OpenGL avancé

Instances OpenGl

```
for(unsigned int i = 0; i < amount_of_models_to_draw; i++)
{
    DoSomePreparations(); // bind VAO, bind textures, set uniforms
etc.
    glDrawArrays(GL_TRIANGLES, 0, amount_of_vertices);
}
```

=> les performances chutes en raison du nb d’appels

- **Préparations avant de pourvoir le rendu les données des sommets**
  - Spécifier les buffers stockant les données où trouver les attributs de sommets
  - **Ceci se fait via le bus CPU / GPU relativement lent**
- **Même si le rendu des sommet est très rapide, envoyer les commande de rendu au GPU ralenti considérablement le rendu**
  - (voir dans quel mesure => tests

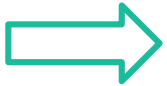
Solution

- **Envoyer les données de rendu une seule fois (1 seul objet)**
  - Une seule instance d’un objet
- **Et dire à Opengl de faire de multiples rendus de l’objet**
  - En 1 SEUL APPEL
  - en utilisant les données de l’objet

=> instanciation

Comment

glDrawArrays
ou
glDrawElements



GlDrawArraysInstanced
ou
glDrawElementsInstanced

- Ces fonctions utilisent un paramètre supplémentaire :
  - le nb d'instances que l'on souhaite rendre
- Les données sont envoyées au GPU une seule fois (**1 seul « objet »**)
- On « dit » au GPU de faire le rendu **de toutes ces instances en un seul appel**
- **Le GPU fait le rendu sans avoir à communiquer continuellement avec le CPU**

Problème = comment faire des rendus différents

- **Solution simple :**
  - Dans les shader pour différencier les instances on dispose d'un identifiant : la variable **gl_InstanceID**
  - La 1ere instance à un gl_InstanceID = 0
  - Il est incrémenté pour les suivantes
  - Exemple d'un rang d'oignons.

```
// vertex shader
...
newpos = (position + vec3(1.,0.,0) * pas  *
gl_InstanceID;
```

Problème = comment faire des rendus différents

- **Solution 2 : données « uniform »**
  - On envoie des données uniformes pour définir la particularité de chaque instance
  - Exemple :
    - Tableau uniform de vec2 pour définir la translation de chaque instance.

=> pb limitation par la taille des variable uniform

=> passer par les instances array

Problème = comment faire des rendus différents

- **Solution 3 : Instance Array**
  - principe utilisation
    - Sont définis comme un vertex attribute
    - permettent de stocker plus de données
    - Les données sont **affectées par instances** au lieu de vertex
  - Pas la peine d'utliser gl_InstanceID

```
// vertex shader
layout (location = 0) in vec2 aPos;

layout (location = 2) in vec2 aOffset;
out vec3 fColor;
void main()
{
    gl_Position = vec4(aPos + aOffset, 0.0, 1.0);
}
```

Instance Array : Mise en oeuvre

- **un « instanced array » est un « vertex attribute » => il faut :**
  - stocker les données dans un « vertex buffer object »
  - configurer son « attribute pointer ».
- **Exemple : stocker les translations:**
  - Créer le buffer et y stocker les données

```
// code C++
unsigned int instanceVBO;
glGenBuffers(1, &instanceVBO);
glBindBuffer(GL_ARRAY_BUFFER, instanceVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec2) * 100, &translations[0], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Instance Array : Mise en oeuvre

- Et définir l'« attribute pointer » + **glVertexAttribDivisor**

