

M2-Images

Rendu Temps Réel - OpenGL 3 et geometry shaders

J.C. Iehl

April 11, 2012

Geometry shaders

- ▶ qu'est ce que c'est ?
- ▶ à quoi ça sert ?
- ▶ comment ça marche ?

Geometry shaders : qu'est ce que c'est ?

une étape *optionnelle* du pipeline :

- ▶ vertex shader responsable de transformer les sommets et leurs attributs,
- ▶ fragment shader responsable de calculer la couleur de chaque fragment.

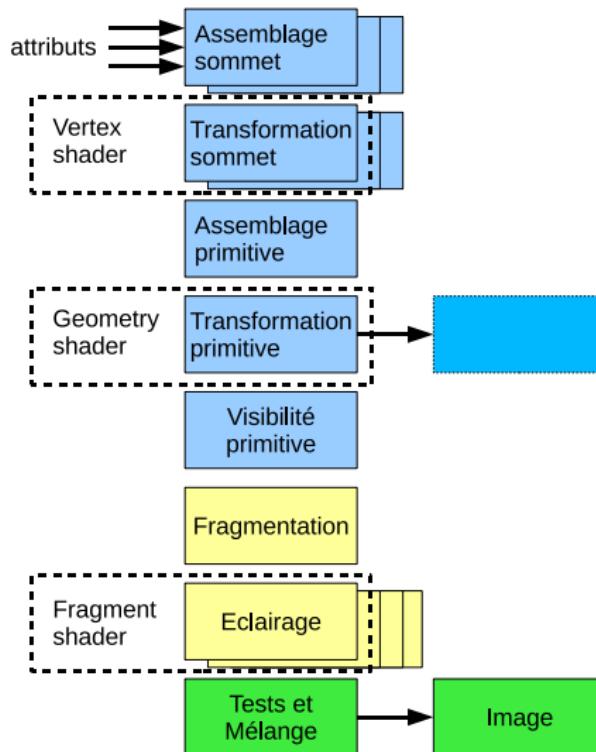
geometry shader :

- ▶ opération sur une primitive complète,
- ▶ peut accéder à tous les sommets de la primitive,
- ▶ peut créer de nouveaux sommets,
- ▶ peut changer le type de primitive,
- ▶ peut détruire une primitive.

Geometry shaders : où sont-ils ?

- ▶ exécutés une fois par primitive dessinée,
- ▶ après les vertex shaders,
- ▶ avant l'assemblage et l'élimination des primitives dos à la camera,
- ▶ et avant la rasterization / fragmentation des primitives,
- ▶ avant les fragment shaders.

Geometry shaders : ou sont-ils ?



Geometry shaders : à quoi ça sert ?

à assouplir le pipeline ?

- ▶ seul endroit où tous les sommets d'une primitive sont disponibles,
- ▶ et manipulables.

Geometry shaders : comment ça marche ?

comment ça marche ?

- ▶ déclarer le type de primitive en entrée (traitée),
- ▶ déclarer le type de primitive en sortie (produite),
- ▶ déclarer le nombre maximum de sommets produits.

accès aux sommets (en entrée) :

- ▶ à travers un tableau de sommets :
`in gl_PerVertex gl_in[],`
- ▶ la structure `gl_PerVertex` est pre-déclarée,
- ▶ il faut déclarer explicitement les tableaux correspondant aux `varying` du vertex shader.

Geometry shader : exemple

```
#version 330      // gl 3.3 core profile

/* declaration implicite
in gl_PerVertex
{
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
} gl_in[]; */

layout (triangles) in;
layout (triangle_strip) out;
layout (max_vertices= 3) out;

void main( void )
{}
```

que fait ce shader ?

utiliser `#version 330 compatibility`
pour accéder à l'api "historique" / pipeline fixe.

