Real-time graphics with OpenGL

Gaëtan Blaise-Cazalet - Head of programming section

Pre-requisites

Modern C++

The evolution of C++

C++ is a standard, implemented by multiple compilers. From 2011, the standard evolved a lot, while keeping backward compatibility. The goal is to keep the performance and flexibility of C++, while proposing safer and simpler interfaces to code.

This trend started in 2011 with C++11, and still continues today with C++14, C++17 and C++20. In studios, those norms are not necessarilly adopted, but they may. Thus it is wise two know the fundamentals of Modern C++.

This lecture, though very small in term of code, use modern C++. It is a prerequisites to understand its main concepts.

An overview of modern C++

Here is a good paper on microsoft C++ website :

https://docs.microsoft.com/en-us/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-160

Please consult its links to go deeper into each concept.

A small framework

An evolution of the OpenGL intro

This lectures uses an evolution of the Introduction to OpenGL and Shaders (year 2) lecture.

Before you start to code, it is wise to understand its simple structure. You can find the code at :

https://github.com/Gaetz/opengl-training/tree/master/AdvancedOpenGL

Main concepts

In addition to the Timer, the Log system and the Texture class, please take a look at :

- The Window interface and the WindowSDL implementation, used this time as a pointer. It shows how to use a std::unique_ptr in a Strategy pattern.
- The Game class that, this time, implement a Scene (GameState) pattern and handle scene transitions and stacking
- From Scene_006_..., the code use a system of macros to auto-change scenes and shaders. Macros are visible in the MacroUtils.h files

CMake compilation

CMake introduction

CMake is a well known and performant compilation system. It uses specific files, called CMakeLists.txt, to configure compilation.

Each source folder usually have it's CMakeLists.txt file. It allows to use each folder as a library, whose dependencies are inner folders.

This lectures uses Visual Code and CMake compilation. After installing CMake, you should install the CMake and CMake Tools extensions. You should then configue the project (Ctrl + Shift + P then CMake Configure), then build with F7.

CMake project configuration

Here is the main CMake file of the project :

```
cmake_minimum_required(VERSION 3.15)
project(OpenGLTraining VERSION 0.1.0)

set(OpenGL_GL_PREFERENCE "GLVND")
set(CMAKE_CXX_STANDARD 17)

include(CTest)
enable_testing()

# Includes and libraries
if (WIN32)
    set(SDL2_DIR ${CMAKE_SOURCE_DIR}/external/SDL2-2.0.12)
    set(GLEW_DIR ${CMAKE_SOURCE_DIR}/external/glew-2.1.0)
endif (WIN32)


find_package(OpenGL REQUIRED COMPONENTS OpenGL)

find_package(SDL2 REQUIRED)
include_directories(${SDL2_INCLUDE_DIRS})

find_package(GLEW REQUIRED)
include_directories(${GLEW_INCLUDE_DIRS})

find_package(OpenGL)

# subdirectories
add_subdirectory( src/engine )
add_subdirectory( src/game )

# Executable and link
if (NOT WIN32)
    string(STRIP ${SDL2_LIBRARIES} SDL2_LIBRARIES)
endif (NOT WIN32)
add_executable(OpenGLTraining src/main.cpp)
target_link_libraries(OpenGLTraining game engine maths input ${GLEW_LIBRARIES} ${SDL2_LIBRARIES} OpenGL::GL OpenGL::GLU)


if (WIN32)
    file(COPY external/SDL2-2.0.12/lib/x64/SDL2.dll DESTINATION ${CMAKE_BINARY_DIR}/Debug)
    file(COPY external/SDL2_mixer-2.0.4/lib/x64/SDL2_mixer.dll DESTINATION ${CMAKE_BINARY_DIR}/Debug)
    file(COPY external/SDL2_ttf-2.0.15/lib/x64/SDL2_ttf.dll DESTINATION ${CMAKE_BINARY_DIR}/Debug)
    file(COPY external/SDL2_ttf-2.0.15/lib/x64/zlib1.dll DESTINATION ${CMAKE_BINARY_DIR}/Debug)
    file(COPY external/glew-2.1.0/bin/Release/x64/glew32.dll DESTINATION ${CMAKE_BINARY_DIR}/Debug)
endif (WIN32)

if (WIN32)
    file(COPY assets DESTINATION ${CMAKE_BINARY_DIR}/Debug)
else (WIN32)
    file(COPY assets DESTINATION ${CMAKE_BINARY_DIR})
endif (WIN32)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

Set the directories to find libs

Includes the libs, so that the find_package function can find them

Add code subfolders. They will also possess a CMakeLists file

We can copy files

An example of a subdirectory CMakeLists file, for the engine folder :

```
file( GLOB engine_SOURCES *.cpp )
add_subdirectory( maths )
add_subdirectory( input )
add_library( engine ${engine_SOURCES} )
target_include_directories(engine PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})
```

Get all cpp files

Other subfolders

Declare this folder as a library

Finding libraires

Under Windows at least, you need a specific file in the folder of your dependancies to help CMake find them.

For instance the ./external/SDL2-2.0.12 contains a SDL2Config.cmake file. This file details where to find the includes and compiled libs.

Overview of the complete graphics pipeline

History and concepts of OpenGL

Read the 01.IntroToOpenGL intro document to learn about the basic concepts of OpenGL.

First use of the framework (1/2)

A colored screen

This first application will be a simple use of the framework. We will "erase" the screen with a color.

Scene_001_Color.cpp

```
void Scene_001_Color::draw() {
    static const GLfloat red[] = {1.0f, 0.0f, 0.0f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, red);
}
```

Because WindowSdl run the glClear command, the screen is erased in red each frame. We could update with time the color in the draw function, to get different colors.

First primitive : a point

OpenGL can draw several primitives : points, lines, triangles, and different combinations of triangles.

We will now use a simple shader to draw a first primitive : a point.

002_Point.vert

```
#version 450 core

void main()
{
    gl_Position = vec4(0.0, 0.0, 0.5, 1.0);
}
```

002_Point.frag

```
#version 450 core
out vec4 color;

void main()
{
    color = vec4(0.0, 0.8, 1.0, 1.0);
}
```

We will change the point size so it is visible and draw using GL_POINTS instead of the usual GL_TRIANGLES.

Scene_002_Point.h

```
...
private:
    Game *game;
    GLuint vao;
    Shader shader;
...
```

Scene_002_Point.cpp

```
...
void Scene_002_Point::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader("assets/shaders/002_Point.vert", "assets/shaders/002_Point.frag", "", "", "", "002_Point");
    shader = Assets::getShader("002_Point");
    glCreateVertexArrays(1, &vao);
    glBindVertexArray(vao);
    glPointSize(40.0f);
}
...
void Scene_002_Point::draw() {
    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.0f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    shader.use();
    glDrawArrays(GL_POINTS, 0, 1);
}
```

First use of the framework (2/2)

Second primitive : a triangle

003_Triangle.vert