```
// code C++
GlEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, instanceVBO);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glVertexAttribDivisor(2, 1);
```

https://perso.univ-lyon1.fr/jean-claude.iehl/Public/educ/M2PROIMA/OpenGL.html

Instance Array : Mise en oeuvre

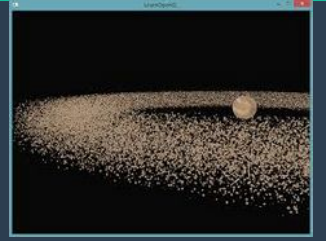
- **Focus sur glVertexAttribDivisor(GLuint index, GLuint divisor);**
  - indique à OpenGL quand mettre à jour le contenu de l'attribut de sommet avec le prochain élément.
    - index = l'attribut de sommets pour ((layout(location=xx)) (on met le même que glVertexAttribPointer)
    - divisor = le diviseur d'attribut = combien d'instance faut-il passer pour prendre l'attribut suivant
  - Par défaut, divisor = 0,
    - indique à OpenGL de mettre à jour l'attribut de sommet à chaque itération dans le vertex shader
    - i.e. à chaque sommet
  - divisor =1,
    - mettre à jour le contenu de l'attribut de sommet pour à chaque nouvelle instance
  - divisor = 2,
    - le contenu est mis à jour toutes les deux instances
  - ...

https://perso.univ-lyon1.fr/jean-claude.iehl/Public/educ/M2PROIMA/OpenGL.html

Exemple de départ

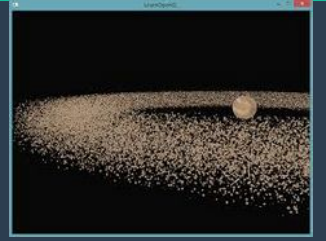
- **Comment faire ?**
- **Hypothèses :**
  - On dispose
    - d’une scène et de la position de la planète
    - 1 maillage + texture(s) pour astéroïde

Exemple de départ



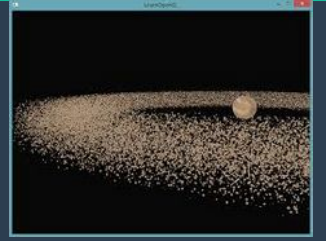
- **On fait comme si on visualiser un seul astéroïde**
  - Charger le maillage (position + normale + coord. Texture)
  - Créer un VBO (position + normale + coord texture) (ou plusieurs au choix)
  - Définir la localisation de l’astéroïde / la planète

Exemple de départ



- **Intances multiples**
- => **glDrawElementsInstanced**
- Solution 1 :
  - Dans le vertex shader on utilise **gl_InstanceID**
  - Définir la localisation aléatoire de l'astéroïde / la planète
    - => utiliser une fonction de générateur de nombre aléatoire (random)
    - avec en input (ou seed) **gl_InstanceID** (ou un multiple)
    - Theta= rand(), r = rmin + rand() * amplitude_anneau ;
    - PositionDansReperePlanete = vec3(r * cos(tetha), r*sin(theta),0) ;

Variable aléatoire / Random / bruit



- **Random en GLSL**
  - float noise1(genType x);
  - vec2 noise2(genType x);
  - vec3 noise3(genType x);
  - vec4 noise4(genType x);

NE FONCTIONNENT PAS (sur ma carte NVIDIA)

  - Retourne des valeurs d'une fonction a stochastique (bruit), scalaire ou vectorielle,

    en fonction de la valeur d'entrée x.
  - Les valeurs retournées sont similaire à des valeurs aléatoires mais ne le sont pas exactement.

Variable aléatoire / Random / bruit

- **On fait à la mimine : par exemple**

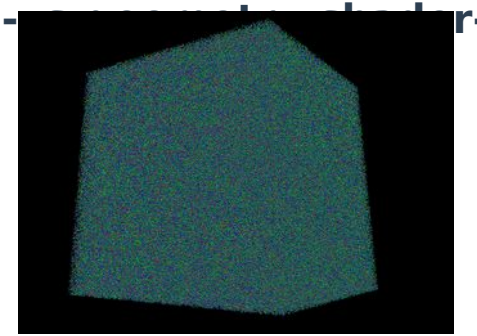
```
float rand(float xx)
{
   float x0 = floor(xx);
   float x1 = x0+1;
   float v0 = fract(sin (x0*.014686)*31718.927+x0);
   float v1 = fract(sin (x1*.014686)*31718.927+x1);