Geometry shader : exemple

```
#version 330      // gl 3.3 core profile, vertex shader

uniform mat4 mvp;
in vec4 position;

out vec3 couleur;
out vec3 normale;

void main( void )
{
    gl_Position= mvp * position;
    couleur= vec3(1.0, 0.0, 0.0);
    normale= vec3(0.0, 1.0, 0.0);
}

#version 330      // gl 3.3 core profile, geometry shader

in vec3 couleur[];
in vec3 normale[];

layout (triangles) in;
layout (triangle_strip) out;
layout (max_vertices= 3) out;

void main( void )
{
    // ...
}
```

Geometry shader : produire un sommet

produire un sommet :

- ▶ fonctionne comme un vertex shader,
- ▶ y compris les varyings, et gl_Position,
- ▶ EmitVertex() pour produire le sommet,
- ▶ tous les varying nécessaires à l'exécution du fragment shader doivent être définis pour le sommet, avant l'appel EmitVertex().

Geometry shader : produire une primitive

produire une primitive :

- ▶ produire tous les sommets de la primitive, puis,
- ▶ `EndPrimitive()`

attention :

- ▶ les primitives sont indexées de manière implicite, ce sont des *strips* : points, line_strip, triangle_strip,
- ▶ on peut "casser" la description d'un strip, avec un `EndPrimitive()` supplémentaire pour produire des primitives indépendantes.

Rappel : description indexée de primitives

utilisation de sommets partagés :

- ▶ indexation *explicite* des sommets : cf. `glDrawElements()`,
- ▶ indexation *implicite* des sommets, réutilisation du(des) sommets précédents pour définir la primitive :
`GL_LINE_STRIP`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`.

exemple : `GL_TRIANGLE_STRIP`

- ▶ abc, construit un triangle abc,
- ▶ abcd, construit 2 triangles : abc, cbd.

cf. **GLPG, ch3, geometric primitive types.**

Geometry shader : exemple complet

```
#version 330      // gl 3.3 core profile, geometry shader

layout (triangles) in;
layout (triangle_strip) out;
layout (max_vertices= 3) out;

void main( void )
{
    for(int i = 0; i < gl_in.length(); i++)
    {
        gl_Position= gl_in[i].gl_Position;
        // termine la description du sommet
        EmitVertex();
    }

    // termine la description de la primitive (triangle)
    EndPrimitive();
}
```

Geometry shader : exercice

dessiner la normale des triangles, au lieu de dessiner les triangles eux memes ...

Geometry shader : exercice

dessiner une boite englobante, à la place de la ligne `bbox.pMin`,
`bbox.pMax` ...

Gestion de scène et Geometry shader

instanciation :

- ▶ `glDrawElementsInstanced()`, dessine plusieurs fois le même objet indexé,
- ▶ `glDrawArraysInstanced()`, dessine plusieurs fois le même objet,
- ▶ comment introduire des différences entre les différentes instances ? (la position et l'orientation, par exemple ?)

Instanciation et Geometry shader

instanciation :

- ▶ dessine plusieurs copies du même objet ...
- ▶ les sommets de chaque copie sont traités par le pipeline graphique,
- ▶ comment paramétrier les copies / les instances ?

dans un shader :

- ▶ `gl_InstanceID` : numero de l'instance $\in [0 \ n)$
- ▶ déclarer un tableau de paramètres d'instance dans le shader :
`positions [gl_InstanceID]`
- ▶ mais : la taille des tableaux est limitée ... quelques Ko.
- ▶ autre chose ?

Instanciation et Geometry shader

autre chose ?

- ▶ calculer le paramètre de l'instance en fonction de `gl_InstanceID`,
- ▶ utiliser une texture au lieu d'un tableau dans le shader,
- ▶ (associer un buffer au tableau),
- ▶ déclarer un attribut dans le shader et utiliser un vertex buffer pour stocker sa valeur pour chaque instance.