```
#version 450 core

void main(void)
{
    const vec4 vertices[3] = vec4[3](
        vec4(0.25, -0.25, 0.5, 1.0),
        vec4(-0.25, -0.25, 0.5, 1.0),
        vec4(0.25, 0.25, 0.5, 1.0)
    );
    gl_Position = vertices[gl_VertexID];
}
```

003_Triangle.frag

```
#version 450 core
out vec4 color;

void main()
{
    color = vec4(0.0, 0.8, 1.0, 1.0);
}
```

Scene_003_Triangle.h

```
...
private:
    Game *game;
    GLuint vao;
    Shader shader;
...
```

Scene_003_Triangle.cpp

```
...
void Scene_003_Triangle::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader("assets/shaders/003_Triangle.vert", "assets/shaders/003_Triangle.frag", "", "", "", "003_Triangle");
    shader = Assets::getShader("003_Triangle");
    glCreateVertexArrays(1, &vao);
    glBindVertexArray(vao);
}
...
void Scene_003_Triangle::draw() {
    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.0f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    shader.use();
    glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

Passing data from stage to stage

Changing the color of the triangle

We want to draw a changing color moving triangle. We will pass an offset data to the vertex shader, and a color that the vertex shader will forward to the fragment shader. It is a good habit to store flowing data into a struct. The vertex shader will send data and the fragment shader will receive it.

004_PassingData.vert

```
#version 450 core

layout (location = 0) in vec4 offset;
layout (location = 1) in vec4 color;

out VS_OUT {
    vec4 color;
} vs_out;

void main(void)
{
    const vec4 vertices[3] = vec4[3](
        vec4(0.25, -0.25, 0.5, 1.0),
        vec4(0.25, 0.25, 0.5, 1.0),
        vec4(0.25, 0.25, 0.5, 1.0)
    );

    gl_Position = vertices[gl_VertexID] + offset;
    vs_out.color = color;
}
```

004_PassingData.frag

```
#version 450 core

in VS_OUT {
    vec4 color;
} fs_in;

out vec4 color;

void main()
{
    color = fs_in.color;
}
```

We need to compute some variable in the CPU application. The Color class is presented on next slide.

Scene_004_PassingData.h

```
...
private:
    Game *game;
    GLuint vao;
    Shader shader;

    float timeSinceStart;
    Color displayColor;
...
```

Scene_004_PassingData.cpp

```
...
void Scene_004_PassingData::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader("assets/shaders/004_PassingData.vert", "assets/shaders/004_PassingData.frag", "", "", "", "004_PassingData");
    shader = Assets::getShader("004_PassingData");
    glCreateVertexArrays(1, &vao);
    glBindVertexArray(vao);
}
...
void Scene_004_PassingData::update(float dt) {
    timeSinceStart = (float)SDL_GetTicks() / 1000.0f;
    displayColor = Color((float)sin(timeSinceStart) *0.5f * 255.0f, (float)cos(timeSinceStart) *0.5f * 255.0f, 0.0f, 255);
}

void Scene_004_PassingData::draw() {

    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.0f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    GLfloat offset[] = { (float)sin(timeSinceStart) *0.5f + 0.5f, (float)cos(timeSinceStart) *0.5f + 0.5f, 0.0f, 1.0f};
    glVertexAttrib4fv(0, offset);
    glVertexAttrib4fv(1, displayColor.toGlArray());

    shader.use();
    glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

Color

Our color class will handle color data and interact with OpenGL.

src/engine/Color.h

```
#ifndef COLOR_H
#define COLOR_H

#include <cstdint>
#include <GL/glew.h>
#include "maths/Vector3.h"
#include "maths/Vector4.h"

class Color
{
public:
    Color();
    Color(uint8_t r, uint8_t g, uint8_t b, uint8_t a = 255);
    Color(uint32_t i);
    virtual ~Color();

    uint8_t r;
    uint8_t g;
    uint8_t b;
    uint8_t a;
    GLfloat* glArray;

    void setColor(uint8_t p_r, uint8_t p_g, uint8_t p_b, uint8_t p_a = 255);

    friend Color operator *(Color value, float scale)
    {
        return Color::multiply(value, scale);
    }

    Vector3 toVector3();
    Vector4 toVector4();
    GLfloat* toGlArray();

    static Color lerp(Color value1, Color value2, float amount);
    static Color multiply(Color value, float scale);

    static Color black;
    static Color white;
    static Color red;
    static Color green;
    static Color blue;
    static Color yellow;
    static Color lightYellow;
    static Color lightBlue;
    static Color lightPink;
    static Color lightGreen;
};

#endif
```

Color.cpp

```
#include "Color.h"

Color::Color() : r(255), g(255), b(255), a(255), glArray(nullptr) {}

Color::Color(uint8_t r, uint8_t g, uint8_t b, uint8_t a)
: r(r), g(g), b(b), a(a), glArray(nullptr) {}

Color::Color(uint32_t i) : glArray(nullptr)
{
    unsigned char r,g,b,a;
    r = static_cast<unsigned char>(i & 0x000000FF);
    g = static_cast<unsigned char>((i & 0x0000FF00) >> 8);
    b = static_cast<unsigned char>((i & 0x00FF0000) >> 16);
    a = static_cast<unsigned char>((i & 0xFF000000) >> 24);
}

Color::~Color() {}

void Color::setColor(uint8_t p_r, uint8_t p_g, uint8_t p_b, uint8_t p_a)
{
    r = p_r;
    g = p_g;
    b = p_b;
    a = p_a;
}


Color Color::lerp(Color value1, Color value2, float amount)
{
    uint8_t r = Maths::clamp<uint8_t>(static_cast<uint8_t>(Maths::lerp(value1.r, value2.r, amount)), 0, 255);
    uint8_t g = Maths::clamp<uint8_t>(static_cast<uint8_t>(Maths::lerp(value1.g, value2.g, amount)), 0, 255);
    uint8_t b = Maths::clamp<uint8_t>(static_cast<uint8_t>(Maths::lerp(value1.b, value2.b, amount)), 0, 255);
    uint8_t a = Maths::clamp<uint8_t>(static_cast<uint8_t>(Maths::lerp(value1.a, value2.a, amount)), 0, 255);

    return Color(r, g, b, a);
}

Color Color::multiply(Color value, float scale)
{
    uint8_t r = (uint8_t)(value.r * scale);
    uint8_t g = (uint8_t)(value.g * scale);
    uint8_t b = (uint8_t)(value.b * scale);
    uint8_t a = (uint8_t)(value.a * scale);

    return Color(r, g, b, a);
}

Vector3 Color::toVector3()
{
    Vector3 vector = Vector3(r, g, b);
    return vector;
}

Vector4 Color::toVector4()
{
    Vector4 vector = Vector4(r, g, b, a);
    return vector;
}

Color Color::black {0, 0, 0};
Color Color::white {255, 255, 255};
Color Color::red {255, 0, 0};
Color Color::green {0, 255, 0};
Color Color::blue {0, 0, 255};
Color Color::yellow {255, 255, 0};
Color Color::lightYellow {255, 255, 225};
Color Color::lightBlue {170, 217, 230};
Color Color::lightPink {255, 180, 200};
Color Color::lightGreen {142, 240, 142};

