http://tiny.cc/ift3355

IFT 3355: INFOGRAPHIE

SHADERS, OPENGL, JS

Livre de référence: **G:**Appendix A*

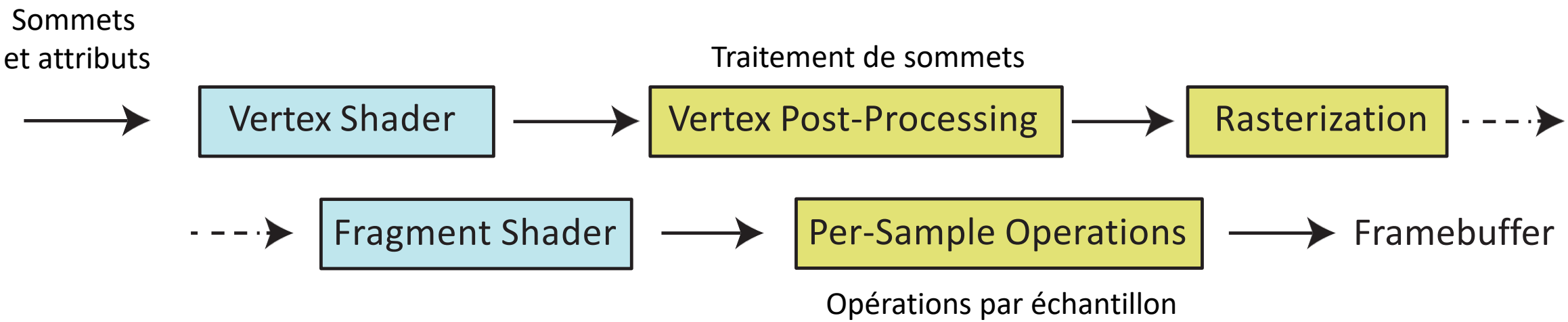
(pas la même version d'OpenGL)

Mikhail Bessmeltsev

DEVOIR 1

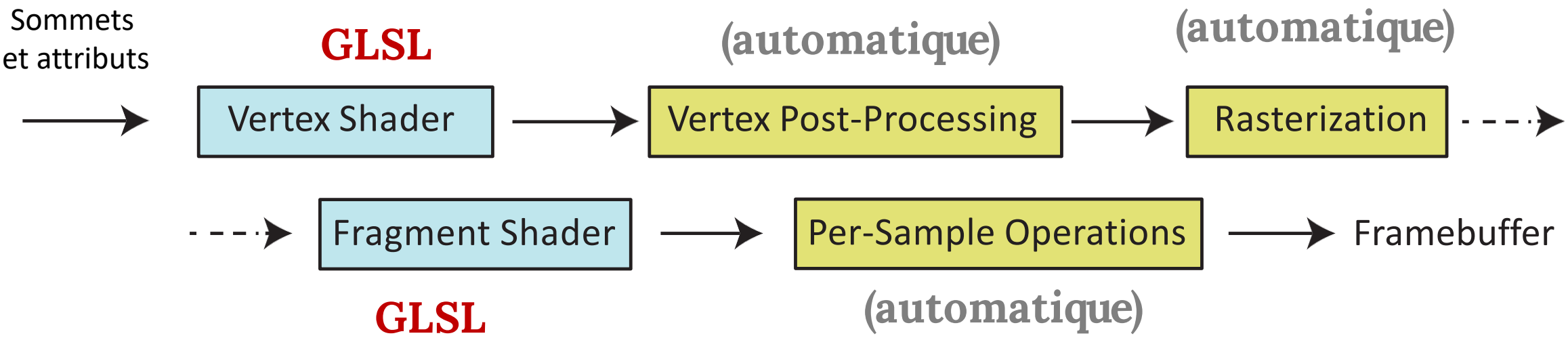
- Ne paniquez pas!
- Code de base fonctionne?
- Aujourd’hui vous allez comprendre bien plus