      return (v0*(1-fract(xx))+v1*(fract(xx)))*2-1*sin(xx);
}
```

Ou encore https://gist.github.com/patriciogonzalezvivo/670c22f3966e662d2f83

- **Performance à voir après goemetry shader**

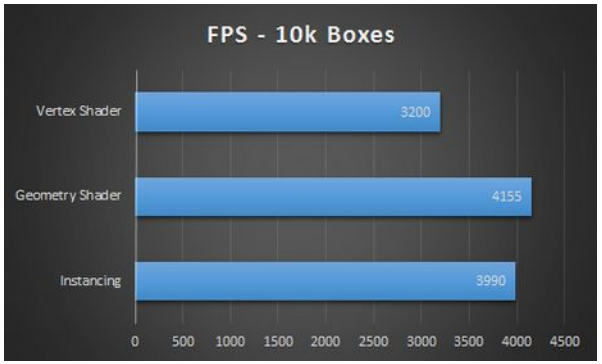
Instanciation vs geometry shader

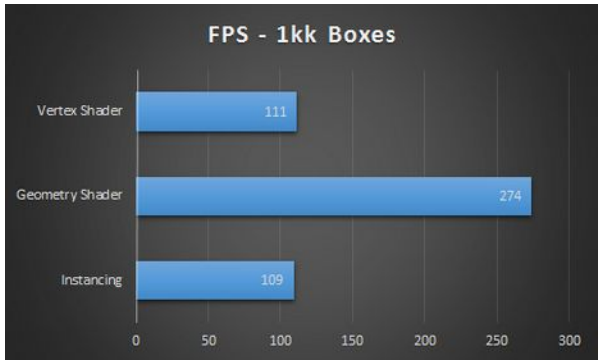
- **Instancing is not always fast if the individual meshes are very simple, like quads are. It depends on the hardware but I would avoid it.**
-
- **Exemple de comparaison**
  https://nbertoa.wordpress.com/2016/02/02/instancing-[illegible]r-vs-vertex-shader/
  - Instancing Technique
  - Geometry Shader Technique
  - Vertex Shader Technique

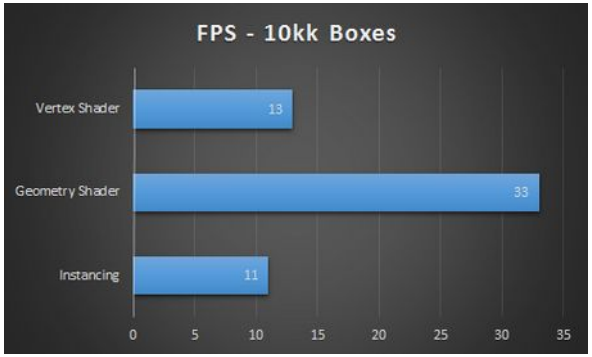


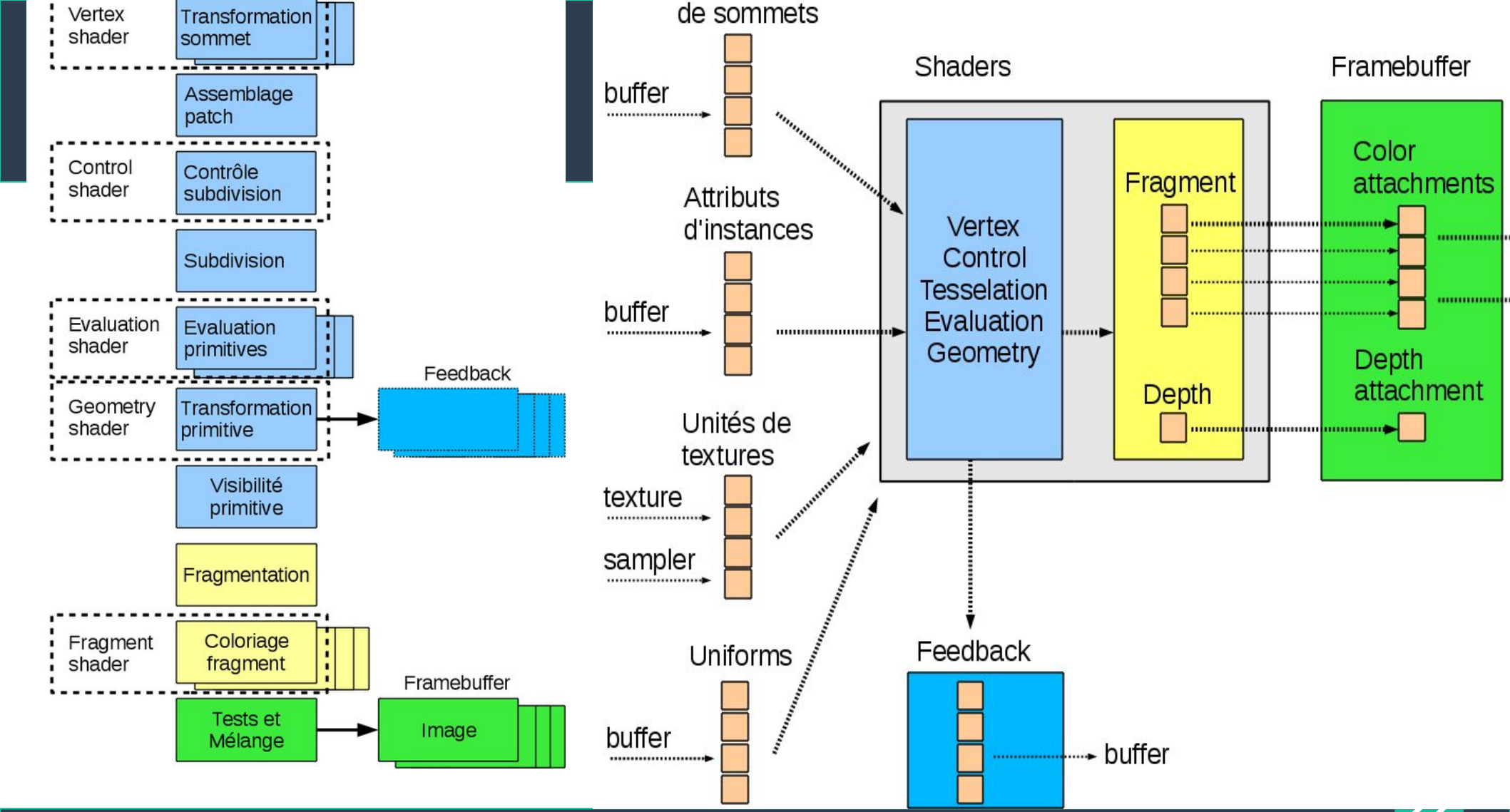
- **Instancing Technique**
  - Vertex buffer: 8 vertices representing box vertices positions
  - Index buffer: 36 indices representing indices to build 12 triangles
  - Instancing buffer: NUM_BOXES direction vectors (float3) that represent direction to translate each box vertex.
  -

- **Geometry Shader Technique**