GLfloat* Color::toGlArray() {
    glArray = new GLfloat[4] { (float)r/255.0f, (float)g/255.0f, (float)b/255.0f, (float)a/255.0f };
    return glArray;
}
```

Tessellation and geometry shader

Start by reading the 02.TessellationGeometryShader document. This document also tackles the pipeline stages between the geometry shader and the fragment shader.

Tessellation and geometry example

This shader will use all shaders types in the graphics pipeline - except compute shader, but they are not considered as a part of the graphics pipeline per se.

005_Tessellation.vert

```
#version 450 core

void main(void)
{
    const vec4 vertices[3] = vec4[3](
        vec4(0.25, -0.25, 0.5, 1.0),
        vec4(-0.25, -0.25, 0.5, 1.0),
        vec4(0.25, 0.25, 0.5, 1.0)
    );

    gl_Position = vertices[gl_VertexID];
}
```

005_Tessellation.tecs

```
#version 450 core

layout (vertices = 3) out;

void main(void)
{
    if(gl_InvocationID == 0) {
        gl_TessLevelInner[0] = 5.0;
        gl_TessLevelOuter[0] = 5.0;
        gl_TessLevelOuter[1] = 5.0;
        gl_TessLevelOuter[2] = 5.0;
    }
    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
}
```

005_Tessellation.tese

```
#version 450 core

layout (triangles, equal_spacing, cw) in;

void main(void)
{
    gl_Position = (gl_TessCoord.x * gl_in[0].gl_Position +
        gl_TessCoord.y * gl_in[1].gl_Position +
        gl_TessCoord.z * gl_in[2].gl_Position);
}
```

005_Tessellation.geom

```
#version 450 core

layout (triangles) in;
layout (points, max_vertices = 3) out;

void main(void)
{
    int i;
    for (i=0; i < gl_in.length(); i++)
    {
        gl_Position = gl_in[i].gl_Position;
        EmitVertex();
    }
}
```

005_Tessellation.frag

```
#version 450 core

out vec4 color;

void main()
{
    color = vec4(0.0f, 0.8f, 1.0f, 1.0f);
}
```

The scene will use classic data.

Scene_005_Tessellation.h

```
...
private:
    Game *game;
    GLuint vao;
    Shader shader;
...
```

Scene_005_Tessellation.cpp

```
...
void Scene_005_Tessellation::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader("assets/shaders/005_tessellation.vert", "assets/shaders/005_tessellation.frag",
                       "assets/shaders/005_tessellation.tecs", "assets/shaders/005_tessellation.tese",
                       "assets/shaders/005_tessellation.geom", "005_tessellation");

    glCreateVertexArrays(1, &vao);
    glBindVertexArray(vao);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    shader = Assets::getShader("005_tessellation");
}
...
void Scene_005_Tessellation::draw() {
    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.2f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    shader.use();
    glPointSize(5.0f);
    glDrawArrays(GL_PATCHES, 0, 3);
}

void Scene_005_Tessellation::clean() {
    glDeleteVertexArrays(1, &vao);
}
```

Using a macro to switch between scenes

Before reaching the fragment shader, we will make advantage of a macro to easily switch between scenes and shaders. This will avoid us to copy and past scene and shader names everywhere.

First create a MacroUtils.h file. We need a lot of successive steps to get the names formatted the right way.

src/engine/MacroUtils.h

```
#define IDENT(x) x
#define XSTR(x) #x
#define STR(x) XSTR(x)
#define PATH(x, y) STR(IDENT(x)IDENT(y))
#define PATH3(p, x, y) STR(IDENT(p)IDENT(x)IDENT(y))
#define EXT .h
#define EXT_VERT .vert
#define EXT_FRAG .frag
#define SHADER_PATH assets/shaders/
#define SHADER_VERT(s) PATH3(SHADER_PATH, s, EXT_VERT)
#define SHADER_FRAG(s) PATH3(SHADER_PATH, s, EXT_FRAG)
#define SHADER_ID(s) STR(s)

#define SCENE_NAME Scene_006_Fragment
#define SHADER_NAME IDENT(006_Fragment)
```

We modify Game.cpp to load the correct scene.

Game.cpp

```
#include "../engine/MacroUtils.h"
#include "../engine/Game.h"
#include "../engine/Assets.h"
#include "../engine/Scene.h"
#include PATH(SCENE_NAME,EXT)
...
void Game::load() {
    // Game state
    changeState(std::make_unique<SCENE_NAME>());
}
...
```

To load and use a shader in a scene, we now easily call the shader at one place.

Scene_0xx_Xxx.cpp

```
void Scene_0xx_Xxx::load() {
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));
    ...
    shader = Assets::getShader(SHADER_ID(SHADER_NAME));
}
```

When we want to change the scene and the shader, we just have to edit the MacroUtils.h last two lines.

Fragment shader

The fragment shader is the last programmable stage in OpenGL`s graphics pipeline. This stage is responsible for determining the color of each fragment before it is sent to the framebuffer for possible composition into the window. After the rasterizer processes a primitive, it produces a list of fragments that need to be colored and passes this list to the fragment shader. Here, an explosion in the amount of work in the pipeline occurs, as each triangle could produce hundreds, thousands, or even millions of fragments.

The term fragment is used to describe an element that may ultimately contribute to the final color of a pixel. The pixel may not end up being the color produced by any particular invocation of the fragment shader due to a number of other effects such as depth or stencil tests, blending, and multi-sampling, all of which will be covered later.

006_Fragment.vert

```
#version 450 core

layout (location = 0) in vec4 offset;

out vec4 vs_color;

void main(void)
{
    const vec4 vertices[3] = vec4[3](
        vec4(0.25, -0.25, 0.5, 1.0),
        vec4(0.25, 0.25, 0.5, 1.0),
        vec4(0.25, 0.25, 0.5, 1.0)
    );

    const vec4 colors[3] = vec4[3](
        vec4(1.0, 0.0, 0.0, 1.0),
        vec4(0.0, 1.0, 0.0, 1.0),
        vec4(0.0, 0.0, 1.0, 1.0)
    );

    gl_Position = vertices[gl_VertexID] + offset;
    vs_color = colors[gl_VertexID];
}
```

006_Fragment.frag

```
#version 450 core

out vec4 color;

void main()
{
    color = vec4(
        sin(gl_FragCoord.x * 0.25) *0.5 + 0.5,
        cos(gl_FragCoord.y * 0.25) *0.5 + 0.5,
        sin(gl_FragCoord.z * 0.15) * cos(gl_FragCoord.y * 0.15),
        1.0
    );
}
```

Advanced buffers and uniforms (WIP)

About buffers and uniforms

Read 03.BuffersUniforms to go deeper into those concepts.

Buffer and uniforms example (1/2)

We'll make a spinning cube to get position data through the buffer and matrix view and projection data from the uniforms.

007_SpinningCube.vert

```
#version 450 core

in vec4 position;

out VS_OUT
{
    vec4 color;
} vs_out;

