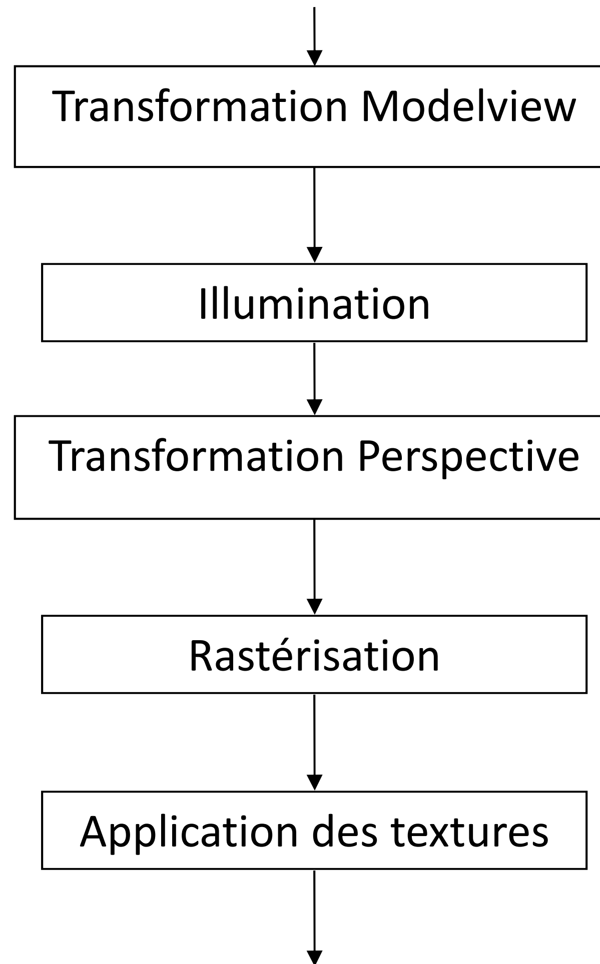
```
// Dessiner un triangle avec: coordonnées texture, normales, et couleur
glBegin(GL_TRIANGLES);
   glTexCoord2i(0, 0);
   glColor3f(1, 0, 0);
   glNormal3f(0, 0, 1); // normale par sommet
   glVertex3f(0, 0, 0);

   glTexCoord2i(0, 1);
   glColor3f(0, 1, 0);
   glNormal3f(0, 0, 1); // normale par sommet
   glVertex3f(0, 1, 0);

   glTexoord2i(1, 0);
   glColor3f(0, 0, 1);
   glNormal3f(0, 0, 1); // normale par sommet
   glVertex3f(1, 0, 0);
glEnd();
```

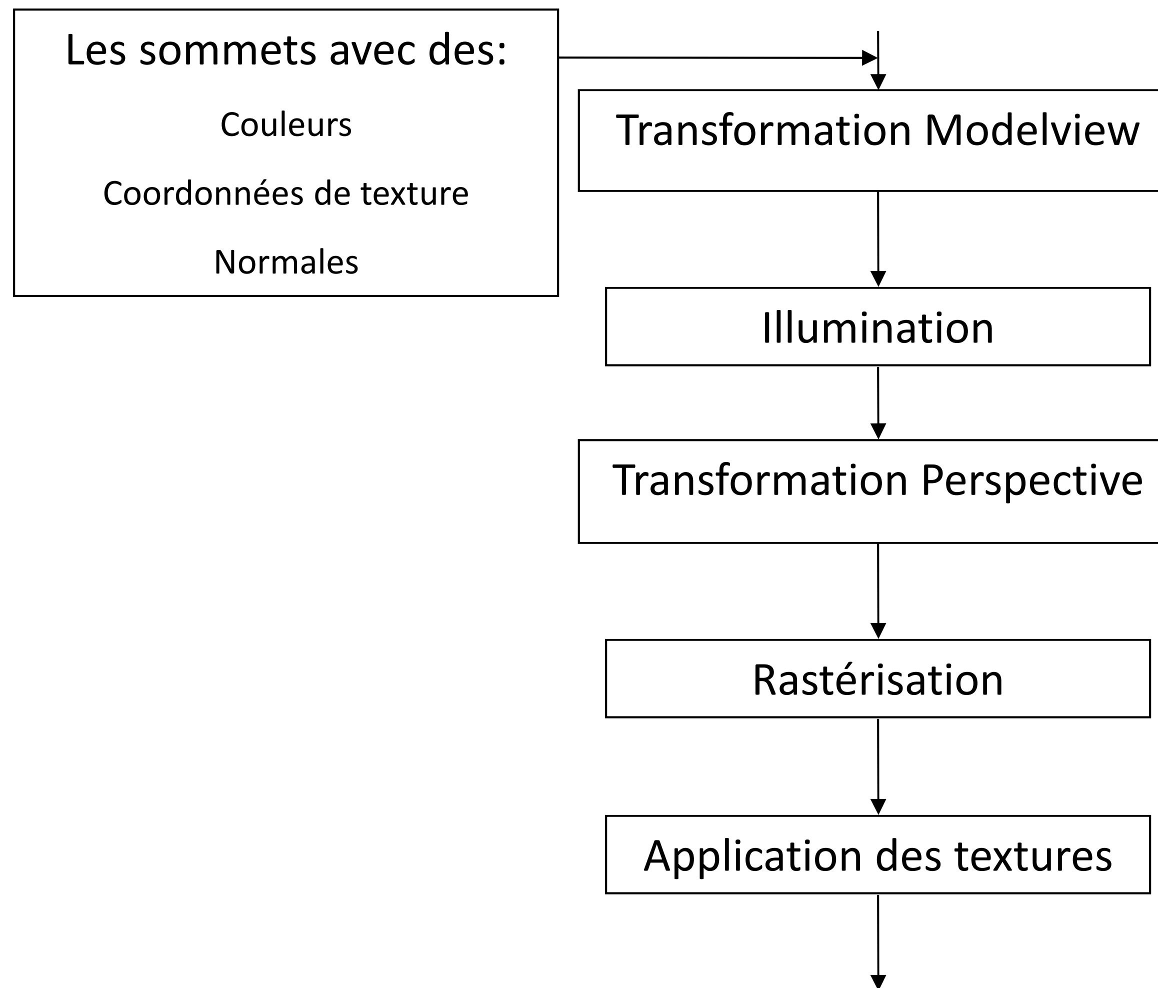
Le pipeline OpenGL fixe

(*Fixed Function Pipeline*)



