OpenGL

il était une fois dans l'ouest sauvage …

Sources

- http://raptor.developpez.com/tutorial/opengl/vbo/
- http://www.opengl-tutorial.org/fr
- http://www.fevrierdorian.com/blog/pages/Mes-Tutoriaux-Scripts
- http://antongerdelan.net/opengl
- https://openclassrooms.com/courses/wikipedia

OpenGL : Histoire

janvier 1992 -

- ARB (Architecture Review Board) => Khronos Group
  - AMD/ATI, Apple, Dell, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Matrox, Nvidia, SGI et Sun,…
  - Microsoft sort en 2003 (… car Direct X)
- 1.5 Shaders en extentions
- 2.0 GLSL officiel … OpenGL ES 2 ans plus tard
- 3.0 Deprecation, refonte, nouveau modèle …
- 3.3 / 4.0 Remise au gout du jour face à Direct X 10/11
- Vulkan : Tres bas niveau … pour le performances

NV|ATI

EXT

ARB

Avant

```
glClear(GL_COLOR_BUFFER_BIT);// Effacer la surface graphique
glColor3f(0, 0, 1.0);  // Encre bleue au départ
glBegin(GL_POLYGON);    // Commencer un polygone
  glVertex2i(100, 100); // Coordonnées des trois points
  glVertex2i(400, 100);
  glVertex2i(250, 400);
glEnd();                // Fermer le polygone
glFlush();         // Dessiner le polygone
```

mais …

- Avant de lancer du code OpenGL
  - Il faut une fenêtre avec un « contexte OpenGL »
  - Savoir quelles extensions sont disponibles
  - Une librairie de math (math .h ?)
  - Une librairie d'image (pour charger des imgs)
- Ce n'est pas simple !

Fenêtre + contexte

- Solution native (X11, win32, etc.)
- GLUT (GL utility toolkit) -ANCIEN- … freeGLUT
- SDL - Orienté JEU -
- GLFW
    - solution avec le vent en poupe
    - input devices + event loop

Extensions

- Tester les fonctions une à une …
- GLEW (Extension Wrangler)

  - attentions aux indiens en embuscade (GL3W, glLoadGen, glad, etc)

NV|ATI

EXT

ARB

```
#include <GL/glew.h> // AVANT les autres includes du genre GL/gl.h

…

if (glewInit() != GLEW_OK) {

    fprintf(stderr, "Failed to initialize GLEW\n");

    return -1;

} // AVANT d'utiliser GL wt GLEW mas après l'initialisation de GLFW
```

Librairie de math

- Math.h
  - ne gère pas le calcul matriciel !
- GLM (OpenGL Mathematics)
  - basé sur la syntaxe GLSL (vec3, mat4, etc.)

```
#include <glm/vec3.hpp> // glm::vec3
#include <glm/vec4.hpp> // glm::vec4
#include <glm/mat4x4.hpp> // glm::mat4
#include <glm/gtc/matrix_transform.hpp> // glm::translate, glm::rotate, glm::scale, glm::perspective
glm::mat4 camera(float Translate, glm::vec2 const & Rotate)
{
  glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 4.0f / 3.0f, 0.1f, 100.f);
  glm::mat4 View = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, -Translate));
  View = glm::rotate(View, Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
  View = glm::rotate(View, Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));
  glm::mat4 Model = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));
  return Projection * View * Model;
}
```