OPENGL PIPELINE DE RENDU



OPENGL PIPELINE DE RENDU

Javascript
+ Three.JS



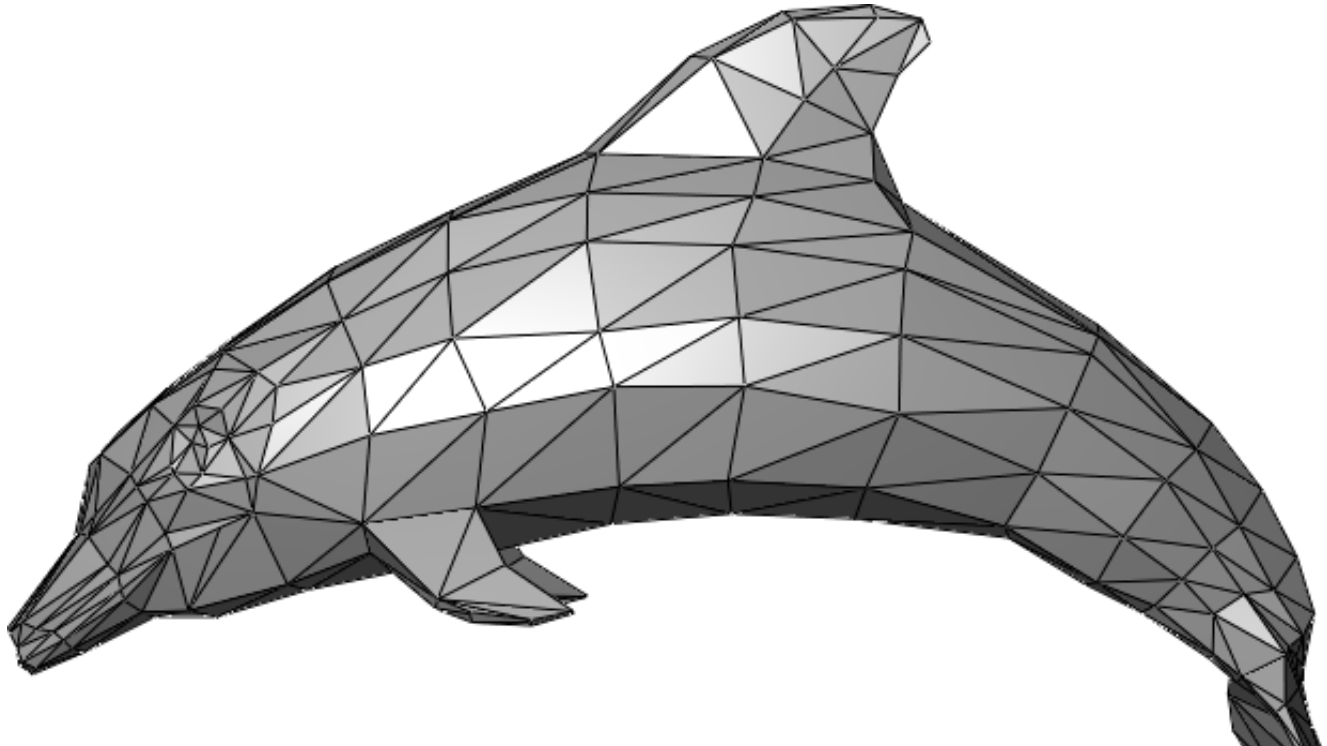
THREE.JS

- Une bibliothèque de haut niveau pour Javascript
- Utilise WebGL pour le rendu

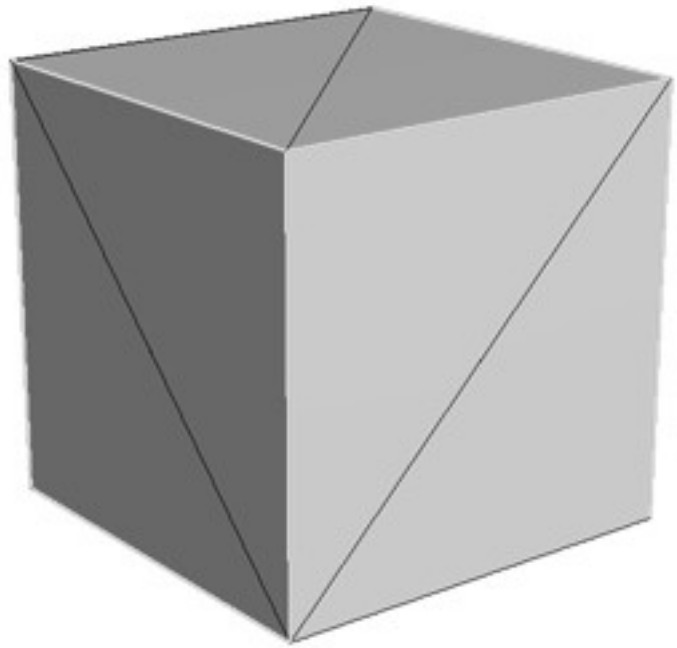
- Contient les objets: **Scene**, **Mesh, Camera**
- **Scene** est hiérarchique
- **Mesh** a la géométrie et les propriétés matérielles
- **Camera** est utilisée pour le rendu

LA GÉOMÉTRIE

- Les maillages de triangles
  - La liste de sommets
  - Les triangles sont définis comme {vertex_index1, vertex_index2, vertex_index3}



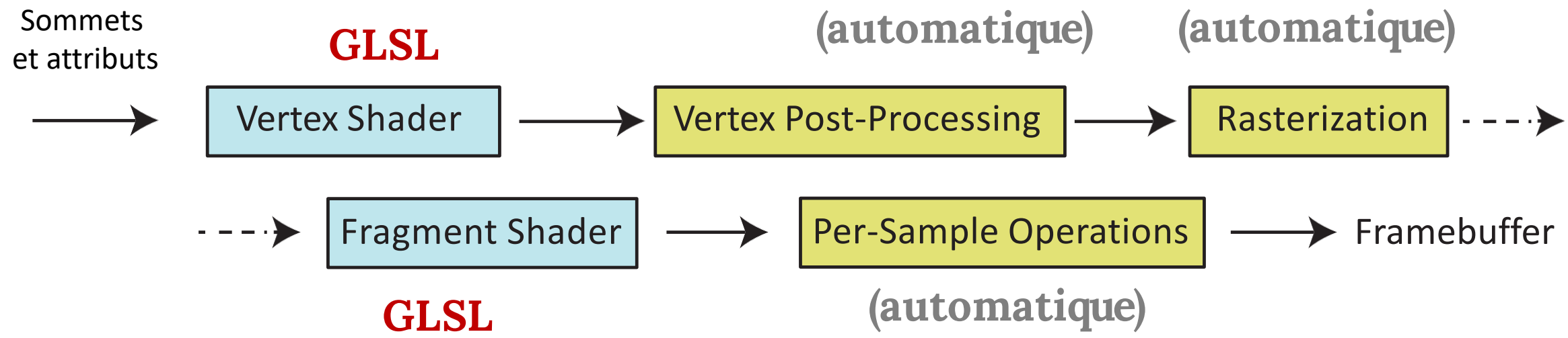
```
var verticesOfCube = [
    -1,-1,-1,    1,-1,-1,    1, 1,-1,    -1, 1,-1,
    -1,-1, 1,    1,-1, 1,    1, 1, 1,    -1, 1, 1,
];
var indicesOfFaces = [
    2,1,0,    0,3,2,
    0,4,7,    7,3,0,
    0,1,5,    5,4,0,
    1,2,6,    6,5,1,
    2,3,7,    7,6,2,
    4,5,6,    6,7,4
];
var geometry = new THREE.PolyhedronGeometry(
verticesOfCube, indicesOfFaces, 6, 2 );
```



GÉOMÉTRIE (JAVASCRIPT/THRE.JS)

OPENGL PIPELINE DE RENDU

Javascript
+ Three.JS



GLSL

- Un langage d'OpenGL pour le shading
- Utilisé par Fragment et Vertex shaders
- Plusieurs choses utiles:
  - vec3, vec4, dvec4, mat4, sampler2D
  - mat*vec, mat*mat
  - Réfléchir, réfracter
  - vec3 v(a.xy, 1)

VERTEX SHADER

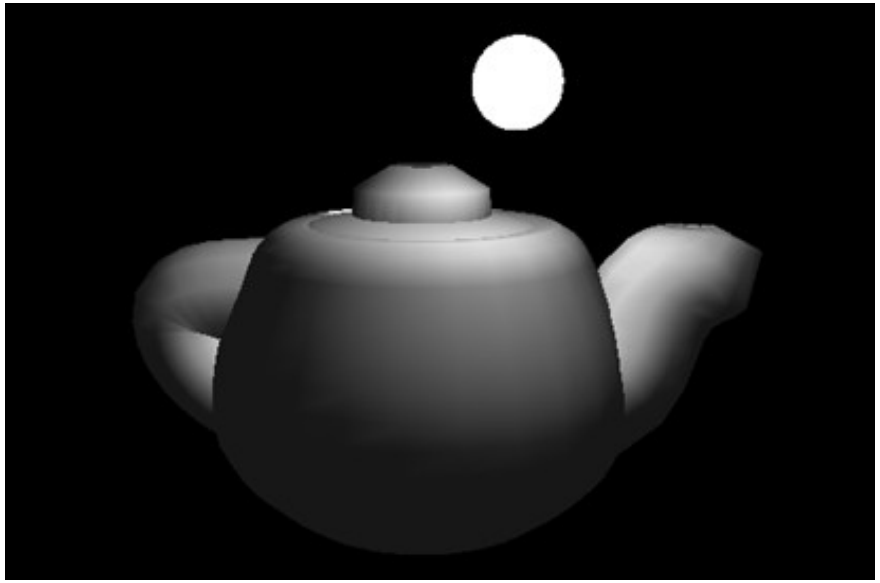
- VS fonctionne par chaque sommet séparément
- Par défaut, aucune notion de connectivité