Le pipeline OpenGL fixe

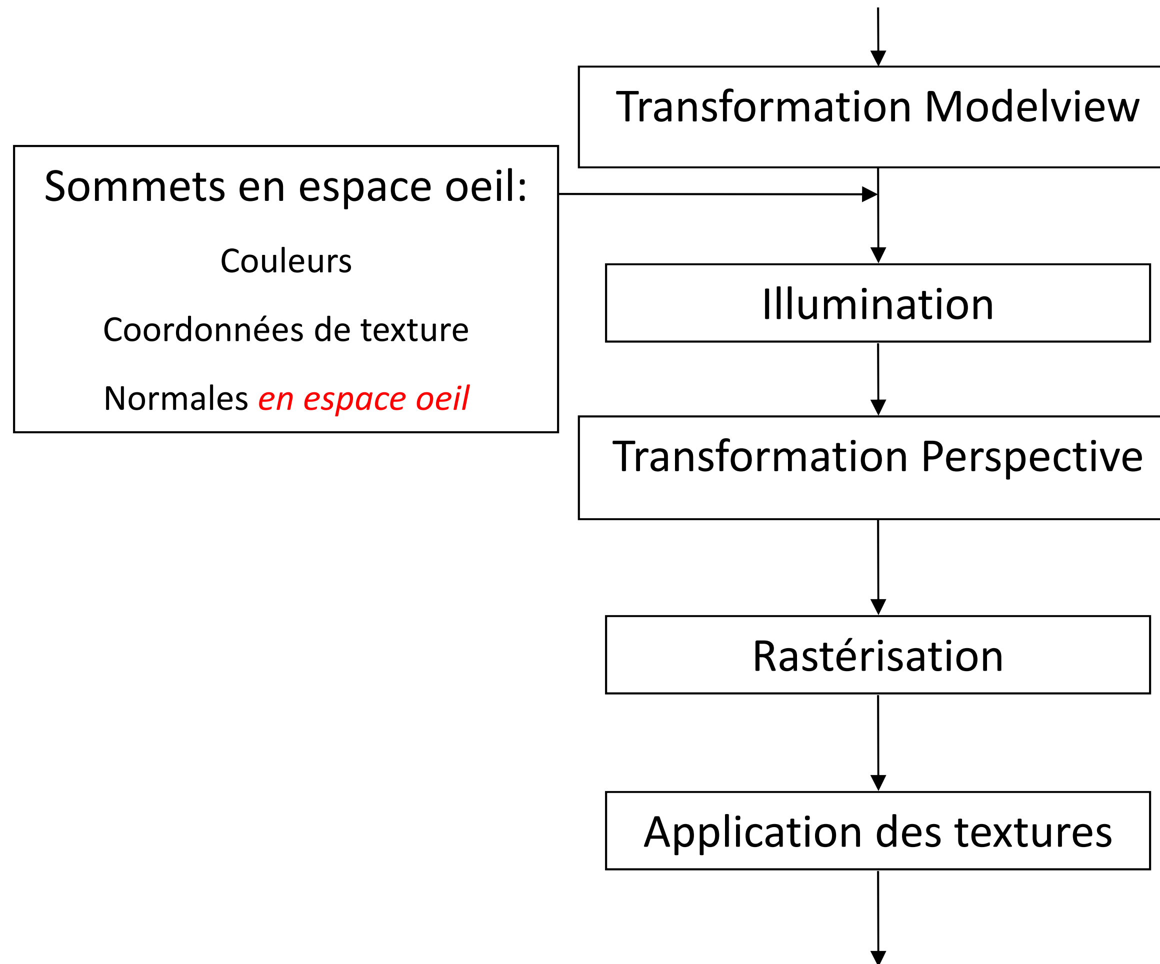
(*Fixed Function Pipeline*)



Université de Montréal

Le pipeline OpenGL fixe

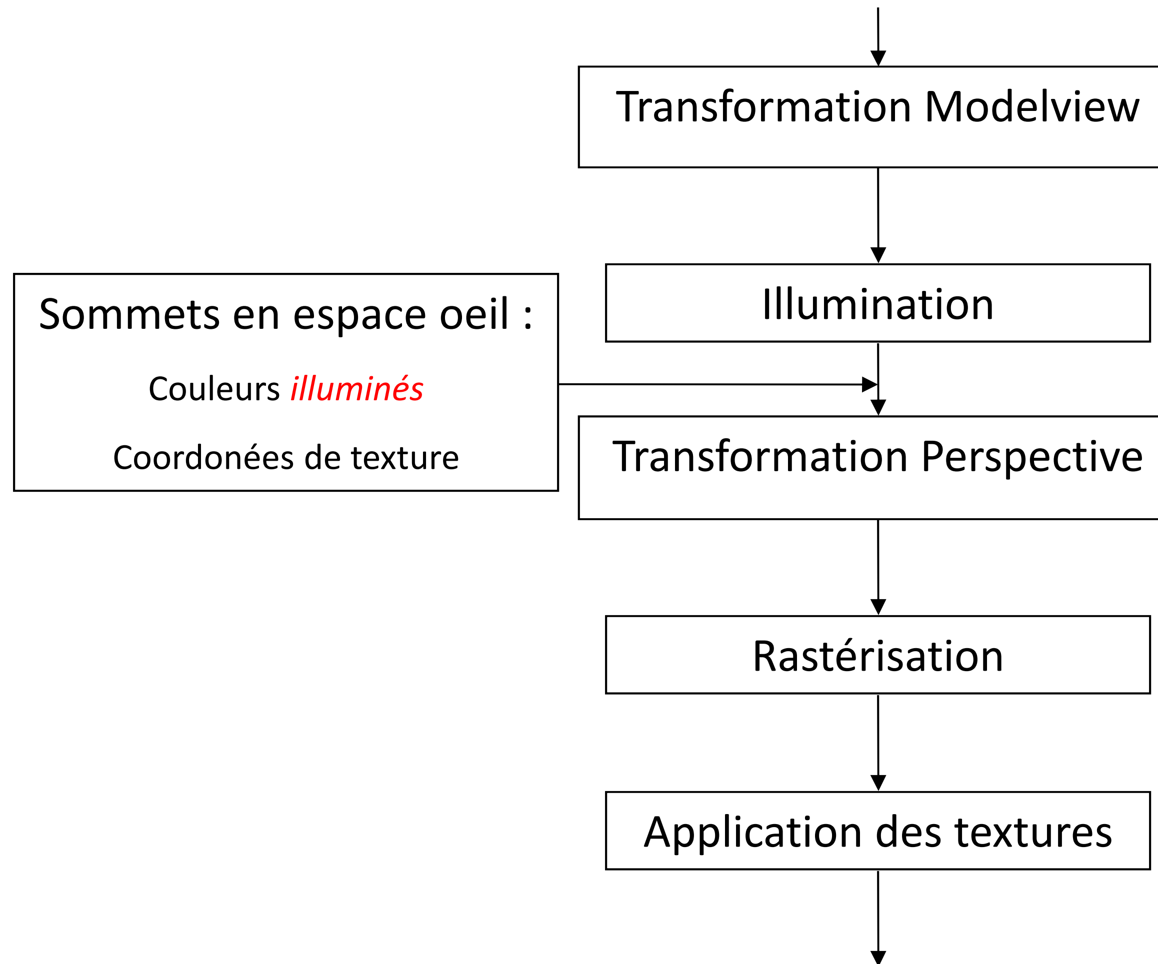
(*Fixed Function Pipeline*)