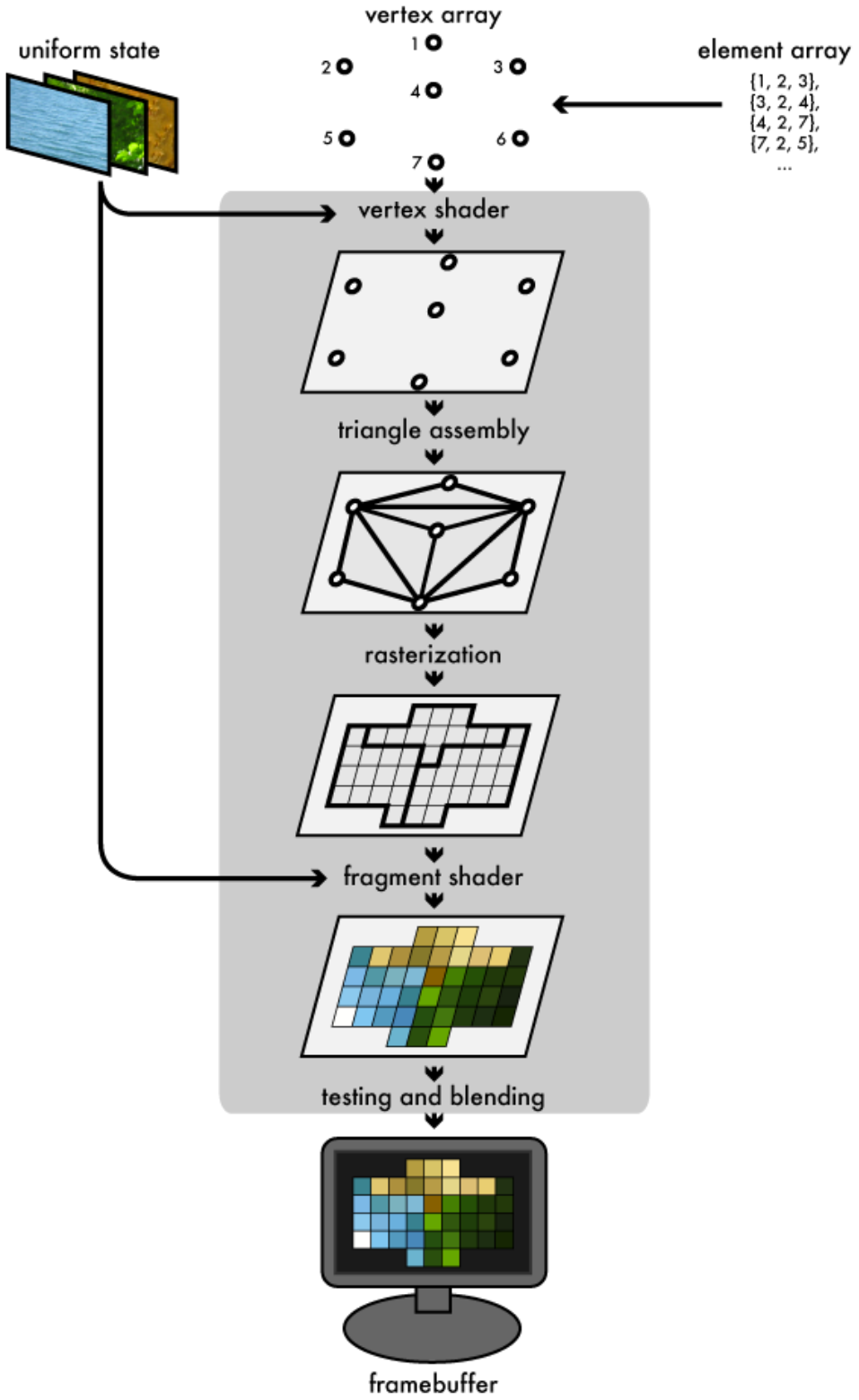
Librairie d'image

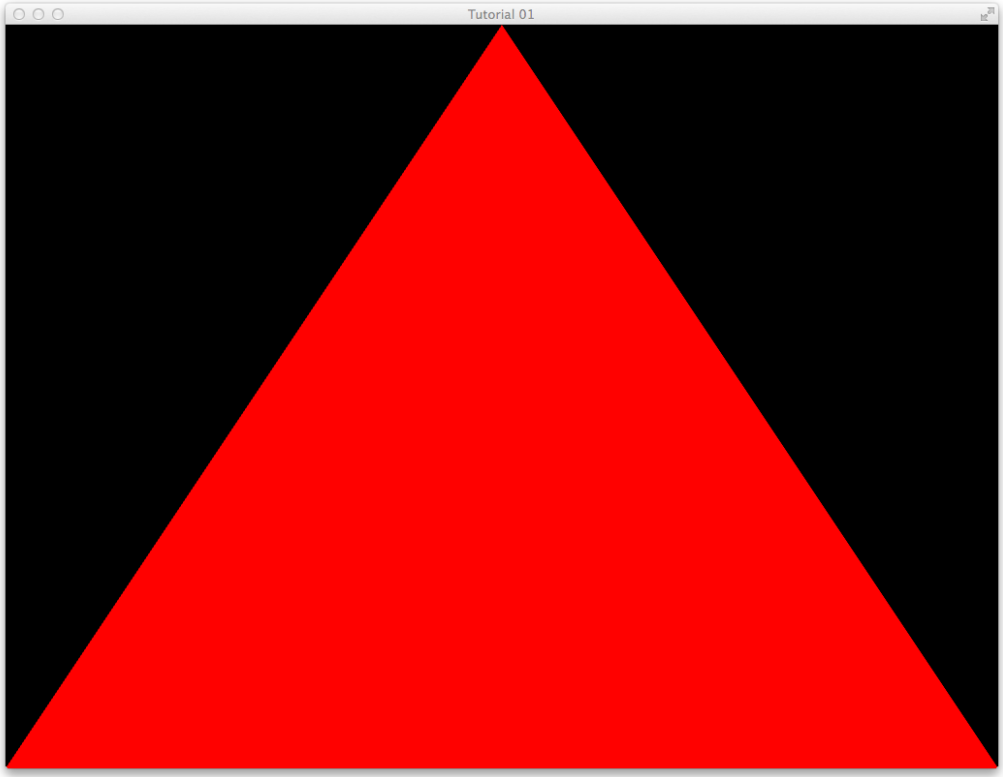
- Pour charger des textures dans OpenGL
- doit gérer de très nombreux formats (png, jpeg, gif, bmp, raw, etc), les encodages (RGBA, BGRA, etc)
