M2-Images

Rendu Temps Réel - openGL et shaders

J.C. Iehl

September 30, 2015

Présentation de l'api

api C ansi :

- pas de surcharge, mais une famille de fonctions,
- types opaques,
- objet implicite.

exemple :

```
// version c++
GLContext *context= new GLContext();
GLBuffer *buffer= context->createBuffer();
context->bindAttributeBuffer(0, buffer);

// version c
struct context *context= context_create();
struct buffer *buffer= context_createBuffer(context);
context_bindAttributeBuffer(context, 0, buffer);

// version GL
createContext();
GLuint buffer= createBuffer();
bindAttributeBuffer(0, buffer);
```

exemple : creation d'un buffer

```
GLuint buffer;
glGenBuffers( 1, &buffer );

// selectionne le buffer
glBindBuffer( GL_ARRAY_BUFFER, buffer );

// modifie une propriete du buffer selectionne sur GL_ARRAY_BUFFER
glBufferData( GL_ARRAY_BUFFER, size, data );

// question :
glBufferData( GL_ELEMENT_ARRAY_BUFFER, size, data );
```

pas de surcharge en C :

```
// version c++
glUniform( float x );
glUniform( vec3 v );

// version c
glUniform1f( float x );
glUniform3fv( float *v );
```

Objets principaux

nécessaires pour un `draw( )` :

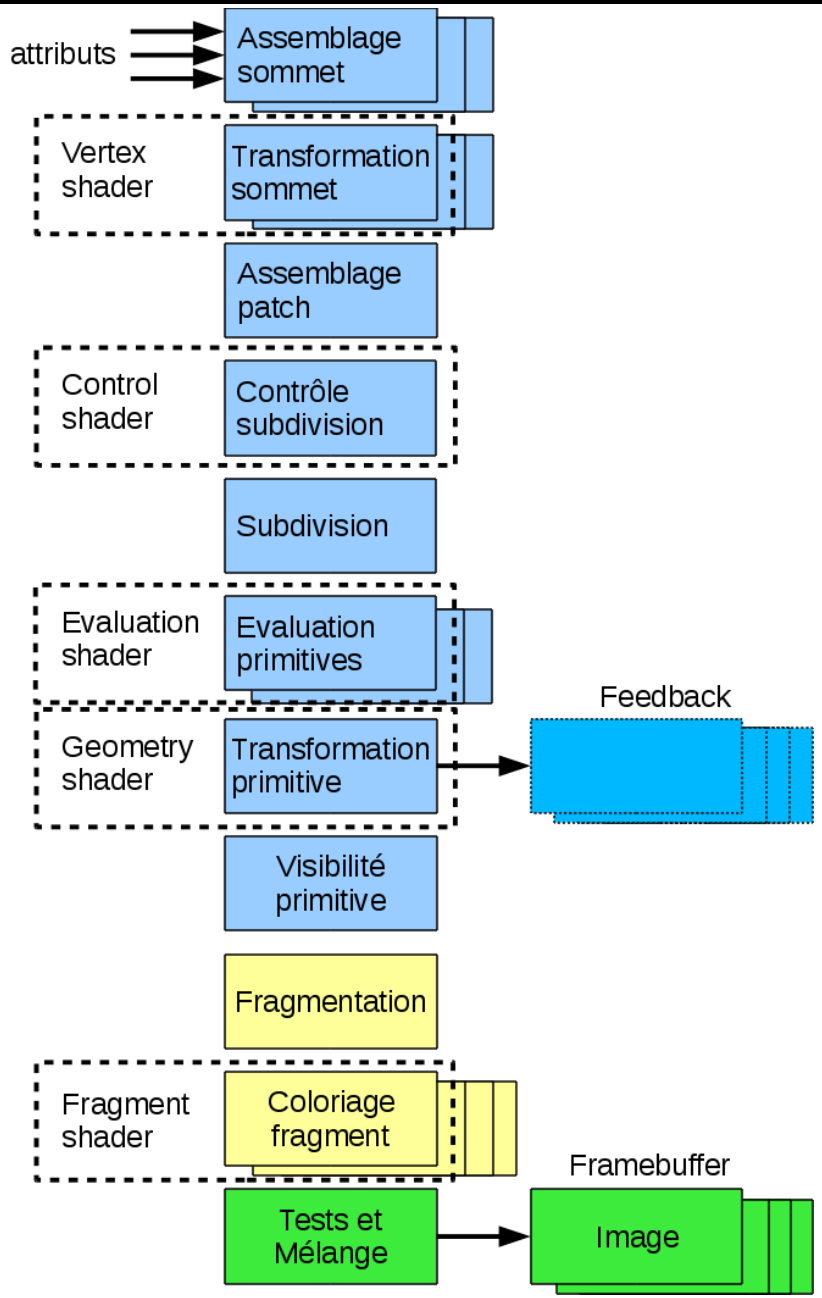
- nécessaires à l'exécution du pipeline.