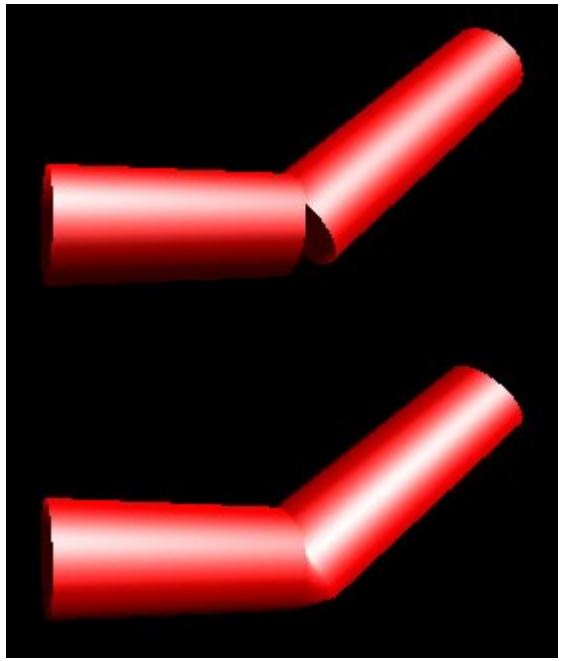
- Les données: les coordonnées dans le système de coordonnées de l'objet
- Son objectif principal est de définir **gl_Position**

Les coordonnées de l'objet → Les coordonnées du monde → Les coordonnées de VUE

VERTEX SHADER

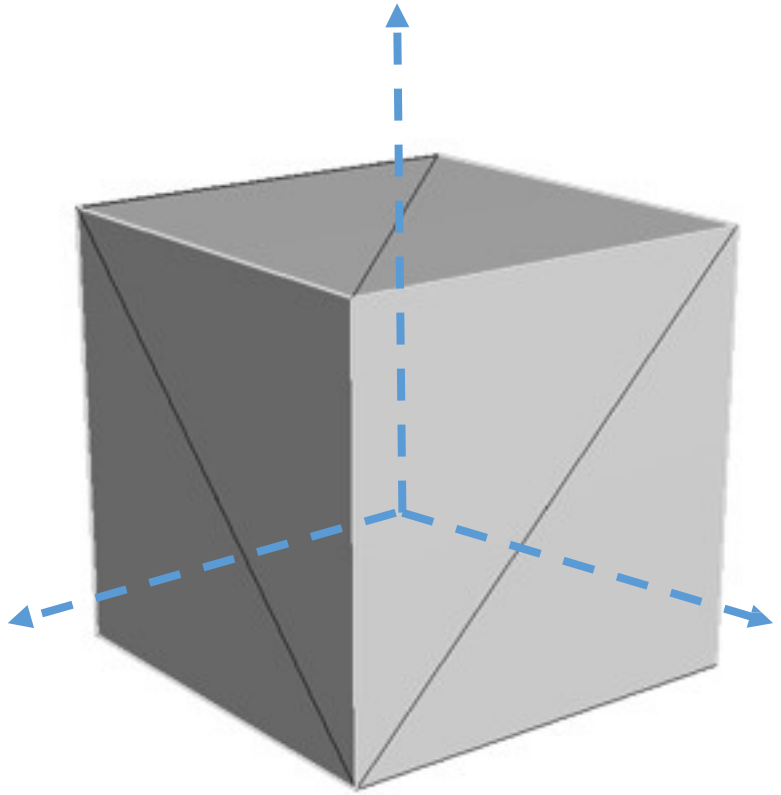
- En plus de convertir dans le système de coordonnées de vue
- On peut faire n'importe quoi avec les coordonnées (ou autres attributs)
  - e.g. déformer les sommets
  - e.g. skinning!