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;


void main(void)
{
    gl_Position = proj_matrix * mv_matrix * position;
    vs_out.color = position * 2.0 + vec4(0.5, 0.5, 0.5, 0.0);
}
```

007_SpinningCube.frag

```
#version 450 core

out vec4 color;

in VS_OUT
{
    vec4 color;
} fs_in;

void main()
{
    color = fs_in.color;
}
```

Scene_007_SpinningCube.h

```
...
private:
    Game *game;
    GLuint vao;
    GLuint buffer;
    Matrix4 transform;
    Matrix4 projection;

    Shader shader;
...
```

Scene_007_SpinningCube.cpp

```
...
void Scene_007_SpinningCube::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));

    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);
    projection = Matrix4::createPerspectiveFOV(70.0f, game->windowWidth, game->windowHeight, 0.1f, 1000.0f);
    static const GLfloat vertexPositions[] =
    {
        -0.25f,  0.25f, -0.25f,
        -0.25f, -0.25f, -0.25f,
         0.25f, -0.25f, -0.25f,

         0.25f, -0.25f, -0.25f,
         0.25f,  0.25f, -0.25f,
        -0.25f,  0.25f, -0.25f,

         0.25f, -0.25f, -0.25f,
         0.25f, -0.25f,  0.25f,
         0.25f,  0.25f, -0.25f,

         0.25f, -0.25f,  0.25f,
         0.25f,  0.25f,  0.25f,
         0.25f,  0.25f, -0.25f,

         0.25f, -0.25f,  0.25f,
        -0.25f, -0.25f,  0.25f,
         0.25f,  0.25f,  0.25f,

        -0.25f, -0.25f,  0.25f,
        -0.25f,  0.25f,  0.25f,
         0.25f,  0.25f,  0.25f,

        -0.25f, -0.25f,  0.25f,
        -0.25f, -0.25f, -0.25f,
        -0.25f,  0.25f,  0.25f,

        -0.25f, -0.25f, -0.25f,
        -0.25f,  0.25f, -0.25f,
        -0.25f,  0.25f,  0.25f,

        -0.25f, -0.25f,  0.25f,
         0.25f, -0.25f,  0.25f,
         0.25f, -0.25f, -0.25f,

         0.25f, -0.25f, -0.25f,
        -0.25f, -0.25f, -0.25f,
        -0.25f, -0.25f,  0.25f,

        -0.25f,  0.25f, -0.25f,
         0.25f,  0.25f, -0.25f,
         0.25f,  0.25f,  0.25f,

         0.25f,  0.25f,  0.25f,
        -0.25f,  0.25f,  0.25f,
        -0.25f,  0.25f, -0.25f
    };

    // Generate data and put it in buffer object
    glGenBuffers(1, &buffer);
    glBindBuffer(GL_ARRAY_BUFFER, buffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexPositions), vertexPositions, GL_STATIC_DRAW);

    // Setup vertex attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, nullptr);
    glEnableVertexAttribArray(0);

    glEnable(GL_CULL_FACE);
    glFrontFace(GL_CW);

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);

    shader = Assets::getShader(SHADER_ID(SHADER_NAME));
}
...
void Scene_007_SpinningCube::update(float dt) {
    const float t = Timer::getTimeSinceStart() * 0.3f;
    transform = Matrix4::createTranslation(Vector3(0.0f, 0.0f, -4.0f))
        * Matrix4::createTranslation(Vector3(Maths::sin(2.1f * t) * 0.5f, Maths::cos(1.7f * t) * 0.5f, Maths::sin(1.3f * t) * Maths::cos(1.5f * t) * 2.0f))
        * Matrix4::createRotationY(t * 45.0f / 10.0f)
        * Matrix4::createRotationX(t * 81.0f / 10.0f);
}

void Scene_007_SpinningCube::draw() {

    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.2f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    shader.use();
    shader.setMatrix4("mv_matrix", transform);
    shader.setMatrix4("proj_matrix", projection);

    glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

Buffer and uniforms example (2/2)

Of course, now that we have our cube geometry in a buffer object and a model–view matrix in a uniform, there's nothing to stop us from updating the uniform and drawing many copies of the cube in a single frame. We've modified the rendering function to calculate a new model–view matrix many times and repeatedly draw our cube. To enable depth testing, we make sure the depth test function is equal to GL_LEQUAL.

Scene_008_SpinningCubes.h

```
...
private:
    Game *game;
    GLuint vao;
    GLuint buffer;
    Matrix4 transform[24];
    Matrix4 projection;

    Shader shader;
...
```

Scene_008_SpinningCubes.cpp

```
...
void Scene_008_SpinningCubes::update(float dt) {
    for(int i = 0; i < 24; ++i)
    {
        const float t = i + Timer::getTimeSinceStart() * 0.3f;
        transform[i] = Matrix4::createTranslation(Vector3(0.0f, 0.0f, -10.0f))
            * Matrix4::createRotationY(t * Maths::toRadians(45.0f))
            * Matrix4::createRotationX(t * Maths::toRadians(21.0f))
            * Matrix4::createTranslation(Vector3(Maths::sin(2.1f * t) * 2.0f, Maths::cos(1.7f * t) * 2.0f, Maths::sin(1.3f * t) * Maths::cos(1.5f * t) *
2.0f));
    }
}

void Scene_008_SpinningCubes::draw()
{
    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.2f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    shader.use();
    shader.setMatrix4("proj_matrix", projection);
    for(int i = 0; i < 24; ++i)
    {
        shader.setMatrix4("mv_matrix", transform[i]);
        glDrawArrays(GL_TRIANGLES, 0, 36);
    }
}
```

About shader storage blocks and atomics

Read 04.StorageBlocksAtomics. Those concept won't be applied right now, so you don't have to grasp everything.

Advanced textures (WIP)

About textures

Read 05.Textures.

In the next slides, we will implement the simple texture example (with data hardcoded), then the torus, then the infinite corridor with mipmap textures.

We will need two new classes, for manage a specific type of texture and model.