- SDL, freeGLUT, SFML ou openCV ?
  - openCV est une énorme usine à gaz mais gérera les flux videos (live ou fichiers videos)

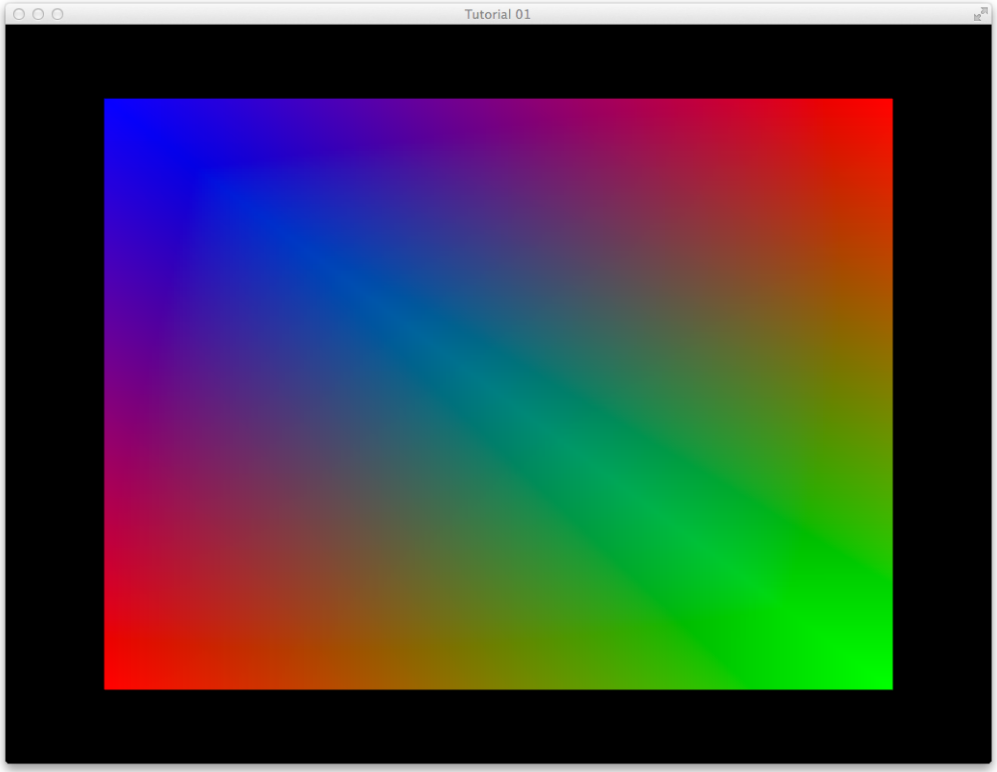
OK c’est parti.