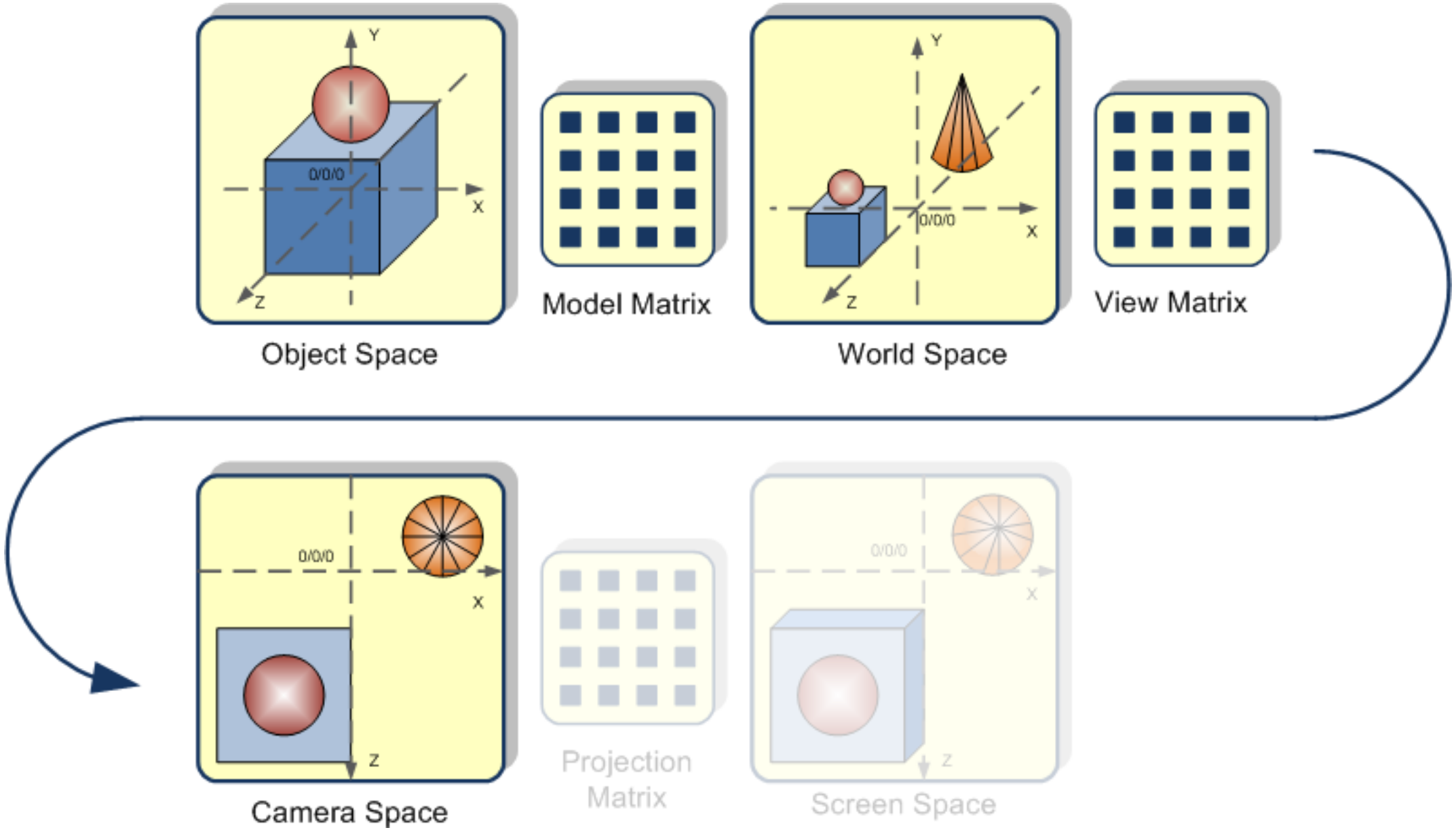


[courtesy NVIDIA]

```
var verticesOfCube = [
    -1,-1,-1,    1,-1,-1,    1, 1,-1,    -1, 1,-1,
    -1,-1, 1,    1,-1, 1,    1, 1, 1,    -1, 1, 1,
];
var indicesOfFaces = [
    2,1,0,    0,3,2,
    0,4,7,    7,3,0,
    0,1,5,    5,4,0,
    1,2,6,    6,5,1,
    2,3,7,    7,6,2,
    4,5,6,    6,7,4
];
var geometry = new THREE.PolyhedronGeometry(
verticesOfCube, indicesOfFaces, 6, 2 );
```



GÉOMETRIE (JAVASCRIPT/THRE.JS)



APERÇU DES TRANSFORMATIONS

- Toutes les transformations sont faites par des matrices 4x4:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix}$$

APERÇU DES TRANSFORMATIONS

- Toutes les transformations sont faites par des matrices 4x4:

Les coordonnées transformées

Les coordonnées du point

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix}$$

La matrice de transformation

APERÇU DES TRANSFORMATIONS

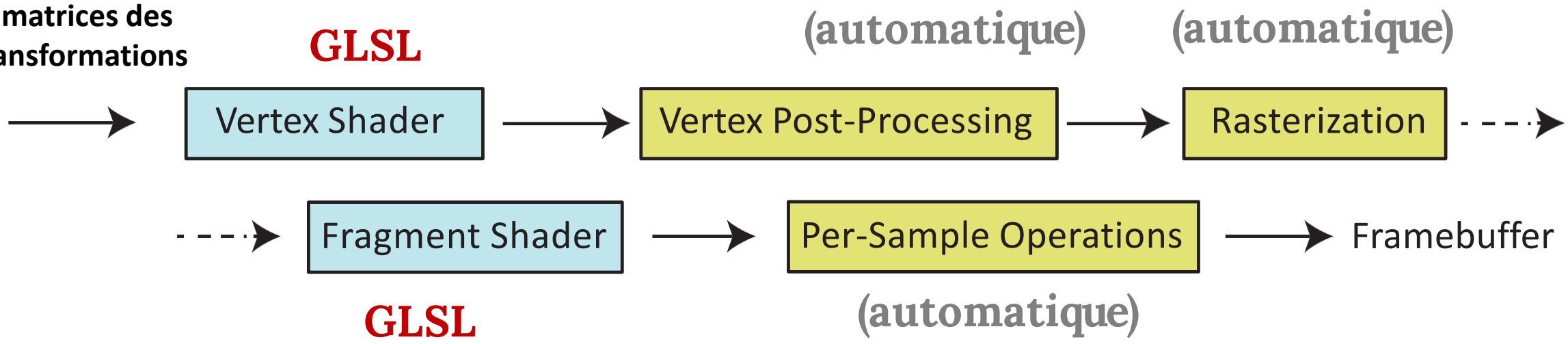
- Que fait cette transformation?

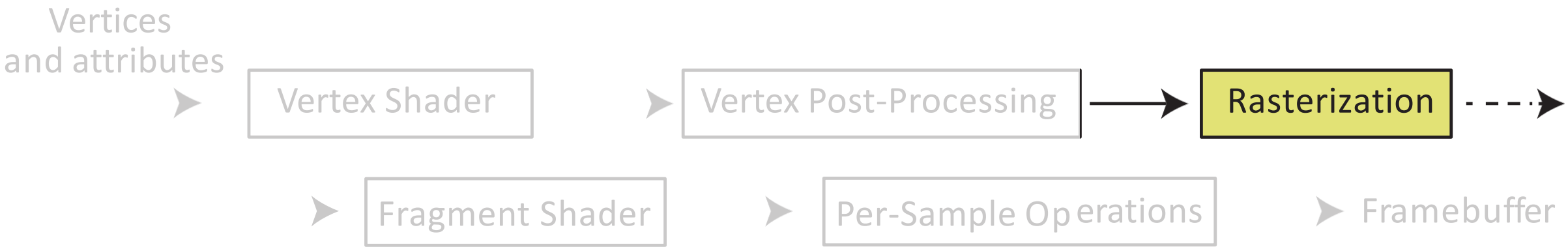
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} x \\ 2y \\ 3z \\ 1.0 \end{bmatrix}$$