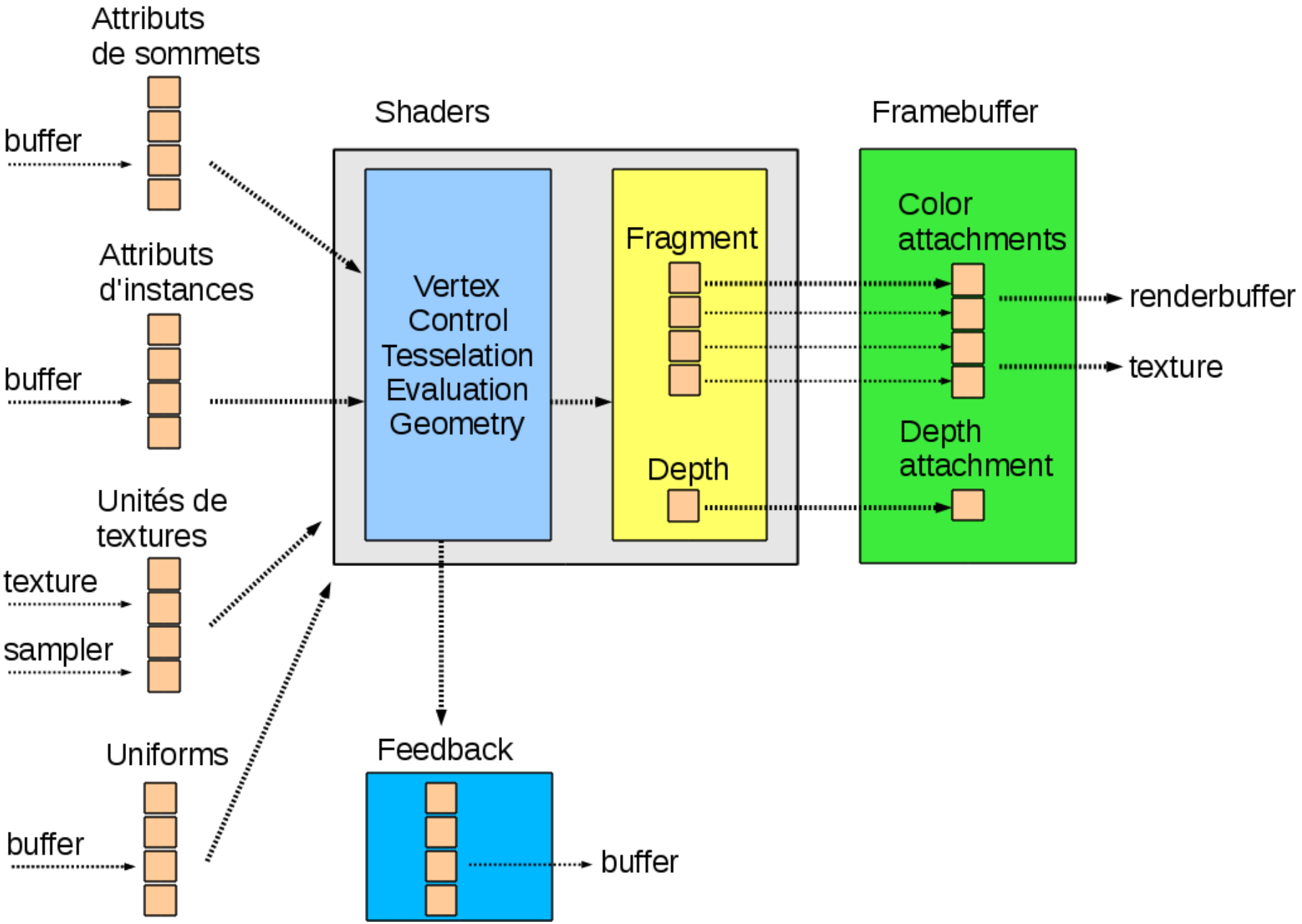
2 étapes :

- création des objets openGL,
- configuration du pipeline,
- draw( ).