`glVertexAttribDivisor(location, divisor) :`

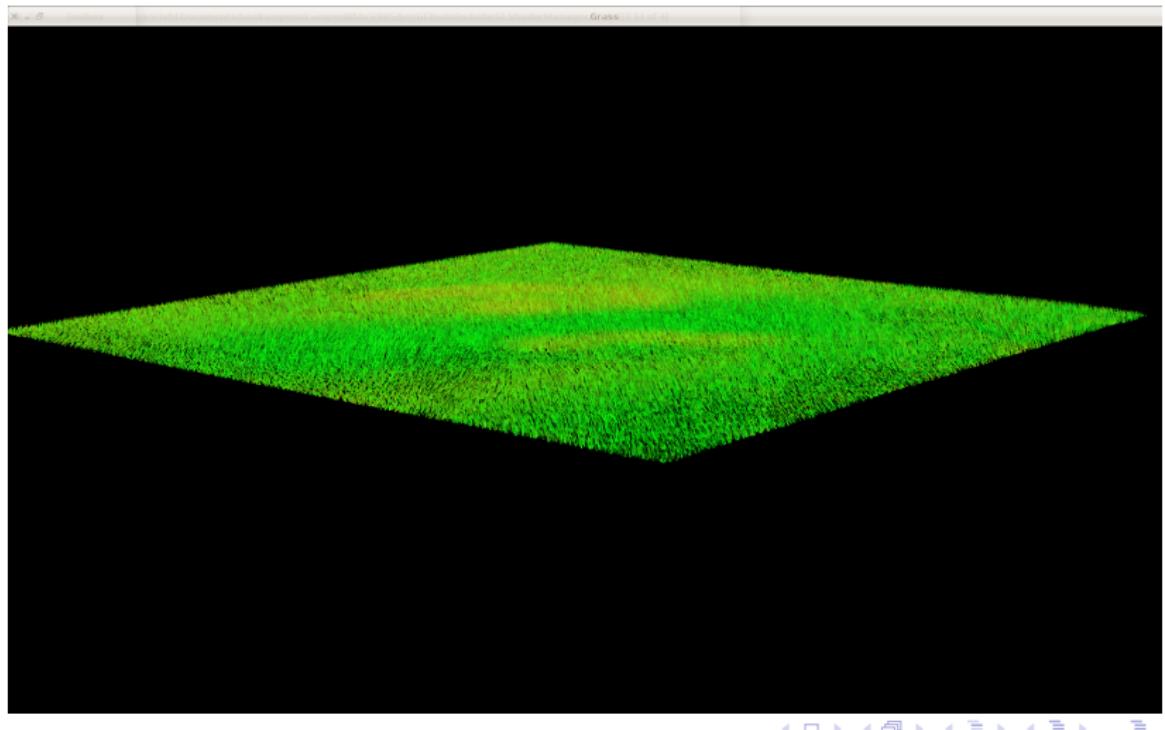
- ▶ `divisor == 1`,
chaque instance lira une nouvelle valeur dans le buffer,
- ▶ `divisor == 0`,
chaque sommet lit une nouvelle valeur dans le buffer.

Instanciation : exemple

afficher une prairie :

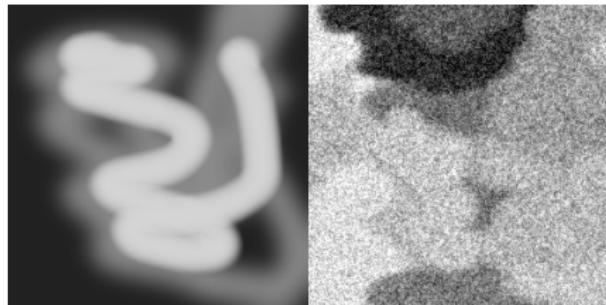
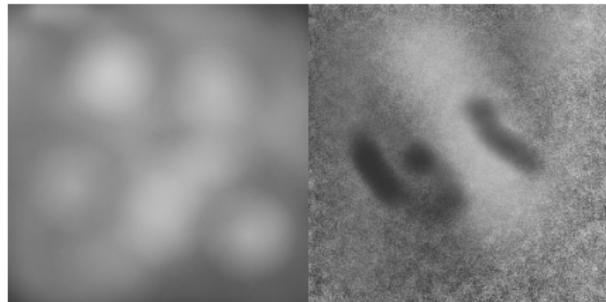
- ▶ trop lent : dessiner chaque brin d'herbe (n draws de 4 triangles),
- ▶ nettement plus efficace : 1 seul draw pour $4n$ triangles.