Texture as data

009_TextureAsData.vert

```
#version 450 core

void main(void)
{
    const vec4 vertices[] = vec4[] (vec4( 0.75, -0.75, 0.5, 1.0),
                                    vec4(-0.75, -0.75, 0.5, 1.0),
                                    vec4( 0.75,  0.75, 0.5, 1.0));

    gl_Position = vertices[gl_VertexID];
}
```

009_TextureAsData.frag

```
#version 450 core

uniform sampler2D tex;
out vec4 color;

void main()
{
    color = texture(tex, gl_FragCoord.xy / textureSize(tex, 0));
}
```

Scene_009_TextureAsData.h

```
...
private:
    Game *game;
    GLuint vao;
    GLuint texture;
    Shader shader;

    void generateTexture(float * data, int width, int height);
};
```

Scene_009_TextureAsData.cpp

```
...
void Scene_009_TextureAsData::generateTexture(float * data, int width, int height)
{

    int x, y;
    for (y = 0; y < height; y++)
    {
        for (x = 0; x < width; x++)
        {
            data[(y * width + x) * 4 + 0] = (float)((x & y) & 0xFF) / 255.0f;
            data[(y * width + x) * 4 + 1] = (float)((x | y) & 0xFF) / 255.0f;
            data[(y * width + x) * 4 + 2] = (float)((x ^ y) & 0xFF) / 255.0f;
            data[(y * width + x) * 4 + 3] = 1.0f;
        }
    }
}

void Scene_009_TextureAsData::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));

    // Generate a name for the texture
    glGenTextures(1, &texture);

    // Now bind it to the context using the GL_TEXTURE_2D binding point
    glBindTexture(GL_TEXTURE_2D, texture);

    // Specify the amount of storage we want to use for the texture
    glTexStorage2D(GL_TEXTURE_2D,   // 2D texture
                   8,               // 8 mipmap levels
                   GL_RGBA32F,      // 32-bit floating-point RGBA data
                   256, 256);       // 256 x 256 texels

    // Define some data to upload into the texture
    float * data = new float[256 * 256 * 4];

    // generate_texture() is a function that fills memory with image data
    generateTexture(data, 256, 256);

    // Assume the texture is already bound to the GL_TEXTURE_2D target
    glTexSubImage2D(GL_TEXTURE_2D,  // 2D texture
                    0,              // Level 0
                    0, 0,           // Offset 0, 0
                    256, 256,       // 256 x 256 texels, replace entire image
                    GL_RGBA,        // Four channel data
                    GL_FLOAT,       // Floating point data
                    data);          // Pointer to data

    // Free the memory we allocated before - \GL now has our data
    delete [] data;

    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    shader = Assets::getShader(SHADER_ID(SHADER_NAME));
}
...
void Scene_009_TextureAsData::draw()
{
    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.2f, 1.0f};
    glClearBufferfv(GL_COLOR, 0, bgColor);

    shader.use();
    glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

KTX texture

TextureKtx.h

```
#ifndef TEXTURE_KTX_H
#define TEXTURE_KTX_H
#include <GL/glew.h>

struct TextureKtxHeader {
   unsigned char identifier[12],
   unsigned int endianness,
   unsigned int gltype,
   unsigned int gltypesize,
   unsigned int glformat,
   unsigned int glinternalformat,
   unsigned int glbaseinternalformat,
   unsigned int pixelwidth,
   unsigned int pixelheight,
   unsigned int pixeldepth,
   unsigned int arrayelements,
   unsigned int faces,
   unsigned int miplevels,
   unsigned int keypairbytes,
};

class TextureKtx {
  public:
   unsigned int id;

   union keyvaluepair {
      unsigned int size;
      unsigned char rawbytes[4],
   };

   unsigned int load(const char* filename, unsigned int tex = 0),
   bool save(const char* filename, unsigned int target, unsigned int tex),
   void use() const,

  private:
   int width, height;
   GLenum textureTarget,

   const unsigned int swap32(const unsigned int u32),
   const unsigned short swap16(const unsigned short u16),
   unsigned int calculateStride(const TextureKtxHeader& h, unsigned int width, unsigned int pad = 4),
   unsigned int calculateFaceSize(const TextureKtxHeader& h),

   static const unsigned char identifier[],
};

#endif
```

TextureKtx.cpp

```
#include "TextureKtx.h"

#include <cstdio>
#include <cstring>

const unsigned char TextureKtx::identifier[] = {0xAB, 0x4B, 0x54, 0x58, 0x20, 0x31, 0x31, 0xBB, 0x0D, 0x0A, 0x1A, 0x0A},

const unsigned int TextureKtx::swap32(const unsigned int u32) {
   union {
      unsigned int u32,
      unsigned char u8[4],
   } a, b;

   a.u32 = u32,
   b.u8[0] = a.u8[3],
   b.u8[1] = a.u8[2],
   b.u8[2] = a.u8[1],
   b.u8[3] = a.u8[0],

   return b.u32,
}

const unsigned short TextureKtx::swap16(const unsigned short u16) {
   union {
      unsigned short u16,
      unsigned char u8[2],
   } a, b;

   a.u16 = u16,
   b.u8[0] = a.u8[1],
   b.u8[1] = a.u8[0],

   return b.u16,
}

unsigned int TextureKtx::calculateStride(const TextureKtxHeader &h, unsigned int width, unsigned int pad) {
   unsigned int channels = 0,

   switch (h.glbaseinternalformat) {
      case GL_RED:
         channels = 1,
         break,
      case GL_RG:
         channels = 2,
         break,
      case GL_BGR:
      case GL_RGB:
         channels = 3,
         break,
      case GL_BGRA:
      case GL_RGBA:
         channels = 4,
         break,
   }

   unsigned int stride = h.gltypesize * channels * width,

   stride = (stride + (pad - 1)) & ~(pad - 1),

   return stride,
}

unsigned int TextureKtx::calculateFaceSize(const TextureKtxHeader &h) {
   unsigned int stride = calculateStride(h, h.pixelwidth),

   return stride * h.pixelheight,
}
```

```
unsigned int TextureKtx::load(const char *filename, unsigned int tex) {
   FILE *fp,
   GLuint temp = 0,
   GLuint retval = 0,
   TextureKtxHeader h,
   size_t data_start, data_end,
   unsigned char *data,
   GLenum target = GL_NONE,

   fp = fopen(filename, "rb"),

   if (!fp) return 0,

   if (fread(&h, sizeof(h), 1, fp) != 1) goto fail_read,

   if (memcmp(h.identifier, &identifier, sizeof(identifier)) != 0) goto fail_header,

   if (h.endianness == 0x04030201) {
      // No swap needed
   } else if (h.endianness == 0x01020304) {
      // Swap needed
      h.endianness = swap32(h.endianness),
      h.gltype = swap32(h.gltype),
      h.gltypesize = swap32(h.gltypesize),
      h.glformat = swap32(h.glformat),
      h.glinternalformat = swap32(h.glinternalformat),
      h.glbaseinternalformat = swap32(h.glbaseinternalformat),
      h.pixelwidth = swap32(h.pixelwidth),
      h.pixelheight = swap32(h.pixelheight),
      h.pixeldepth = swap32(h.pixeldepth),
      h.arrayelements = swap32(h.arrayelements),
      h.faces = swap32(h.faces),
      h.miplevels = swap32(h.miplevels),
      h.keypairbytes = swap32(h.keypairbytes),
   } else {
      goto fail_header,
   }

   // Guess target (texture type)
   if (h.pixelheight == 0) {
      if (h.arrayelements == 0) {
         target = GL_TEXTURE_1D,
      } else {
         target = GL_TEXTURE_1D_ARRAY,
      }
   } else if (h.pixeldepth == 0) {
      if (h.arrayelements == 0) {
         if (h.faces == 0) {
            target = GL_TEXTURE_2D,
         } else {
            target = GL_TEXTURE_CUBE_MAP,
         }
      } else {
         if (h.faces == 0) {
            target = GL_TEXTURE_2D_ARRAY,
         } else {
            target = GL_TEXTURE_CUBE_MAP_ARRAY,
         }
      }
   } else {
      target = GL_TEXTURE_3D,
   }