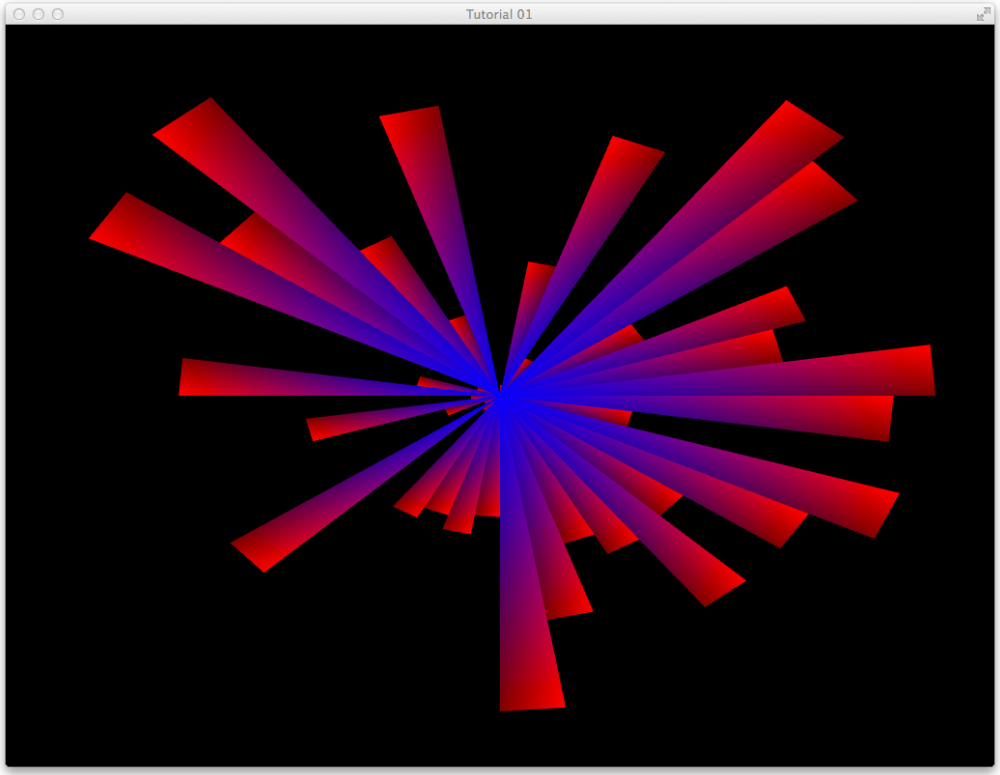
- Dessin Direct (NON !)
- Vertex Buffer Object (VBO) = stockage persistant en mémoire Video
- Vertex Array object (VAO) = automatisation de la manipulation du VBO
- FrameBuffer Object (FBO) = généralisation de la sortie (écran / texture)