Université de Montréal

Le pipeline OpenGL fixe

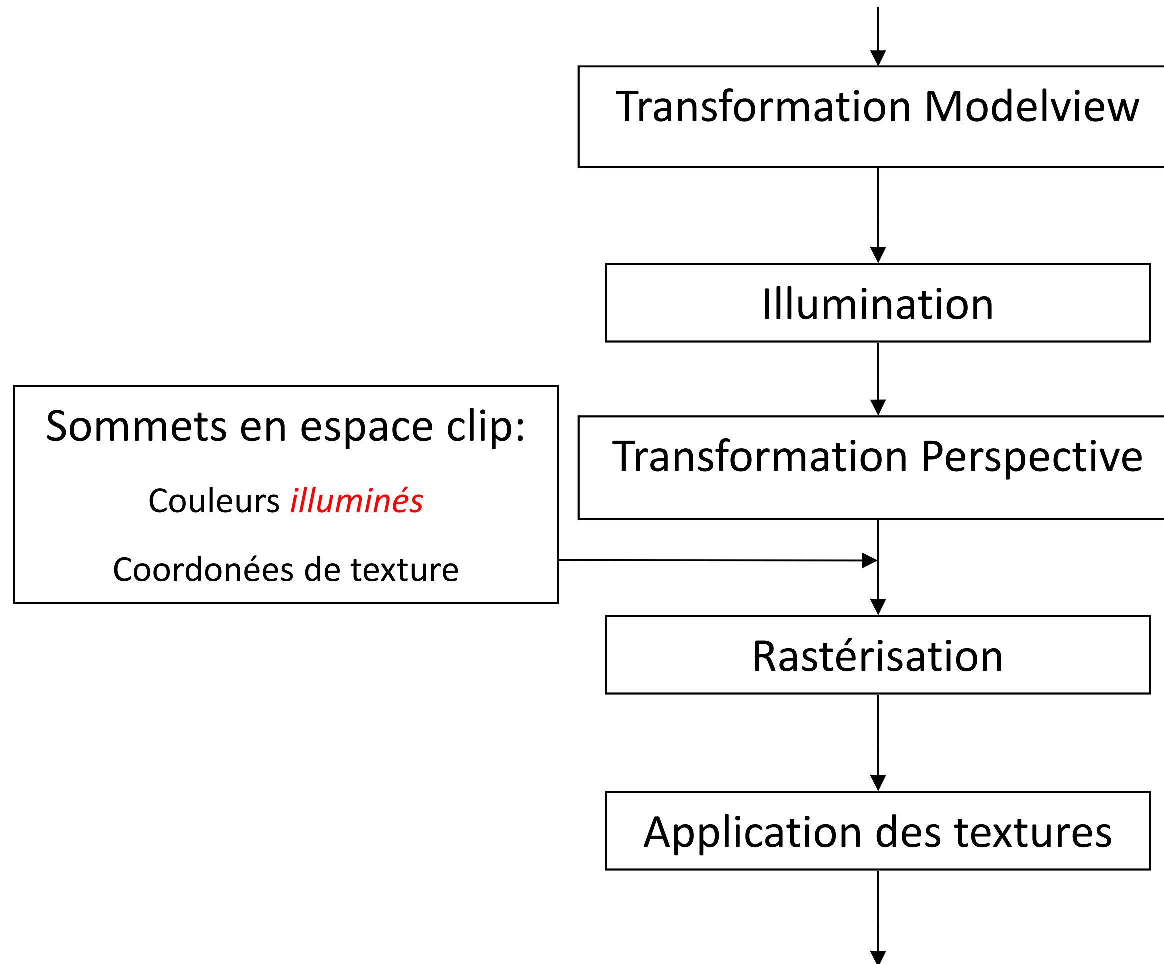
(*Fixed Function Pipeline*)