OPENGL PIPELINE DE RENDU

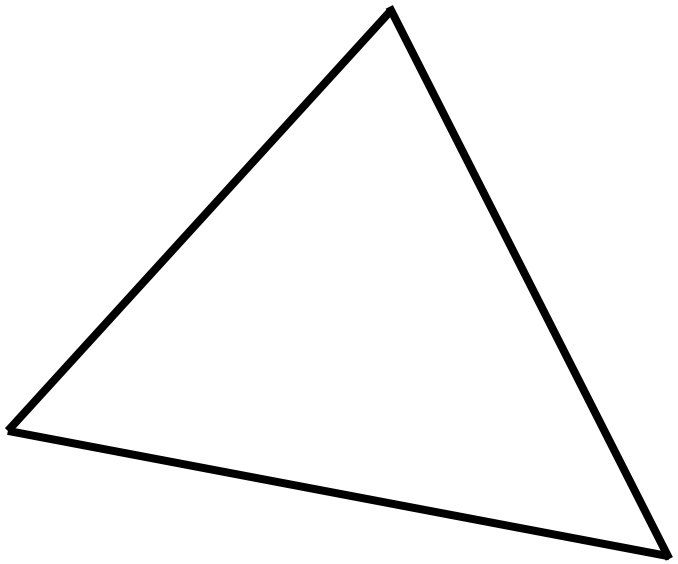
Javascript
+ Three.JS

Sommets
et attributs
+ matrices des transformations

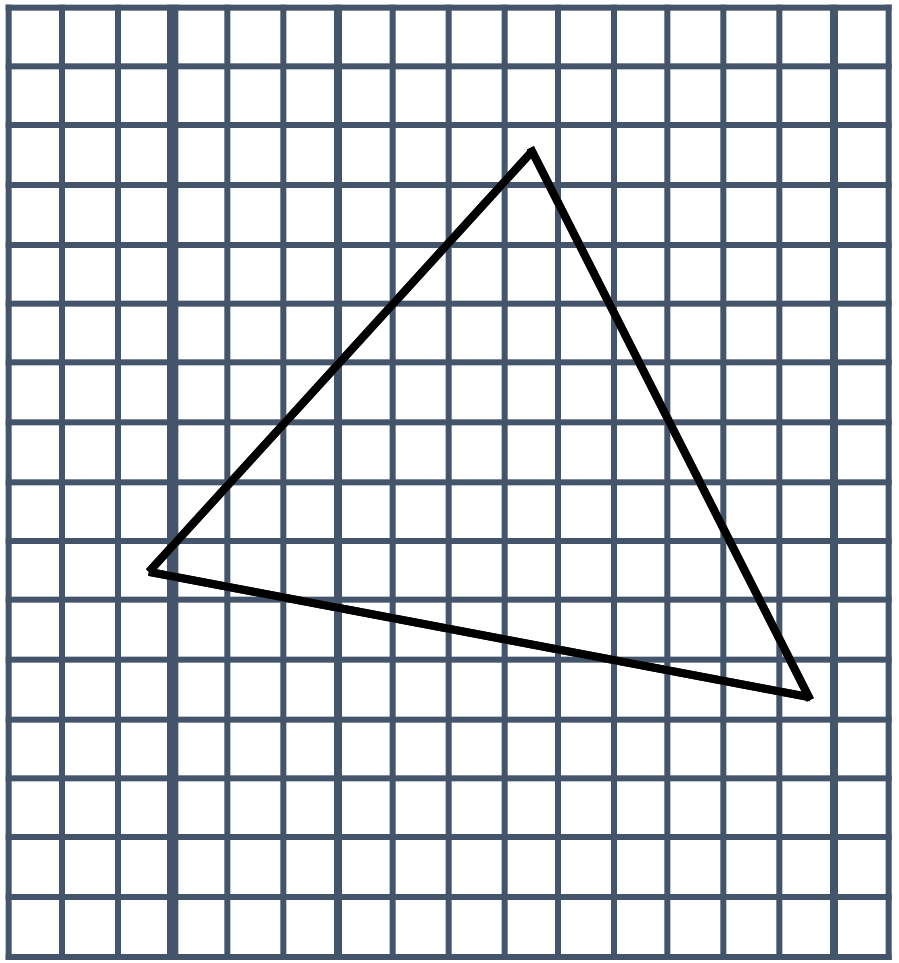



Vertices
and attributes
Vertex Shader
Vertex Post-Processing
Rasterization
Fragment Shader
Per-Sample Operations
Framebuffer