   // Check for insanity...
   if (target == GL_NONE ||                     // Couldn't figure out target
      (h.pixelwidth == 0) ||                    // Texture has no width???
      (h.pixelheight == 0 && h.pixeldepth != 0))  // Texture has depth but no height???
   {
      goto fail_header,
   }

   temp = tex,
   if (tex == 0) {
      glGenTextures(1, &tex),
   }

   glBindTexture(target, tex),

   data_start = ftell(fp) + h.keypairbytes,
   fseek(fp, 0, SEEK_END),
   data_end = ftell(fp),
   fseek(fp, data_start, SEEK_SET),

   data = new unsigned char[data_end - data_start],
   memset(data, 0, data_end - data_start),

   fread(data, 1, data_end - data_start, fp),

   if (h.miplevels == 0) {
      h.miplevels = 1,
   }

   switch (target) {
      case GL_TEXTURE_1D:
         glTexStorage1D(GL_TEXTURE_1D, h.miplevels, h.glinternalformat, h.pixelwidth),
         glTexSubImage1D(GL_TEXTURE_1D, 0, 0, h.pixelwidth, h.glformat, h.glinternalformat, data),
         break,
      case GL_TEXTURE_2D:
         // glTexImage2D(GL_TEXTURE_2D, 0, h.glinternalformat, h.pixelwidth, h.pixelheight, 0, h.glformat, h.gltype,
         // data),
         if (h.gltype == GL_NONE) {
            glCompressedTexImage2D(GL_TEXTURE_2D, 0, h.glinternalformat, h.pixelwidth, h.pixelheight, 0,
                                   420 * 380 / 2, data),
         } else {
            glTexStorage2D(GL_TEXTURE_2D, h.miplevels, h.glinternalformat, h.pixelwidth, h.pixelheight),
            {
               unsigned char *ptr = data,
               unsigned int height = h.pixelheight,
               unsigned int width = h.pixelwidth,
               glPixelStorei(GL_UNPACK_ALIGNMENT, 1),
               for (unsigned int i = 0, i < h.miplevels, i++) {
                  glTexSubImage2D(GL_TEXTURE_2D, i, 0, 0, width, height, h.glformat, h.gltype, ptr),
                  ptr += height * calculateStride(h, width, 1),
                  height >>= 1,
                  width >>= 1,
                  if (!height) height = 1,
                  if (!width) width = 1,
               }
            }
         }
         break,
      case GL_TEXTURE_3D:
         glTexStorage3D(GL_TEXTURE_3D, h.miplevels, h.glinternalformat, h.pixelwidth, h.pixelheight, h.pixeldepth),
         glTexSubImage3D(GL_TEXTURE_3D, 0, 0, 0, 0, h.pixelwidth, h.pixelheight, h.pixeldepth, h.glformat, h.gltype,
                         data),
         break,
      case GL_TEXTURE_1D_ARRAY:
         glTexStorage2D(GL_TEXTURE_1D_ARRAY, h.miplevels, h.glinternalformat, h.pixelwidth, h.arrayelements),
         glTexSubImage2D(GL_TEXTURE_1D_ARRAY, 0, 0, 0, h.pixelwidth, h.arrayelements, h.glformat, h.gltype, data),
         break,
      case GL_TEXTURE_2D_ARRAY:
         glTexStorage3D(GL_TEXTURE_2D_ARRAY, h.miplevels, h.glinternalformat, h.pixelwidth, h.pixelheight,
                        h.arrayelements),
         glTexSubImage3D(GL_TEXTURE_2D_ARRAY, 0, 0, 0, 0, h.pixelwidth, h.pixelheight, h.arrayelements, h.glformat,
                         h.gltype, data),
         break,
      case GL_TEXTURE_CUBE_MAP:
         glTexStorage2D(GL_TEXTURE_CUBE_MAP, h.miplevels, h.glinternalformat, h.pixelwidth, h.pixelheight),
         // glTexSubImage3D(GL_TEXTURE_CUBE_MAP, 0, 0, 0, 0, h.pixelwidth, h.pixelheight, h.faces, h.glformat,
         // h.gltype, data),
         {
            unsigned int face_size = calculateFaceSize(h),
            for (unsigned int i = 0, i < h.faces, i++) {
               glTexSubImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, 0, 0, h.pixelwidth, h.pixelheight,
                               h.glformat, h.gltype, data + face_size * i),
            }
         }
         break,
      case GL_TEXTURE_CUBE_MAP_ARRAY:
         glTexStorage3D(GL_TEXTURE_CUBE_MAP_ARRAY, h.miplevels, h.glinternalformat, h.pixelwidth, h.pixelheight,
                        h.arrayelements),
         glTexSubImage3D(GL_TEXTURE_CUBE_MAP_ARRAY, 0, 0, 0, 0, h.pixelwidth, h.pixelheight,
                         h.faces * h.arrayelements, h.glformat, h.gltype, data),
         break,
      default:  // Should never happen
         goto fail_target,
   }

   if (h.miplevels == 1) {
      glGenerateMipmap(target),
   }

   width = h.pixelwidth,
   height = h.pixelheight,
   textureTarget = target,
   id = tex,
   retval = tex,

fail_target:
   delete[] data,

fail_header:,
fail_read:
   fclose(fp),

   return retval,
}

bool TextureKtx::save(const char *filename, unsigned int target, unsigned int tex) {
   TextureKtxHeader h,

   memset(&h, 0, sizeof(h)),
   memcpy(h.identifier, &identifier, sizeof(identifier)),
   h.endianness = 0x04030201,

   glBindTexture(target, tex),

   glGetTexLevelParameteriv(target, 0, GL_TEXTURE_WIDTH, (GLint *)&h.pixelwidth),
   glGetTexLevelParameteriv(target, 0, GL_TEXTURE_HEIGHT, (GLint *)&h.pixelheight),
   glGetTexLevelParameteriv(target, 0, GL_TEXTURE_DEPTH, (GLint *)&h.pixeldepth),

   return true,
}

void TextureKtx::use() const { glBindTexture(textureTarget, id), }
```

Mesh object

MeshObject.h

```
#ifndef __OBJECT_H__
#define __OBJECT_H__

#include "sb6mfile.h"

#ifndef SB6M_FILETYPES_ONLY

//#include <GL/glcorearb.h>

class MeshObject {
  public:
   MeshObject();
   ~MeshObject();

   inline void render(unsigned int instanceCount = 1, unsigned int baseInstance = 0) {
      renderSubObject(0, instanceCount, baseInstance);
   }

   void renderSubObject(unsigned int objectIndex, unsigned int instanceCount = 1, unsigned int baseInstance = 0);

   void getSubObjectInfo(unsigned int index, GLuint &first, GLuint &count) {
      if (index >= numSubObjects) {
         first = 0;
         count = 0;
      } else {
         first = subObject[index].first;
         count = subObject[index].count;
      }
   }

   unsigned int getSubObjectCount() const { return numSubObjects; }
   GLuint getVao() const { return vao; }
   void load(const char *filename);
   void free();

  private:
   GLuint dataBuffer;
   GLuint vao;
   GLuint indexType;
   GLuint indexOffset;
   unsigned int numSubObjects;

   enum { MAX_SUB_OBJECTS = 256 };
   SB6M_SUB_OBJECT_DECL subObject[MAX_SUB_OBJECTS];
};