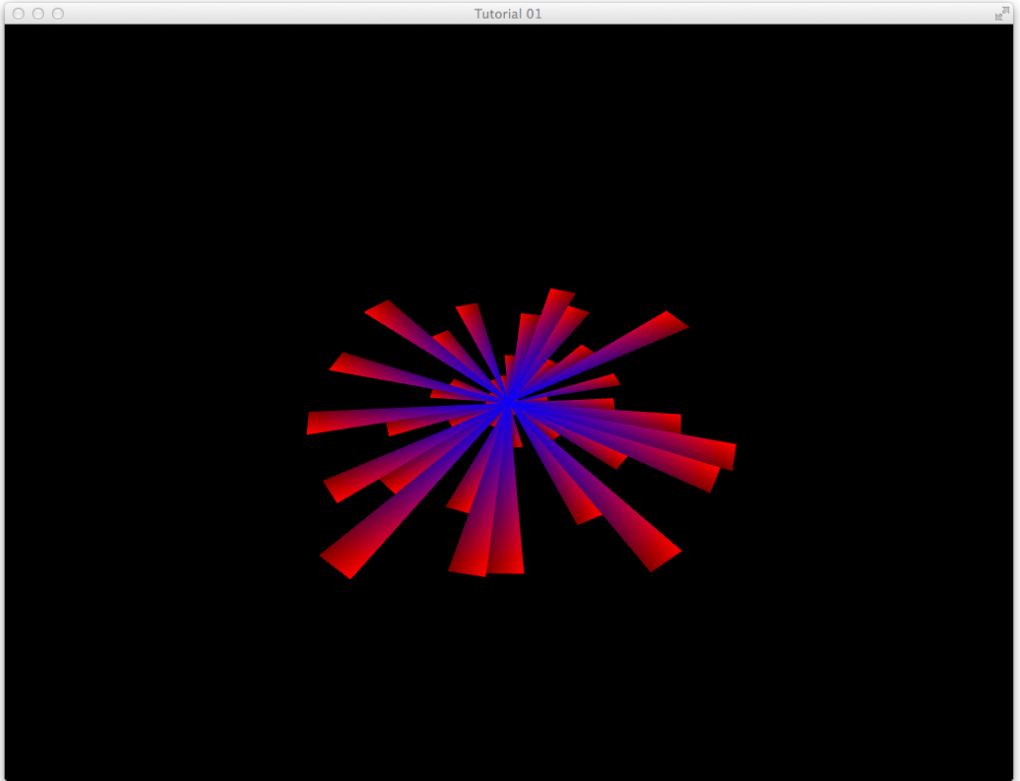
Le pipeline graphique




Tutorial 01

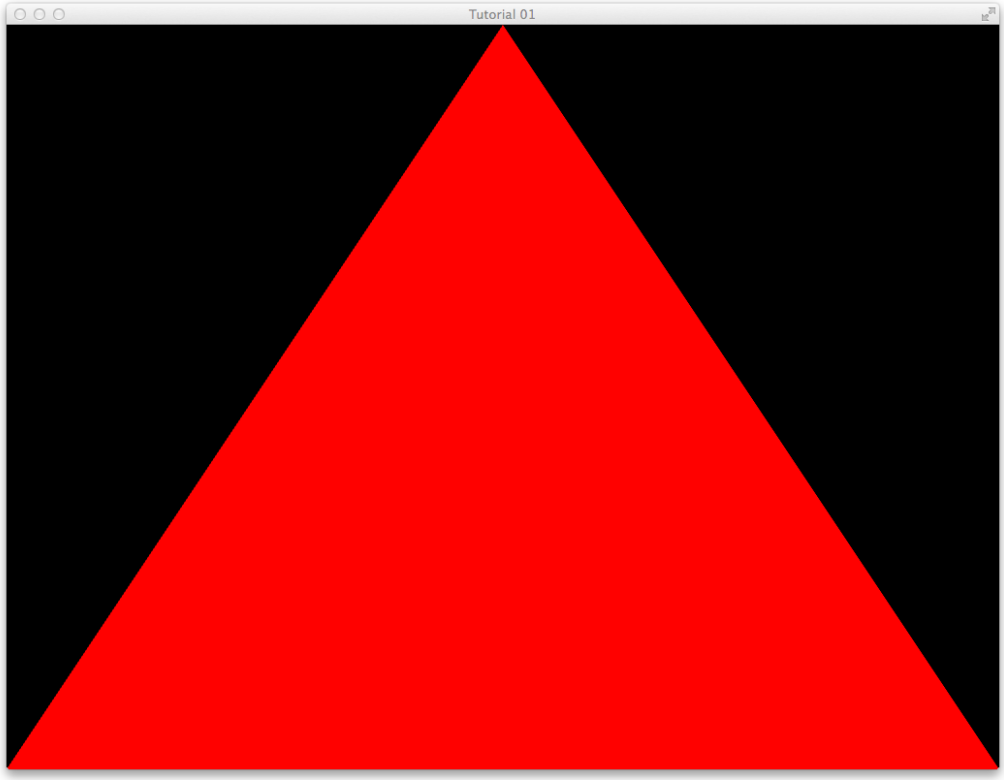

Tutorial 01


Tutorial 01


Tutorial 01

Etape 1 : un triangle

- VBO
- GLFW
- GLSL



le code

```
// Inclut les en-têtes standards

#include <stdio.h>

#include <string>

#include <vector>

#include <iostream>

#include <fstream>

#include <algorithm>

using namespace std;



#include <stdlib.h>

#include <string.h>



#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

#ifdef __APPLE__

#include <OpenGL/gl.h>

#else

#include <GL/gl.h>

#endif
```

```
GLuint LoadShaders(const char * vertex_file_path,const char * fragment_file_path){
	// Create the shaders
	GLuint VertexShaderID = glCreateShader(GL_VERTEX_SHADER);
	GLuint FragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);

	// Read the Vertex Shader code from the file
	std::string VertexShaderCode;
	std::ifstream VertexShaderStream(vertex_file_path, std::ios::in);
	if(VertexShaderStream.is_open()){
		std::string Line = "";
		while(getline(VertexShaderStream, Line))
			VertexShaderCode += "\n" + Line;
		VertexShaderStream.close();
	}else{
		printf("Impossible to open %s. Are you in the right directory ? Don't forget to read the FAQ !\n", vertex_file_path);
		getchar();
		return 0;
	}

	// Read the Fragment Shader code from the file
	std::string FragmentShaderCode;
	std::ifstream FragmentShaderStream(fragment_file_path, std::ios::in);
	if(FragmentShaderStream.is_open()){
		std::string Line = "";
		while(getline(FragmentShaderStream, Line))
			FragmentShaderCode += "\n" + Line;
		FragmentShaderStream.close();
	}

	GLint Result = GL_FALSE;
	int InfoLogLength;

	// Compile Vertex Shader
	printf("Compiling shader : %s\n", vertex_file_path);
	char const * VertexSourcePointer = VertexShaderCode.c_str();
	glShaderSource(VertexShaderID, 1, &VertexSourcePointer , NULL);