Objets principaux

- vertex array object (assemblage de sommets)
- vertex buffer (attributs des sommets),
- index buffer (sommets partagés),
- shader program,
- configuration du pipeline fixe (assemblage primitives, test visibilité primitives, etc.)


attributs
Assemblage sommet
Vertex shader
Transformation sommet
Assemblage patch
Control shader
Contrôle subdivision
Subdivision
Evaluation shader
Evaluation primitives
Feedback
Geometry shader
Transformation primitive
Visibilité primitive
Fragmentation
Fragment shader
Coloriage fragment
Framebuffer
Tests et Mélange
Image


Attributs
de sommets
buffer
Attributs
d'instances
buffer
Unités de
textures
texture
sampler
Uniforms
buffer
Shaders
Vertex
Control
Tesselation
Evaluation
Geometry
Fragment
Depth
Framebuffer
Color
attachments
Depth
attachment
renderbuffer
texture
Feedback
buffer

Création avec gKit

gKit propose (plein) d'utilitaires :

- GLBuffer *createBuffer( );
- GLVertexArray *createVertexArray( ), GLBasicMesh,
- GLProgram *createProgram( );
- GLFramebuffer *createFramebuffer( );

la doc donne l'équivalent openGL de chaque fonction...

Configuration du pipeline avec gKit

GLPipeline :

- version explicite du pipeline et de sa configuration :
- partie programmable,
- partie fixe.

pas complet. mais l'objectif est de commencer avec GLPipeline et de finir avec openGL.

Configuration du pipeline avec gKit

```
// creation pipeline, shader program obligatoire
gk::GLPipeline *pipeline= gk::createPipeline(program->name);

// uniforms du shader program
pipeline->setUniformMatrix( "mvpMatrix", gk::Transform().matrix() );
pipeline->setUniform( "color", 1, 0, 0, 1 );

// configuration assemblage des attributs de sommets
// + index buffer
pipeline->bindVertexArray(vao->name);

// configure la partie fixe
pipeline->bindFramebuffer();
pipeline->setViewport(0, 0, width, height);
pipeline->setClearColor();    // valeur par defaut (noir)
pipeline->setClearDepth();    // valeur par defaut (far = 1)
pipeline->setCullFace();      // valeur par defaut (on, cull back faces)
pipeline->setDepthTest();     // valeur par defaut (on, less)
pipeline->setPolygonMode();   // valeur par defaut (fill)

// draw
pipeline->clear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
pipeline->drawElements(GL_TRIANGLES, mesh->count,
    GL_UNSIGNED_INT, 0);

// debug(pipeline);
```

Configuration du pipeline avec openGL

```
glUseProgram(program->name);
glBindVertexArray(vao->name);

// initialiser les transformations
float mvp[] = {
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
};
GLuint mvp_location= glGetUniformLocation(program->name, "mvpMatrix");
glUniformMatrix4fv(mvp_location, 1, GL_TRUE, mvp);

// donner une couleur a l'objet
float color[] = { 1, 1, 0 };
GLuint color_location= glGetUniformLocation(program->name, "color");
glUniform3fv(color_location, 1, color);

// draw
glDrawElements(GL_TRIANGLES, mesh->count, GL_UNSIGNED_INT, 0);
// utilise implicitement les valeurs par defaut pour la partie fixe
```

Configuration du pipeline avec openGL

```
// framebuffer
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
glDrawBuffer(GL_BACK);

// viewport
glViewport(0, 0, width, height);
glClearColor(0, 0, 0, 1);
glClearDepthf(1);

// back facing culling
glCullFace(GL_BACK);
glFrontFace(GL_CCW);
glEnable(GL_CULL_FACE);

// depth test
glDepthFunc(GL_LESS);
glEnable(GL_DEPTH_TEST);

// polygon mode
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

Squelette application

3 versions :

- openGL : `mini_gl3core.cpp`
- gKit : `mini_gkit2.cpp`
- pipeline : `mini_pipeline.cpp`