Université de Montréal

Le pipeline OpenGL fixe

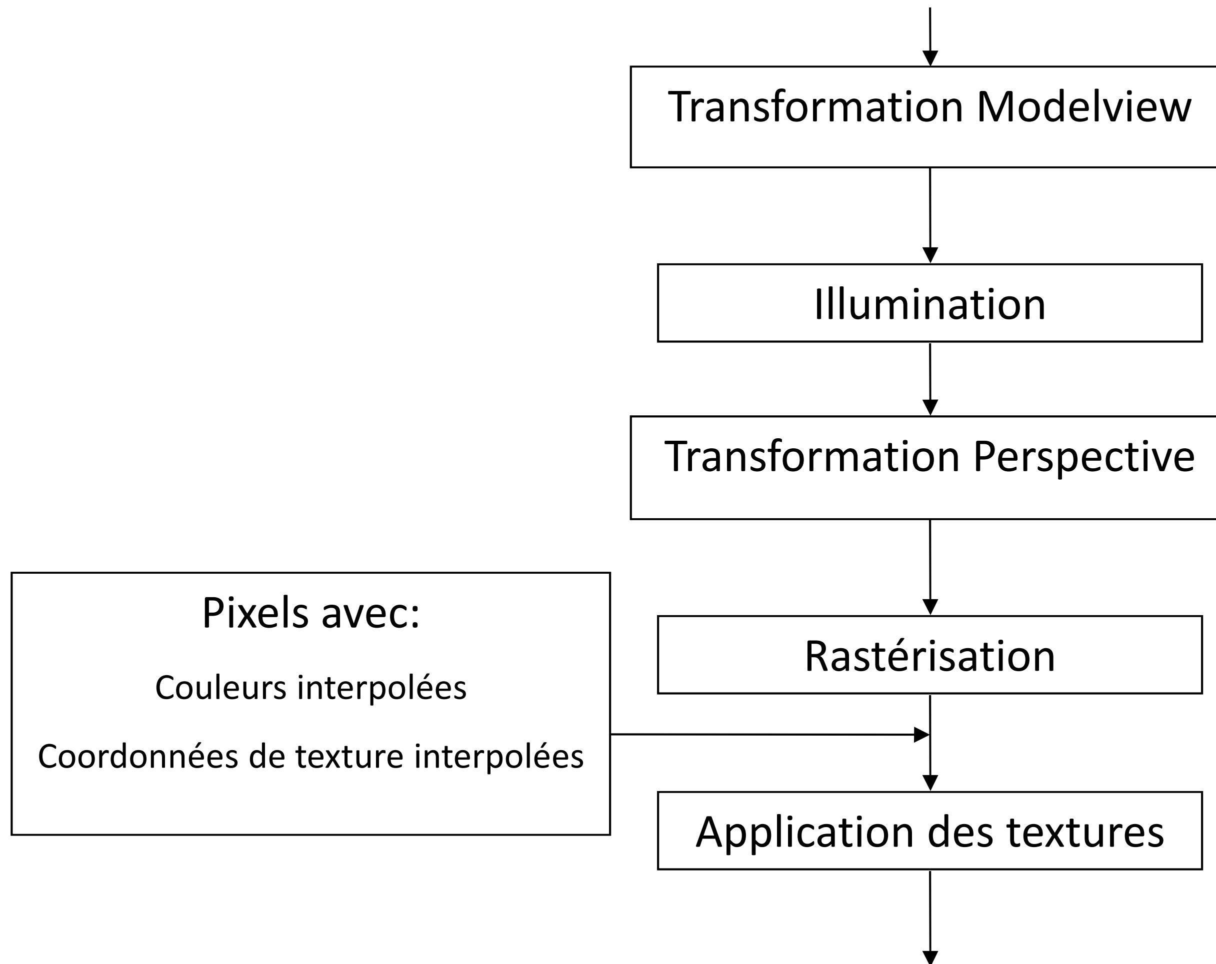
(*Fixed Function Pipeline*)



Université de Montréal

Le pipeline OpenGL fixe

(*Fixed Function Pipeline*)



Université de Montréal

Le pipeline OpenGL fixe

(*Fixed Function Pipeline*)

Transformation Modelview

Illumination

Transformation Perspective

Rastérisation

Application des textures

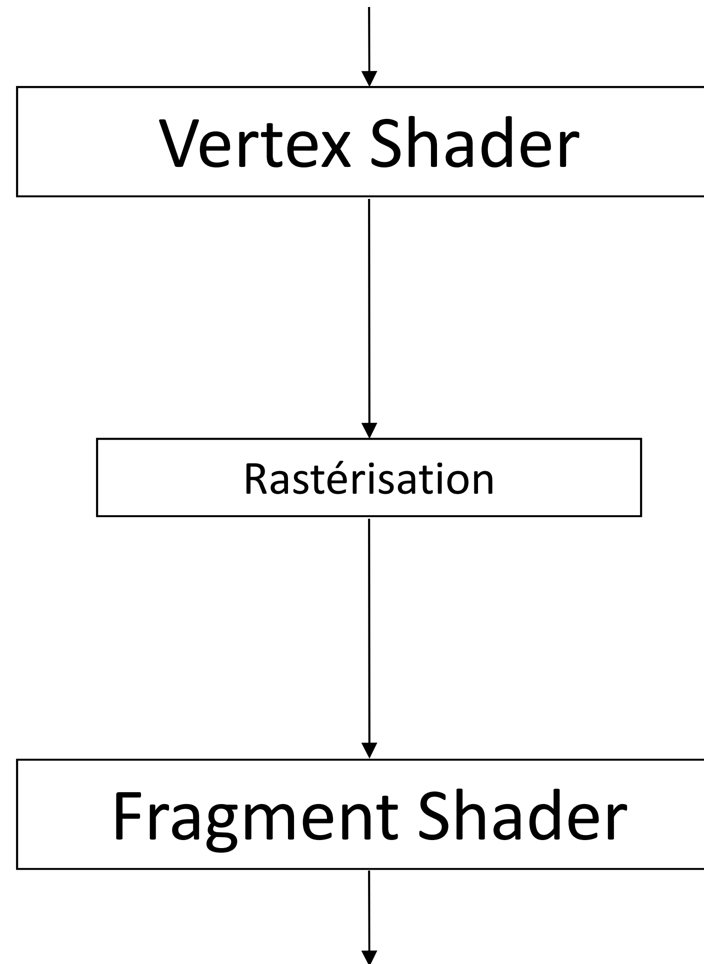
Mémoire de texture sur GPU

Pixels avec:

Couleurs affichés

Le pipeline OpenGL avec les **shaders**

(Programmable Pipeline)