	glCompileShader(VertexShaderID);

	// Check Vertex Shader
	glGetShaderiv(VertexShaderID, GL_COMPILE_STATUS, &Result);
	glGetShaderiv(VertexShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
	if ( InfoLogLength > 0 ){
		std::vector<char> VertexShaderErrorMessage(InfoLogLength+1);
		glGetShaderInfoLog(VertexShaderID, InfoLogLength, NULL, &VertexShaderErrorMessage[0]);
		printf("%s\n", &VertexShaderErrorMessage[0]);
	}

	// Compile Fragment Shader
	printf("Compiling shader : %s\n", fragment_file_path);
	char const * FragmentSourcePointer = FragmentShaderCode.c_str();
	glShaderSource(FragmentShaderID, 1, &FragmentSourcePointer , NULL);
	glCompileShader(FragmentShaderID);

	// Check Fragment Shader
	glGetShaderiv(FragmentShaderID, GL_COMPILE_STATUS, &Result);
	glGetShaderiv(FragmentShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
	if ( InfoLogLength > 0 ){
		std::vector<char> FragmentShaderErrorMessage(InfoLogLength+1);
		glGetShaderInfoLog(FragmentShaderID, InfoLogLength, NULL, &FragmentShaderErrorMessage[0]);
		printf("%s\n", &FragmentShaderErrorMessage[0]);
	}

	// Link the program
	printf("Linking program\n");
	GLuint ProgramID = glCreateProgram();
	glAttachShader(ProgramID, VertexShaderID);
	glAttachShader(ProgramID, FragmentShaderID);
	glLinkProgram(ProgramID);

	// Check the program
	glGetProgramiv(ProgramID, GL_LINK_STATUS, &Result);
	glGetProgramiv(ProgramID, GL_INFO_LOG_LENGTH, &InfoLogLength);
	if ( InfoLogLength > 0 ){
		std::vector<char> ProgramErrorMessage(InfoLogLength+1);
		glGetProgramInfoLog(ProgramID, InfoLogLength, NULL, &ProgramErrorMessage[0]);
		printf("%s\n", &ProgramErrorMessage[0]);
	}

	glDetachShader(ProgramID, VertexShaderID);
	glDetachShader(ProgramID, FragmentShaderID);
	glDeleteShader(VertexShaderID);
	glDeleteShader(FragmentShaderID);
	return ProgramID;
}
```