Instanciation : exemple, opengl super bible 5, ch12

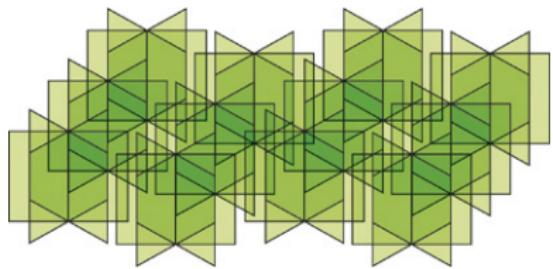
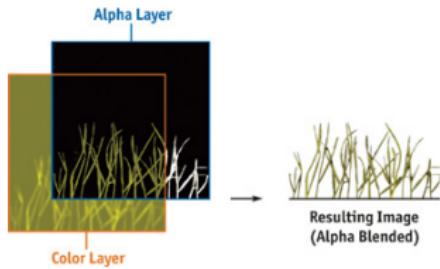
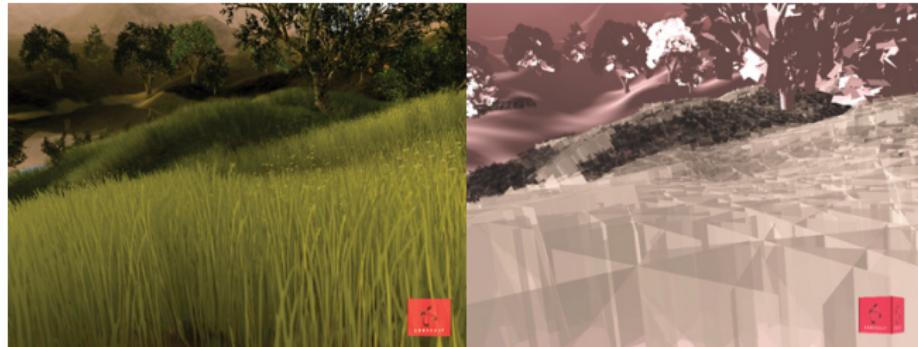


Instanciation : exemple

les données d'instances sont encodées dans plusieurs textures.



Instanciation : encore de l'herbe, gpu gems, ch7



Instanciation : exemple, starcraft2, blizzard



Instanciation : limites

limites :

- ▶ dessine / traite beaucoup de géométrie,
- ▶ trop (dans certains cas) ?
 (l'herbe derrière une colline)
- ▶ comment éliminer les instances non visibles ?
 (les primitives non visibles)

Instanciation : limites

éliminer les primitives non visibles :

- ▶ les "parties" non visibles sont éliminées :
- ▶ en fonction de leur orientation (back-face culling),
- ▶ en fonction de leur distance (z-buffer)
- ▶ mais : beaucoup d'instances dessinées,
- ▶ encore plus de primitives et de fragments !

éliminer les *instances* non visibles ?

Instanciation : éliminer les instances non visibles

tests de visibilité :

- ▶ visible par la camera (frustum culling),
- ▶ orienté vers la camera (back-face culling) ?,
- ▶ non caché par un autre objet (occlusion culling)

Instanciation : éliminer les instances non visibles

tests de visibilité :

- ▶ frustum culling, facile,
- ▶ back-face culling, pas vraiment applicable à une instance,
- ▶ occlusion culling, le plus efficace mais le moins direct.

idée:

construire un z-buffer hiérarchique pour éliminer un objet complet.

splinter cell : conviction, gdc 2010

rastergrid, blog