Université de Montréal

Le pipeline OpenGL avec les **shaders**

(Programmable Pipeline)

Vertex Shader

Rasterization

Fragment Shader

Les shaders sont des petits programmes, coder par le dévéloppeur, qui tourne sur le GPU (en parallèle)

Le pipeline OpenGL avec les **shaders**

(Programmable Pipeline)

Les sommets avec des:

Couleurs

Coordonnées de texture

Normales

Vertex Shader

Rasterization

Fragment Shader

Université de Montréal

Le pipeline OpenGL avec les **shaders**

(*Programmable Pipeline*)

Vertex Shader

Sommets transformés:

(Ce que tu veux ici, *ex:* normales, couleurs, coords texture, etc…)

Mémoire de texture sur GPU

Rasterization

Fragment Shader

Le pipeline OpenGL avec les **shaders**

(*Programmable Pipeline*)

Vertex Shader

Rasterization

Pixels avec:

(Les valeurs fournies en sortie du *vertex shader*, interpolés à travers les pixels)

Fragment Shader

Le pipeline OpenGL avec les **shaders**

(Programmable Pipeline)

Vertex Shader

Rasterization

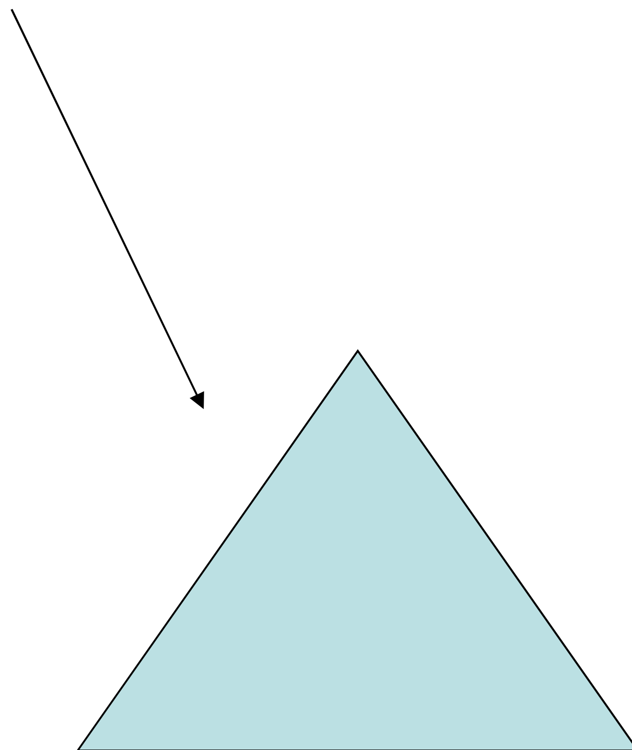
Mémoire de texture sur GPU

Fragment Shader

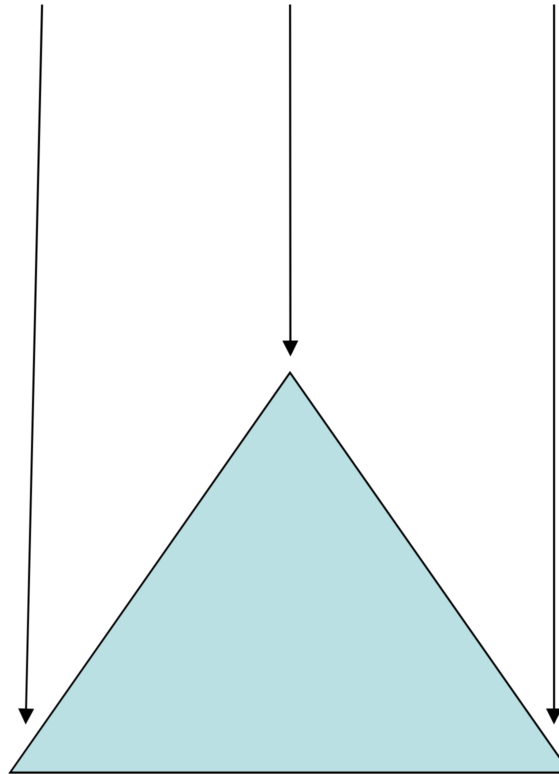
Pixels avec:

Couleurs affichés

Combien de fois est-ce que les *vertex shaders* et *pixel shaders* vont-ils être appelés pour cette scène?



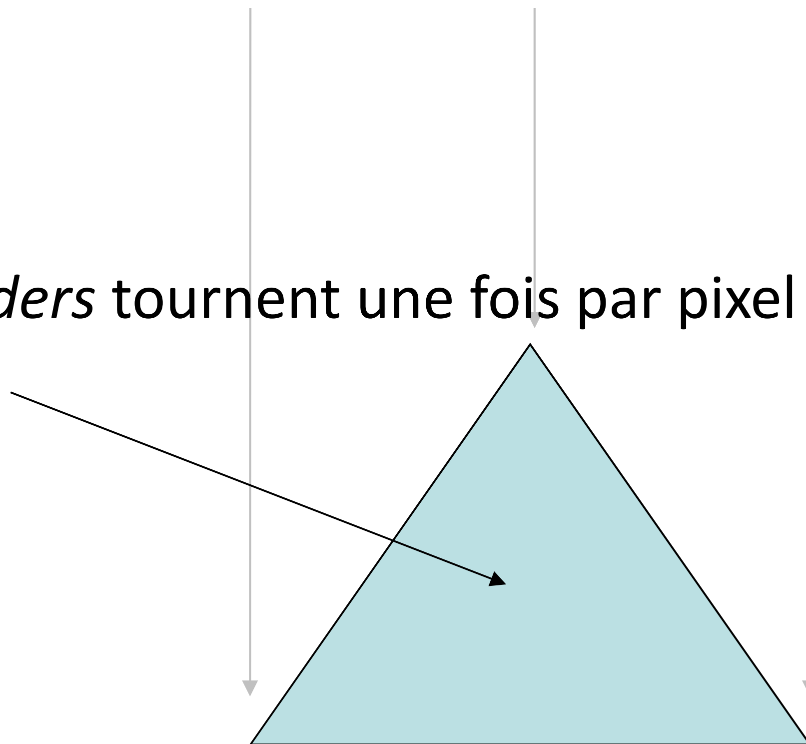
Les *vertex shaders* tournent une fois par sommet