GLFW et GLEW

```
int main(){
     // Initialise GLFW
    if( !glfwInit()) {
        fprintf( stderr, "Failed to initialize GLFW\n" );
        return -1;
    }

    glfwWindowHint(GLFW_SAMPLES, 4); // 4x antialiasing
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3); // On veut OpenGL 3.3
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
            // Pour rendre MacOS heureux ; ne devrait pas être nécessaire
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
            // On ne veut pas l'ancien OpenGL

    // Ouvre une fenêtre et crée son contexte OpenGl
    GLFWwindow* window;
    window = glfwCreateWindow( 1024, 768, "Tutorial 01", NULL, NULL);
    if( window == NULL) {
        fprintf( stderr, "Failed to open GLFW window. If you have an Intel GPU, they are
not 3.3 compatible. Try the 2.1 version of the tutorials.\n" );
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);

    // Initialise GLEW
    glewExperimental=true; // Nécessaire dans le profil de base
    if (glewInit() != GLEW_OK) {
        fprintf(stderr, "Failed to initialize GLEW\n");
        return -1;
    }
```

initialisation de openGL moderne

```
   GLuint VertexArrayID;
   glGenVertexArrays(1, &VertexArrayID);
   glBindVertexArray(VertexArrayID);

   // This will identify our vertex buffer
   GLuint vertexbuffer;
   // Generate 1 buffer, put the resulting identifier in
vertexbuffer
   glGenBuffers(1, &vertexbuffer);
   // The following commands will talk about our 'vertexbuffer'
buffer
   glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
   // Give our vertices to OpenGL.
   glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data),
g_vertex_buffer_data, GL_STATIC_DRAW);

   // Assure que l'on peut capturer la touche ESCAPE
   glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);

   GLuint programID = LoadShaders(
     "SimpleVertexShader0.vertexshader",
     "SimpleFragmentShader0.fragmentshader" );
```

Vertex shader et Fragment Shader

```
#version 330 core

layout(location = 0) in vec3
vertexPosition_modelspace;

void main(){
  gl_Position.xyz = vertexPosition_modelspace;
  gl_Position.w = 1.0;
}
```

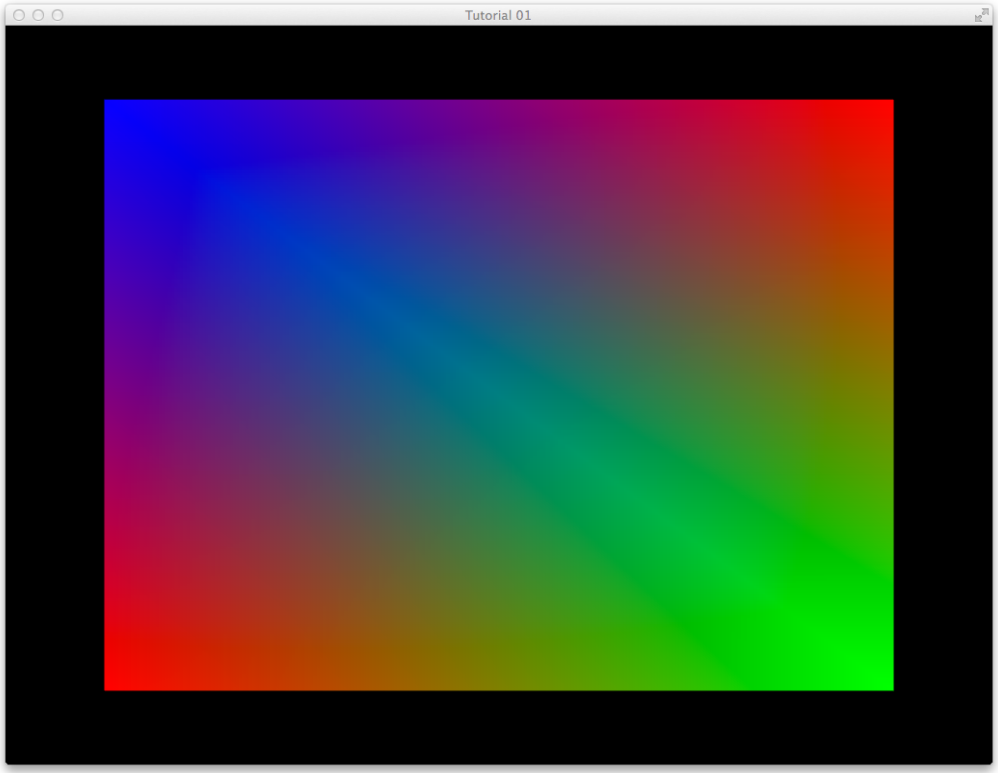
```
#version 330 core
out vec3 color;
void main(){
  color = vec3(1,0,0);
}
```