  - Vertex buffer: NUM_CUBES vertices representing box center position.
- **Vertex Shader Technique**
  - Vertex buffer: It contains 8 vertices per box(position).
  - Index buffer: It contains 36 indices per box, to build 12 triangles per box
  -


FPS - 10k Boxes
Vertex Shader
Geometry Shader
Instancing
3200
4155
3990
0
500
1000
1500
2000
2500
3000
3500
4000
4500


FPS - 1kk Boxes
Vertex Shader
Geometry Shader
Instancing
111
274
109
0
50
100
150
200
250
300


FPS - 10kk Boxes
Vertex Shader
Geometry Shader
Instancing
13
33
11
0
5
10
15
20
25
30
35


Vertex shader
Transformation sommet
Assemblage patch
Control shader
Contrôle subdivision
Subdivision
Evaluation shader
Evaluation primitives
Geometry shader
Transformation primitive
Feedback
Visibilité primitive
Fragmentation
Fragment shader
Coloriage fragment
Framebuffer
Tests et Mélange
Image
de sommets
buffer
Shaders
Framebuffer
Attributs d'instances
buffer
Vertex
Control
Tesselation
Evaluation
Geometry
Fragment
Color attachments
Depth
Depth attachment
Unités de textures
texture
sampler
Uniforms
buffer
Feedback
buffer