Université de Montréal

Les *vertex shaders* tournent une fois par sommet

Les *pixel shaders* tournent une fois par pixel



Vertex Shader de base

```
void main() {
    gl_Position = gl_Vertex;
}
```

Fragment (pixel) shader de base

```
void main() {
    gl_FragColor = vec4(1, 0, 0, 1);
}
```

Vertex Shader de base

```
void main() {
    gl_Position = gl_Vertex;
}
```

Cela ignore tous ce qui concerne les transformations des sommet par les matrices modelview et projection

Fragment (pixel) shader de base

```
void main() {
    gl_FragColor = vec4(1, 0, 0, 1);
}
```

De côté: language de shader

- J’ai choisi (d’une manière arbitraire) d’utiliser GLSL pour cette lecture
- Il existe de nombreuses alternatives (FX, Cg, RSL, HLSL, etc.) chacune avec leurs propre avantages/désavatages

- Bonne attitude:
  - Choissisez un SL selon les besoins de ton platforme/application

Les types GLSL

- Types *floating-point*
  - **`float`**, **`vec2`**, **`vec3`**, **`vec4`**, **`mat2`**, **`mat3`**, **`mat4`**
- Types *boolean*
  - **`bool`**, **`bvec2`**, **`bvec3`**, **`bvec4`**
- Types entiers
  - **`int`**, **`ivec2`**, **`ivec3`**, **`ivec4`**
- Types textures
  - **`sampler1D`**, **`sampler2D`**, **`sampler3D`**

Les types GLSL

- Une variable peut être facultativement:
  - `const`
  - `uniform`
    - Défini sur le CPU en dehors du **`glBegin`** / **`glEnd`**
  - `attribute`
    - Défini sur le CPU à chaque sommet
  - `varying`
    - Défini dans le *vertex shader*, interpolé à travers les pixels et disponible dans le *fragment shader*

Université de Montréal

Vertex Shader

```
attribute float vertexGrayness;  // input du CPU par sommet
varying float fragGrayness;      // output du vertex shader
void main() {
   gl_Position = gl_Vertex;
   fragGrayness = vertexGrayness;
}
```

Fragment Shader

```
uniform float brightness;    // valeur constante par appel de dessin
varying float fragGrayness;  // interpolée pendant la rastérisation
void main() {
   const vec4 red = vec4(1, 0, 0, 1);
   const vec4 gray = vec4(0.5, 0.5, 0.5, 1);
   gl_FragColor = brightness * ( red * (1 – fragGrayness) +
                                 gray * fragGrayness );
}
```

Vertex Shader

```
attribute float vertexGrayness;  // input du CPU par sommet
varying float fragGrayness;      // output du vertex shader
void main() {
    gl_Position = gl_Vertex;
    fragGrayness = vertexGrayness;
}
```

Défini une fois sur le CPU par sommet, comme une couleur, normale, texcoord, etc.

Fragment Shader

```
uniform float brightness;    // valeur constante par appel de dessin
varying float fragGrayness;  // interpolée pendant la rastérisation
void main() {
    const vec4 red = vec4(1, 0, 0, 1);
    const vec4 gray = vec4(0.5, 0.5, 0.5, 1);
    gl_FragColor = brightness * ( red * (1 – fragGrayness) +
                                  gray * fragGrayness );
}
```

Vertex Shader

```
attribute float vertexGrayness;  // input du CPU par sommet
varying float fragGrayness;      // output du vertex shader
void main() {
   gl_Position = gl_Vertex;
   fragGrayness = vertexGrayness;
}
```

Défini une fois par forme sur le GPU, en dehors du **glBegin** / **glEnd**

Fragment Shader

```
uniform float brightness;    // valeur constante par appel de dessin
varying float fragGrayness;  // interpolée pendant la rastérisation
void main() {
   const vec4 red = vec4(1, 0, 0, 1);
   const vec4 gray = vec4(0.5, 0.5, 0.5, 1);
   gl_FragColor = brightness * ( red * (1 – fragGrayness) +
                                 gray * fragGrayness );
}
```

Vertex Shader

```
attribute float vertexGrayness;  // input du CPU par sommet
varying float fragGrayness;      // output du vertex shader
void main() {
   gl_Position = gl_Vertex;
   fragGrayness = vertexGrayness;
}
```

Déclarations correspondantes

Fragment Shader

```
uniform float brightness;     // valeur constante par appel de dessin
varying float fragGrayness;   // interpolée pendant la rastérisation
void main() {
   const vec4 red = vec4(1, 0, 0, 1);
   const vec4 gray = vec4(0.5, 0.5, 0.5, 1);
   gl_FragColor = brightness * ( red * (1 - fragGrayness) +
                                 gray * fragGrayness );
}
```

Vertex Shader

```
attribute float vertexGrayness;  // input du CPU par sommet
varying float fragGrayness;      // output du vertex shader
void main() {
   gl_Position = gl_Vertex;
   fragGrayness = vertexGrayness;
}
```