#endif /* SB6M_FILETYPES_ONLY */

#endif /* __OBJECT_H__ */
```

MeshObject.cpp

```
#define _CRT_SECURE_NO_WARNINGS 1

#include "MeshObject.h"

#include <stdio.h>

#include "GL/glew.h"

MeshObject::MeshObject() : dataBuffer(0), indexType(0), vao(0) {}

MeshObject::~MeshObject() {}

void MeshObject::load(const char *filename) {
   FILE *infile = fopen(filename, "rb");
   size_t filesize;
   char *data;

   this->free();

   fseek(infile, 0, SEEK_END);
   filesize = ftell(infile);
   fseek(infile, 0, SEEK_SET);

   data = new char[filesize];

   fread(data, filesize, 1, infile);

   char *ptr = data;
   SB6M_HEADER *header = (SB6M_HEADER *)ptr;
   ptr += header->size;

   SB6M_VERTEX_ATTRIB_CHUNK *vertex_attrib_chunk = NULL;
   SB6M_CHUNK_VERTEX_DATA *vertex_data_chunk = NULL;
   SB6M_CHUNK_INDEX_DATA *index_data_chunk = NULL;
   SB6M_CHUNK_SUB_OBJECT_LIST *sub_object_chunk = NULL;
   SB6M_DATA_CHUNK *data_chunk = NULL;

   unsigned int i;
   for (i = 0; i < header->num_chunks; i++) {
      SB6M_CHUNK_HEADER *chunk = (SB6M_CHUNK_HEADER *)ptr;
      ptr += chunk->size;
      switch (chunk->chunk_type) {
         case SB6M_CHUNK_TYPE_VERTEX_ATTRIBS:
            vertex_attrib_chunk = (SB6M_VERTEX_ATTRIB_CHUNK *)chunk;
            break;
         case SB6M_CHUNK_TYPE_VERTEX_DATA:
            vertex_data_chunk = (SB6M_CHUNK_VERTEX_DATA *)chunk;
            break;
         case SB6M_CHUNK_TYPE_INDEX_DATA:
            index_data_chunk = (SB6M_CHUNK_INDEX_DATA *)chunk;
            break;
         case SB6M_CHUNK_TYPE_SUB_OBJECT_LIST:
            sub_object_chunk = (SB6M_CHUNK_SUB_OBJECT_LIST *)chunk;
            break;
         case SB6M_CHUNK_TYPE_DATA:
            data_chunk = (SB6M_DATA_CHUNK *)chunk;
            break;
         default:
            break;  // goto failed;
      }
   }

   // failed:

   glGenVertexArrays(1, &vao);
   glBindVertexArray(vao);

   if (data_chunk != NULL) {
      glGenBuffers(1, &dataBuffer);
      glBindBuffer(GL_ARRAY_BUFFER, dataBuffer);
      glBufferData(GL_ARRAY_BUFFER, data_chunk->data_length, (unsigned char *)data_chunk + data_chunk->data_offset,
                   GL_STATIC_DRAW);
   } else {
      unsigned int data_size = 0;
      unsigned int size_used = 0;

      if (vertex_data_chunk != NULL) {
         data_size += vertex_data_chunk->data_size;
      }

      if (index_data_chunk != NULL) {
         data_size += index_data_chunk->index_count *
                      (index_data_chunk->indexType == GL_UNSIGNED_SHORT ? sizeof(GLushort) : sizeof(GLubyte));
      }

      glGenBuffers(1, &dataBuffer);
      glBindBuffer(GL_ARRAY_BUFFER, dataBuffer);
      glBufferData(GL_ARRAY_BUFFER, data_size, NULL, GL_STATIC_DRAW);

      if (vertex_data_chunk != NULL) {
         glBufferSubData(GL_ARRAY_BUFFER, 0, vertex_data_chunk->data_size, data + vertex_data_chunk->data_offset);
         size_used += vertex_data_chunk->data_offset;
      }

      if (index_data_chunk != NULL) {
         glBufferSubData(GL_ARRAY_BUFFER, size_used,
                         index_data_chunk->index_count *
                            (index_data_chunk->indexType == GL_UNSIGNED_SHORT ? sizeof(GLushort) : sizeof(GLubyte)),
                         data + index_data_chunk->index_data_offset);
      }
   }

   for (i = 0; i < vertex_attrib_chunk->attrib_count; i++) {
      SB6M_VERTEX_ATTRIB_DECL &attrib_decl = vertex_attrib_chunk->attrib_data[i];
      glVertexAttribPointer(i, attrib_decl.size, attrib_decl.type,
                            attrib_decl.flags & SB6M_VERTEX_ATTRIB_FLAG_NORMALIZED ? GL_TRUE : GL_FALSE,
                            attrib_decl.stride, (GLvoid *)(uintptr_t)attrib_decl.data_offset);
      glEnableVertexAttribArray(i);
   }

   if (index_data_chunk != NULL) {
      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, dataBuffer);
      indexType = index_data_chunk->indexType;
      indexOffset = index_data_chunk->index_data_offset;
   } else {
      indexType = GL_NONE;
   }

   if (sub_object_chunk != NULL) {
      if (sub_object_chunk->count > MAX_SUB_OBJECTS) {
         sub_object_chunk->count = MAX_SUB_OBJECTS;
      }

      for (i = 0; i < sub_object_chunk->count; i++) {
         subObject[i] = sub_object_chunk->subObject[i];
      }

      numSubObjects = sub_object_chunk->count;
   } else {
      subObject[0].first = 0;
      subObject[0].count = indexType != GL_NONE ? index_data_chunk->index_count : vertex_data_chunk->total_vertices;
      numSubObjects = 1;
   }

   delete[] data;

   fclose(infile);

   glBindVertexArray(0);
   glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
}

void MeshObject::free() {
   glDeleteVertexArrays(1, &vao);
   glDeleteBuffers(1, &dataBuffer);

   vao = 0;
   dataBuffer = 0;
}

void MeshObject::renderSubObject(unsigned int objectIndex, unsigned int instanceCount, unsigned int baseInstance) {
   glBindVertexArray(vao);

   if (indexType != GL_NONE) {
      glDrawElementsInstancedBaseInstance(GL_TRIANGLES, subObject[objectIndex].count, indexType,
                                          (void *)subObject[objectIndex].first, instanceCount, baseInstance);
   } else {
      glDrawArraysInstancedBaseInstance(GL_TRIANGLES, subObject[objectIndex].first, subObject[objectIndex].count,
                                        instanceCount, baseInstance);
   }
}
```

Texture coordinates

010_TextureCoordinates.vert