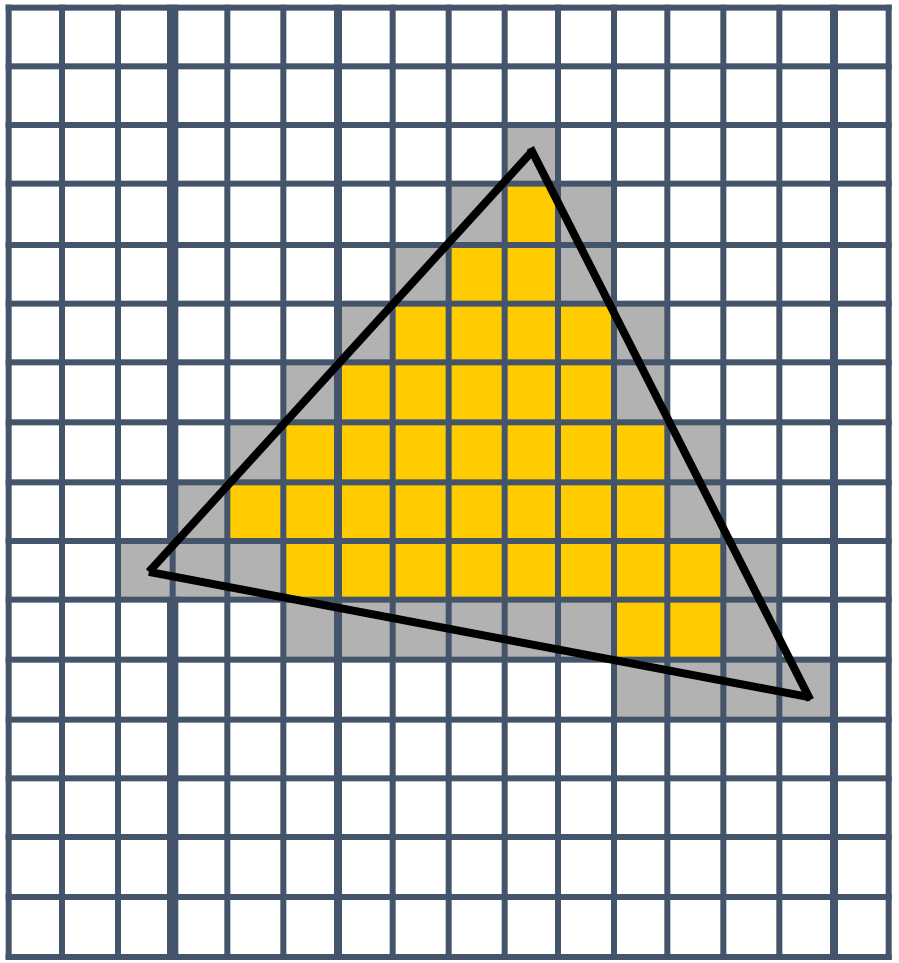
VUE DE LA CAMÉRA



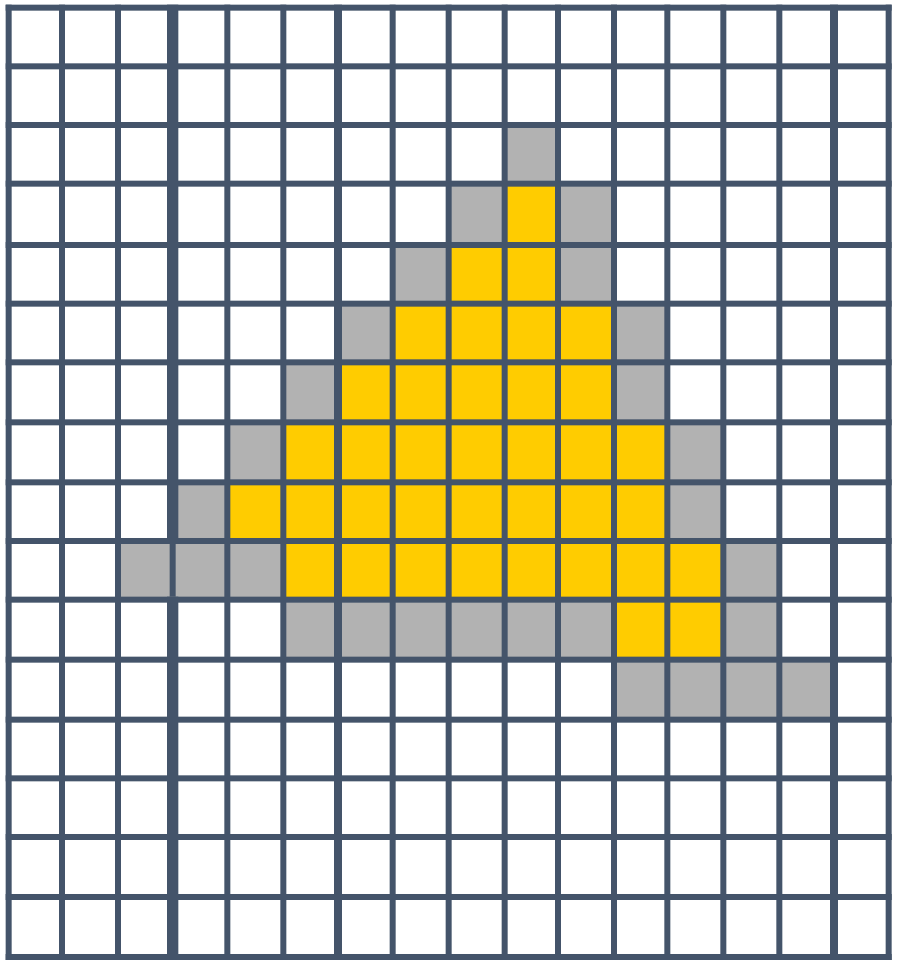
RASTERIZATION



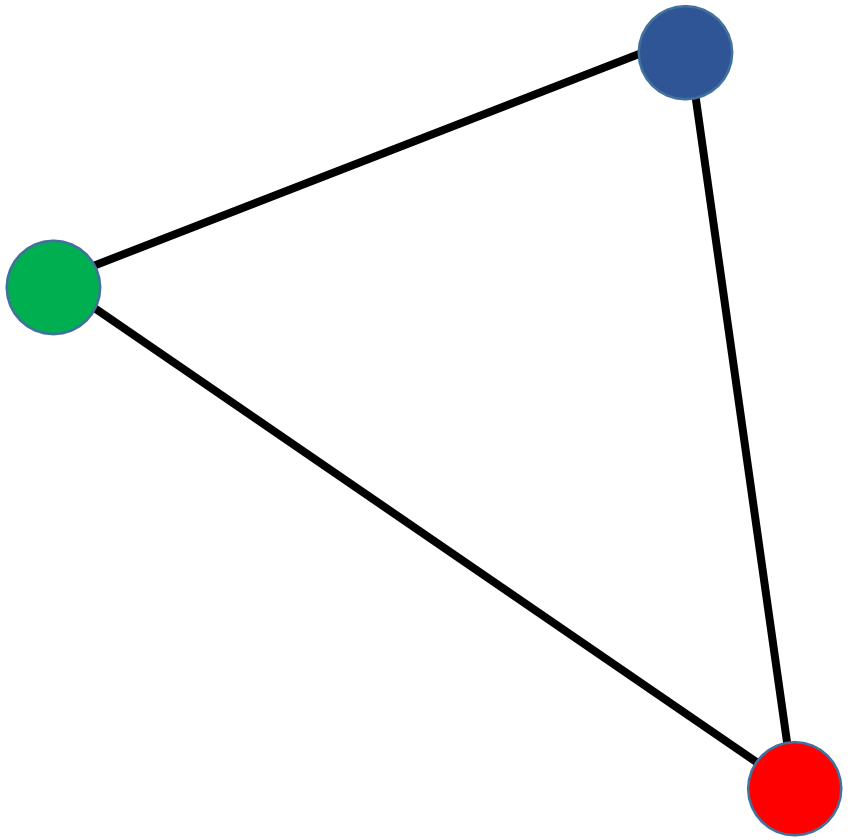
RASTERIZATION



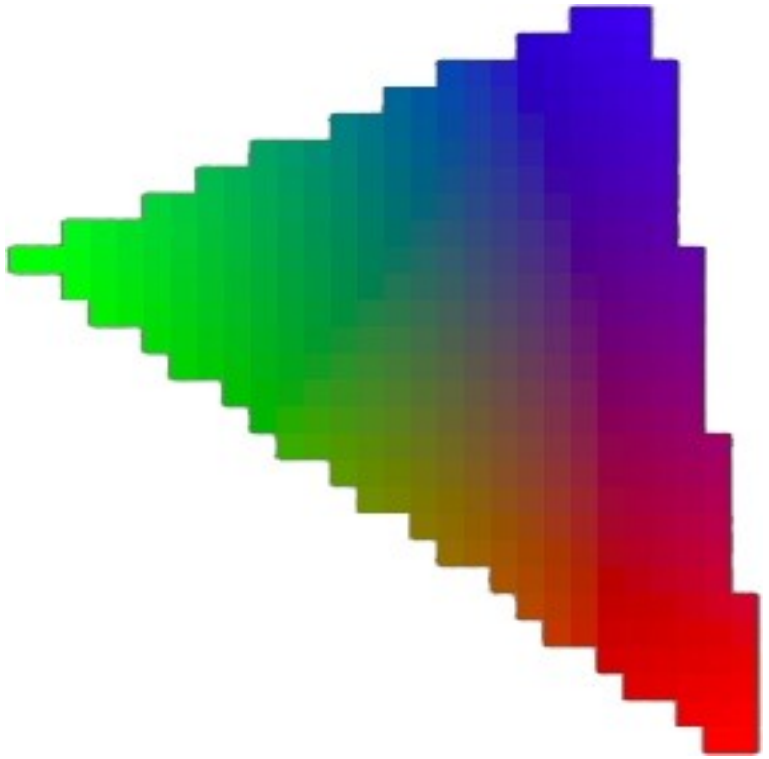
RASTERIZATION



RASTERIZATION - INTERPOLATION

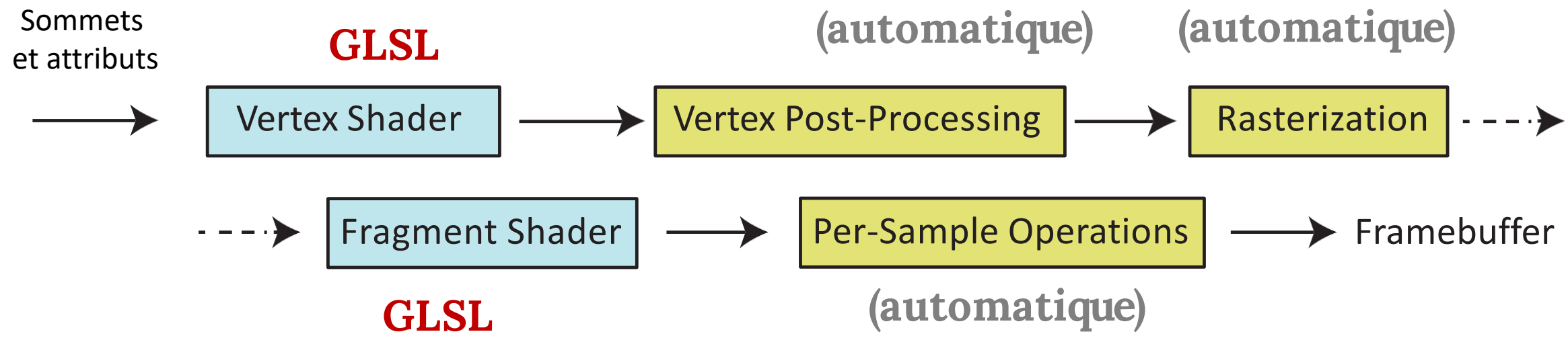


RASTERIZATION - INTERPOLATION



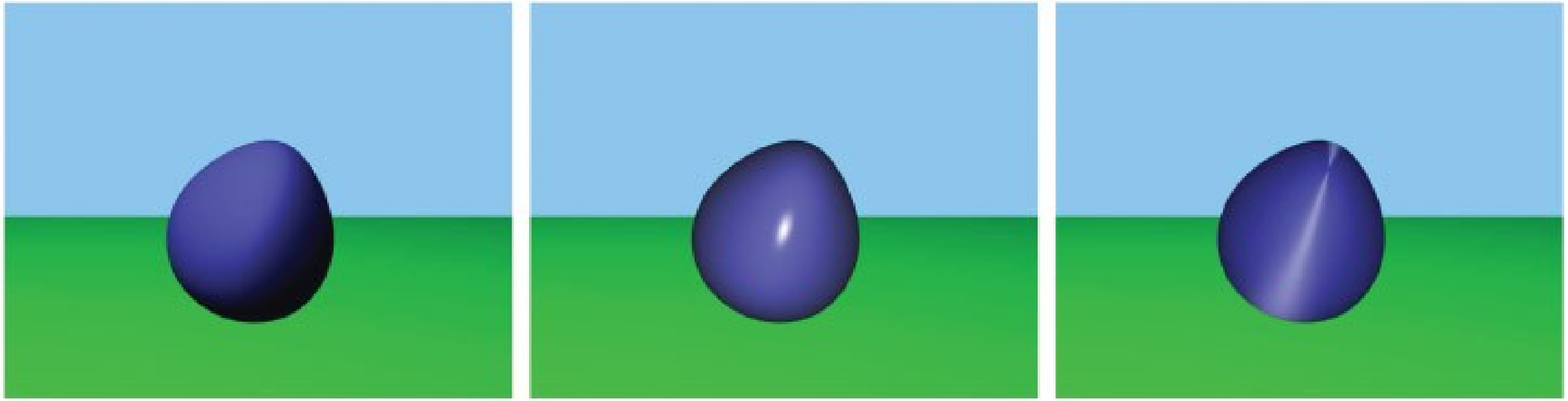
OPENGL PIPELINE DE RENDU

Javascript
+ Three.JS



FRAGMENT SHADER

- Fragment = les données pour affichage du pixel
- A gl_FragCoord – les coordonnées de l'affichage
- Peut définir la couleur!



FRAGMENT SHADER

- Tâches communes
  - texture mapping
  - Éclairage et shading par pixel

- Synonyme de Pixel Shader

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
}
```

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
}
```

Défini par Three.JS

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
}
```

Défini par Three.JS

$$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$$

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

VERTEX SHADER MINIMAL

```
void main()
{
    // Transformer le sommet
    gl_Position = modelViewMatrix * position;
```

$$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$$

Le système de coordonnées de la vue

Défini par Three.JS

FRAGMENT SHADER MINIMAL

```
void main()
{
    // Définir chaque pixel en rouge
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Rouge Vert Bleu Alpha

VERTEX SHADER – EXEMPLE 2

```
uniform float uVertexScale;
attribute vec3 vColor;
varying vec3 fColor;



void main() {
 gl_Position = vec4(position.x * uVertexScale, position.y, 0.0,1.0);
 fColor = vColor;
}
```