Défini à chaque sommet par le vertex shader

Interpolée dans le HW

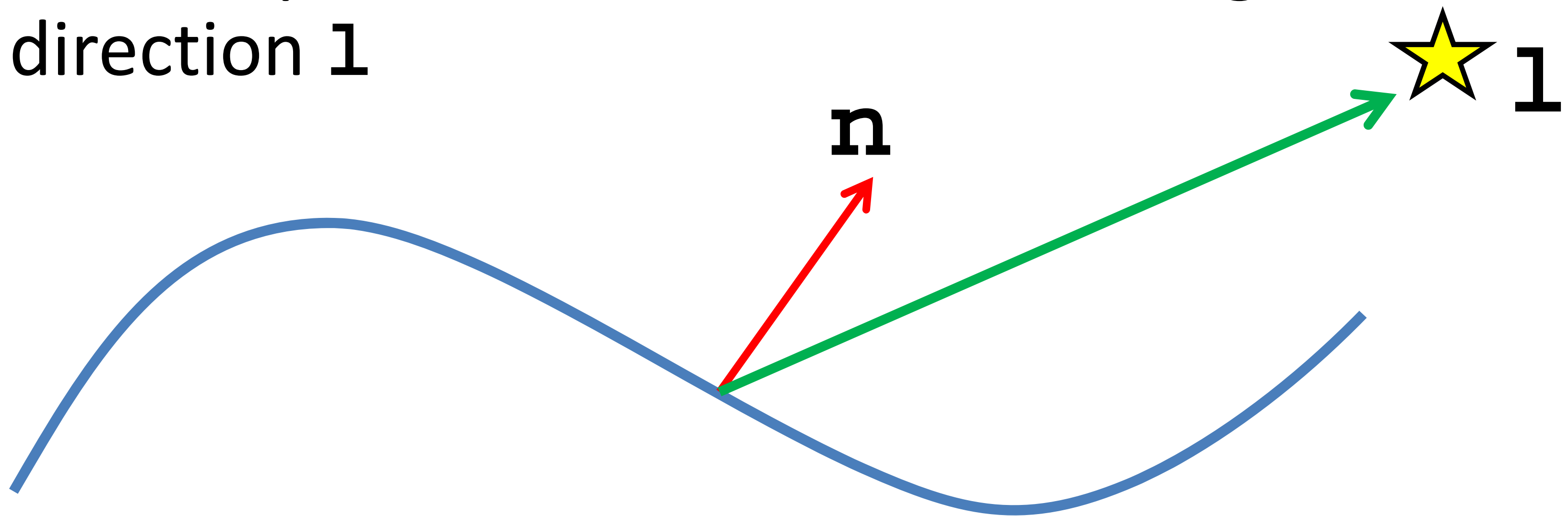
Fragment Shader

```
uniform float brightness;    // valeur constante par appel de dessin
varying float fragGrayness;  // interpolée pendant la rastérisation
void main() {
   const vec4 red = vec4(1, 0, 0, 1);
   const vec4 gray = vec4(0.5, 0.5, 0.5, 1);
   gl_FragColor = brightness * ( red * (1 – fragGrayness) +
                                 gray * fragGrayness );
}
```

Exemple de base:
Illumination diffus `n` dot `l`

Rappel: Illumination diffus **n** dot **l**

- Given a point with normal **n**, and a light direction **l**



- Direct illumination (*ignoring shadows*) is **n** dot **l**

Université de Montréal

- What do the shaders look like?

n dot l Vertex Shader

```
varying vec3 normal;  // output interpolated normal vector
void main() {
   gl_Position = gl_Vertex * gl_ModelViewProjection; // Transform pos
   normal = gl_Normal * gl_NormalMatrix; // Transform normal
}
```

n dot l Fragment Shader

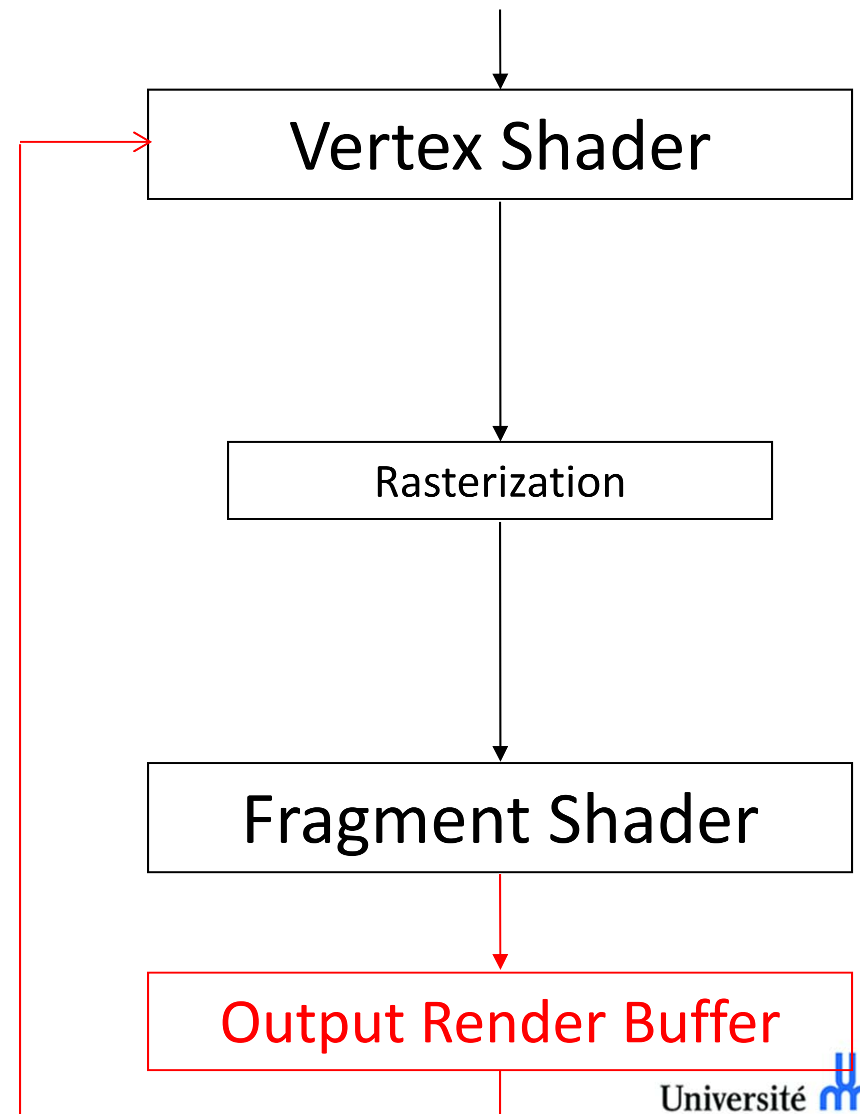
```
uniform vec3 light;   // input from the CPU using OpenGL
varying vec3 normal;  // output from vertex shader, interpolated
void main() {
   // float shade = light.x * normal.x + light.y * normal.y +
   //               light.z * normal.z;
   // gl_FragColor = vec4( shade, shade, shade, 1 );
   gl_FragColor = vec4( dot( light, normal ) );
}
```