la boucle d'affichage

```
do {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Use our shader
        glUseProgram(programID);

        // 1rst attribute buffer : vertices
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
        glVertexAttribPointer(
           0,                  // No particular reason for 0, must match the layout shader.
           3,                  // size
           GL_FLOAT,           // type
           GL_FALSE,           // normalized?
           0,                  // stride
           (void*)0            // array buffer offset
        );
        // Draw the triangle !
        glDrawArrays(GL_TRIANGLES, 0, 3);  // from vertex 0…3 vertices= 1 triangle
        glDisableVertexAttribArray(0);

        // Swap buffers
        glfwSwapBuffers(window);
        glfwPollEvents();

    } // Vérifie si on a appuyé sur la touche échap (ESC) ou si la fenêtre a été fermée
    while( glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS &&
    glfwWindowShouldClose(window) == 0 );
```

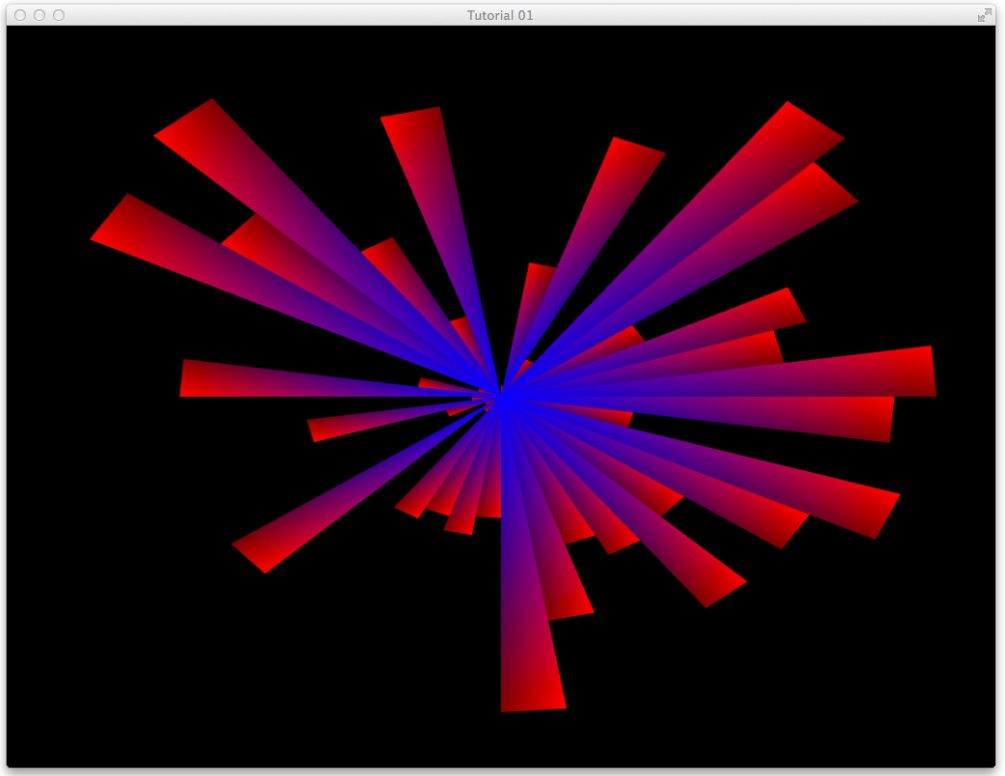
Etape 2 : des couleurs

- 2 buffers
- un shader plus compliqué



Etape 3 : des couleurs

- Des données
- Mises à jour



Etape 4 : des matrices

- projection
- modele
- vue

- dans des variables uniformes