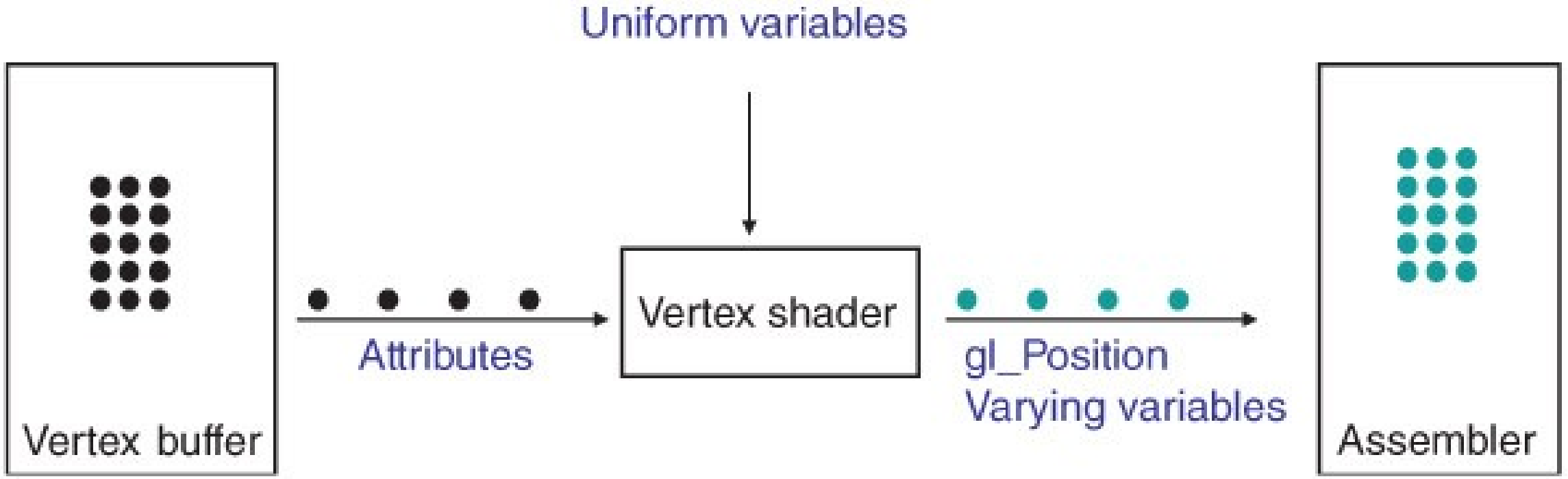
LES CONCEPTS

- uniform
  - La même chose pour tous les sommets
- varying
  - Calculé pour chaque sommet, puis interpolé automatiquement pour les fragments
- attribute
  - N'importe quelle valeur par sommet
  - Disponible uniquement dans Vertex Shader

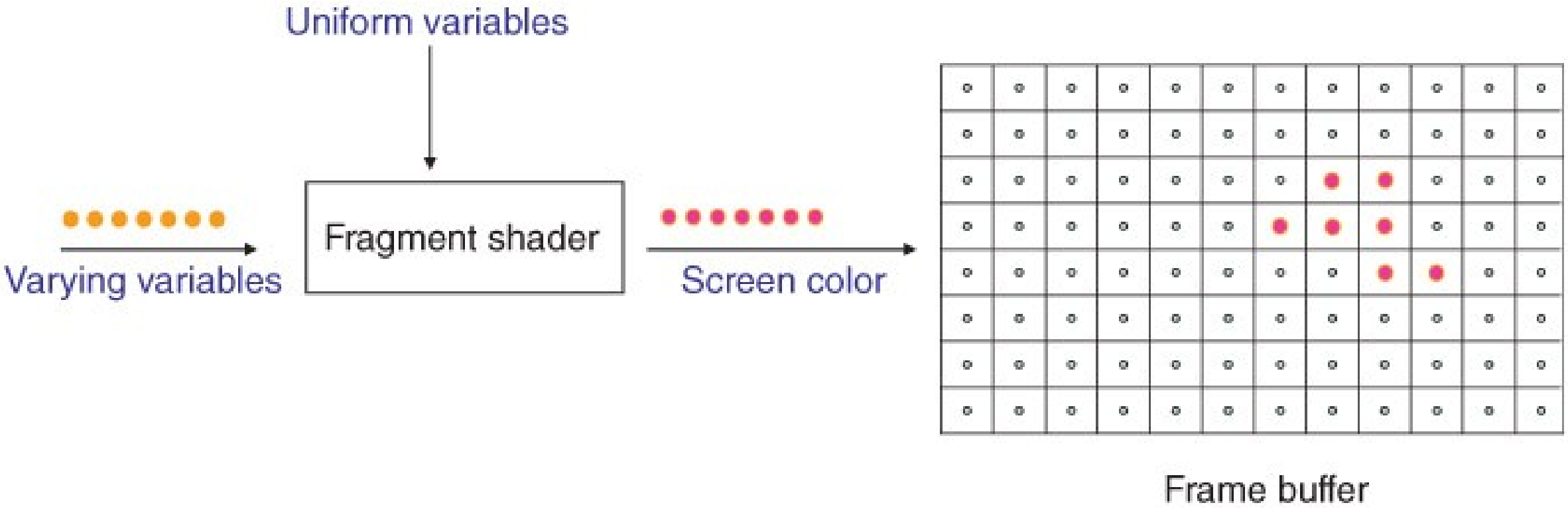
LES CONCEPTS

- uniform — JS + Three.JS → Vertex Shader → Fragment Shader
  - La même chose pour tous les sommets
- varying — Vertex Shader → Fragment Shader
  - Calculé pour chaque sommet, puis interpolé automatiquement pour les fr
- attribute — JS + Three.JS → Vertex Shader
  - N'importe quelle valeur par sommet
  - Disponible uniquement dans Vertex Shader

VERTEX SHADER



FRAGMENT SHADER



ATTACHING SHADERS

```
var remoteMaterial  = new THREE.ShaderMaterial({
  uniforms: {
  remotePosition:  remotePosition,
 },});
//voici le chargement des fichiers de shaders dans shaders[] …
remoteMaterial.vertexShader = shaders['glsl/remote.vs.glsl'];
remoteMaterial.fragmentShader = shaders['glsl/remote.fs'];  var
remoteGeometry = new THREE.SphereGeometry(1, 32, 32);  var
remote = new THREE.Mesh(remoteGeometry,  remoteMaterial);

scene.add(remote);
```

PLUS D’INFORMATION

https://threejs.org/docs/#api/en/renderers/webgl/WebGLProgram
https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf
http://www.shaderific.com/glsl/

OPENGL RENDERING PIPELINE