Les pipelines de shaders *multi-pass*:

Comment coder des algorithmes d'illumination plus complexes

- Les algorithmes de shading complexes exigent des passes de rendu à plusieurs étapes

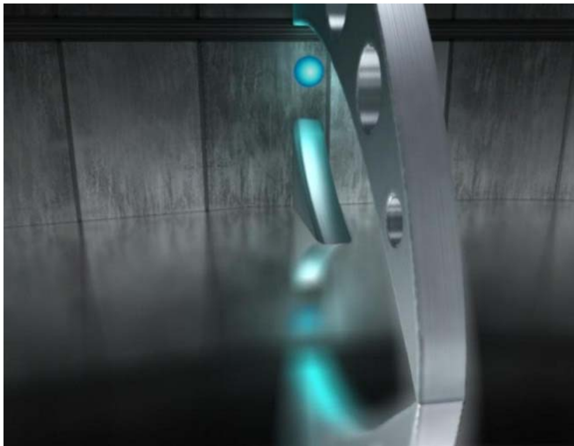
- On peut utiliser plusieurs pairs de shaders dans ce cas



Université de Montréal

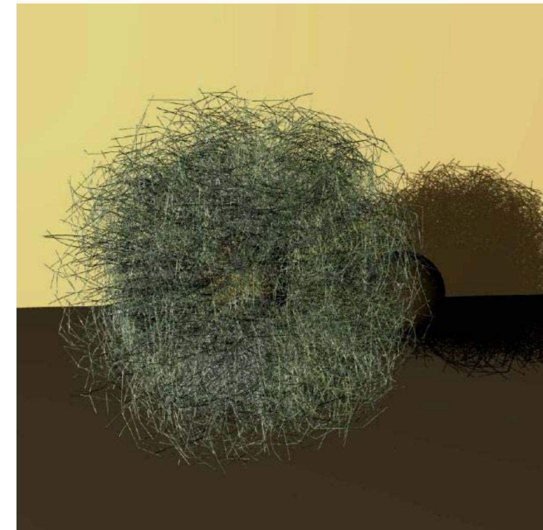
Les pipelines de shaders *multi-pass*

(exemples de [Lefohn09])



"Fast Summed-Area Table Generation and its Applications," Hensley et al. 2005

"Dynamic Ambient Occlusion and Indirect Lighting," Bunnell 2005

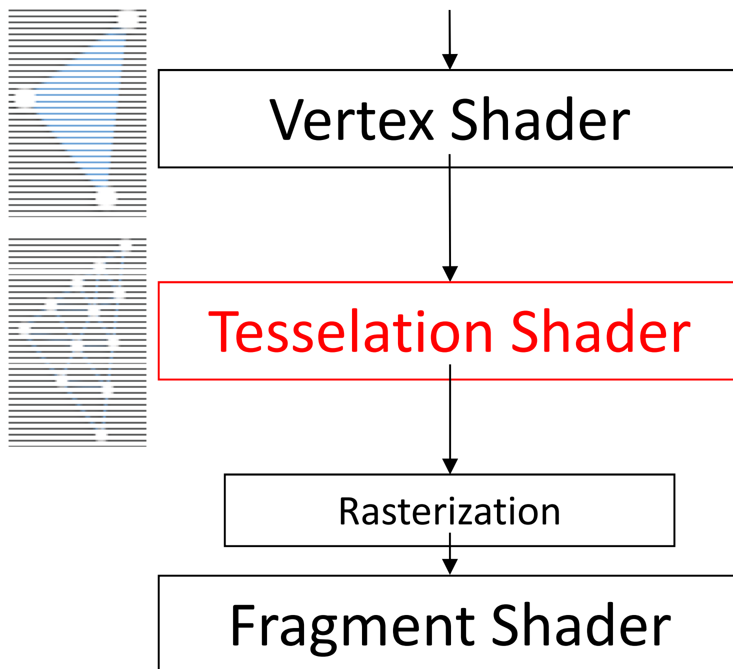


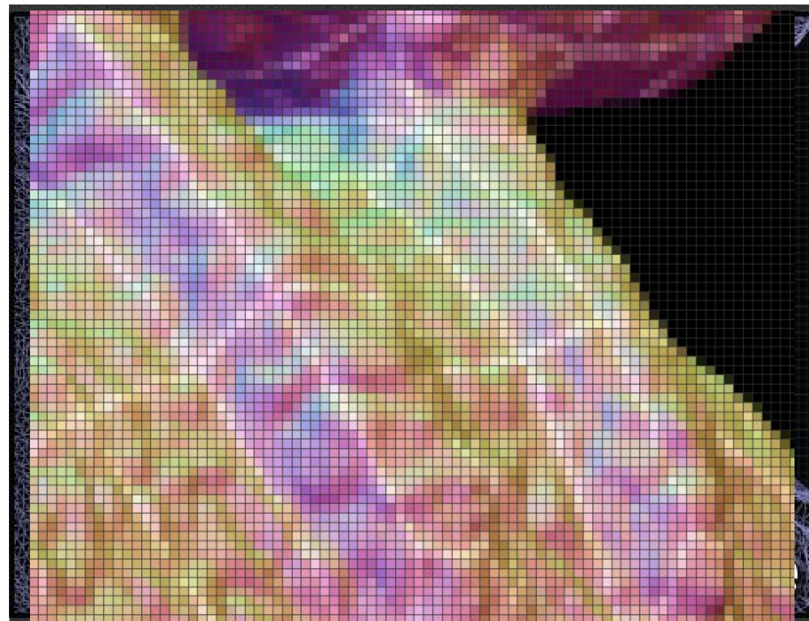
"Resolution Matched Shadow Maps," Lefohn et al. 2007

Université de Montréal

Qu'est-ce que j'ai ignoré…

- … les dernières 4 années du *spec* OpenGL:
  - *Geometry* et *Hull shaders*: permet à créer de la géométrie . Utile pour la tesselation dynamique





[Fatahalian 2009]

  - Compute shaders aussi

Université de Montréal