```
#version 450 core

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;

layout (location = 0) in vec4 position;
layout (location = 4) in vec2 tc;

out VS_OUT
{
    vec2 tc;
} vs_out;

void main(void)
{
    // Calculate the world position of each vertex
    vec4 pos_vs = mv_matrix * position;
    // Pass the texture coordinate through unmodified
    vs_out.tc = tc;
    // Compute screen position of vertex
    gl_Position = proj_matrix * pos_vs;
}
```

010_TextureCoordinates.frag

```
#version 450 core

layout (binding = 0) uniform sampler2D tex_object;

in VS_OUT
{
    vec2 tc;
} fs_in;

out vec4 color;

void main(void)
{
    // Read the texture and scale coordinates, then assign color
    color = texture(tex_object, fs_in.tc * vec2(3.0, 1.0));
}
```

Scene_010_TextureCoordinates.h

```
...
private:
    Game *game;
    Shader shader;
    GLuint texObject[2];
    GLuint texIndex;
    Matrix4 transform;
    Matrix4 projection;
    MeshObject object;
```

Scene_010_TextureCoordinates.cpp

```
...
void Scene_010_TextureCoordinates::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));
    Assets::loadTextureKtx("assets/textures/pattern1.ktx", "pattern1");
    #define B 0x00, 0x00, 0x00, 0x00
    #define W 0xFF, 0xFF, 0xFF, 0xFF
    static const GLubyte texData[] =
    {
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
        B, W, B, W, B, W, B, W, B, W, B, W, B, W, B, W,
        W, B, W, B, W, B, W, B, W, B, W, B, W, B, W, B,
    };
    #undef B
    #undef W

    glGenTextures(1, &texObject[0]);
    glBindTexture(GL_TEXTURE_2D, texObject[0]);
    glTexStorage2D(GL_TEXTURE_2D, 1, GL_RGB8, 16, 16);
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 16, 16, GL_RGBA, GL_UNSIGNED_BYTE, texData);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    texObject[1] = Assets::getTextureKtx("pattern1").id;

    shader = Assets::getShader(SHADER_ID(SHADER_NAME));
    projection = Matrix4::createPerspectiveFOV(70.0f, game->windowWidth, game->windowHeight, 0.1f, 1000.0f);
    object.load("assets/meshes/torus_nrms_tc.sbm");
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);

    shader.use();
    shader.setMatrix4("proj_matrix", projection);
}
...

void Scene_010_TextureCoordinates::handleEvent(const InputState &inputState) {
    if(inputState.keyboardState.isJustPressed(SDL_SCANCODE_T)) {
        texIndex = ++texIndex % 2;
    }
}

void Scene_010_TextureCoordinates::update(float dt) {
    float t = Timer::getTimeSinceStart();
    Quaternion rotY { Vector3::unitY, t * Maths::toRadians(19.3f) };
    Quaternion rotZ { Vector3::unitZ, t * Maths::toRadians(21.1f) };
    Quaternion rotation = Quaternion::concatenate(rotY, rotZ);
    transform = Matrix4::createTranslation(Vector3(0.0f, 0.0f, -3.0f)) * rotation.asMatrix();
}

void Scene_010_TextureCoordinates::draw()
{
    static const GLfloat bgColor[] = {0.0f, 0.0f, 0.2f, 1.0f};
    static const GLfloat ones[] = { 1.0f };
    glClearBufferfv(GL_COLOR, 0, bgColor);
    glClearBufferfv(GL_DEPTH, 0, ones);
    glBindTexture(GL_TEXTURE_2D, texObject[texIndex]);

    shader.setMatrix4("mv_matrix", transform);

    object.render();
}
```

MipMap Tunnel

011_MipmapTunnel.vert

```
#version 450 core

uniform mat4 mvp_matrix;
uniform float offset;

out VS_OUT
{
    vec2 tc;
} vs_out;

void main(void)
{
    const vec2[4] position = vec2[4](vec2(0.5, 0.5),
                                     vec2( 0.5, -0.5),
                                     vec2(-0.5,  0.5),
                                     vec2( 0.5,  0.5));

    vs_out.tc = (position[gl_VertexID].xy + vec2(offset, 0.5)) * vec2(30.0, 1.0);
    gl_Position = mvp_matrix * vec4(position[gl_VertexID], 0.0, 1.0);
}
```

011_MipmapTunnel.frag

```
#version 450 core

layout (location = 0) out vec4 color;

in VS_OUT
{
    vec2 tc;
} fs_in;

layout (binding = 0) uniform sampler2D tex;

void main(void)
{
    color = texture(tex, fs_in.tc);
}
```

Scene_011_MipmapTunnel.h

```
...
private:
    Game *game;
    Shader shader;
    GLuint vao;
    GLuint texFloor, texWall, texCeiling;
    Matrix4 transform[4];
    Matrix4 projection;
```

Scene_011_MipmapTunnel.cpp

```
...
void Scene_011_MipmapTunnel::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);
    Assets::loadTextureKtx("assets/textures/brick.ktx", "brick");
    Assets::loadTextureKtx("assets/textures/ceiling.ktx", "ceiling");
    Assets::loadTextureKtx("assets/textures/floor.ktx", "floor");
    texWall = Assets::getTextureKtx("brick").id;
    texCeiling = Assets::getTextureKtx("ceiling").id;
    texFloor = Assets::getTextureKtx("floor").id;
    projection = Matrix4::createPerspectiveFOV(70.0f, game->windowWidth, game->windowHeight, 0.1f, 1000.0f);
    shader = Assets::getShader(SHADER_ID(SHADER_NAME));

    int i;
    GLuint textures[] = { texFloor, texWall, texCeiling };
    for (i = 0; i < 3; i++)
    {
        glBindTexture(GL_TEXTURE_2D, textures[i]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    }
    shader.use();
}
...
void Scene_011_MipmapTunnel::update(float dt) {
    float t = Timer::getTimeSinceStart();
    int i;
    GLuint textures[] = { texWall, texFloor, texWall, texCeiling };
    for (i = 0; i < 4; i++)
    {
        transform[i] = Matrix4::createRotationZ(Maths::piOver2 * static_cast<float>(i))
            * Matrix4::createTranslation(Vector3(0.5f, 0.0f, 10.0f))
            * Matrix4::createRotationY(Maths::piOver2)
            * Matrix4::createScale(Vector3(30.0f, 1.0f, 1.0f));
    }
}

void Scene_011_MipmapTunnel::draw()
{
    float t = Timer::getTimeSinceStart();
    shader.setFloat("offset", t * 0.003f);
    int i;
    GLuint textures[] = { texWall, texFloor, texWall, texCeiling };
    for (i = 0; i < 4; i++)
    {
        Matrix4 mvp = projection * transform[i];
        shader.setMatrix4("mvp_matrix", mvp);
        glBindTexture(GL_TEXTURE_2D, textures[i]);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    }
}
```

Optimization: texture arrays

Arrays of textures

Read the beginning of 06.ArrayTexture, about this specific point.

The rest of the text is not implemented in this lecture and is left as additional information. You can still implement it if you want : the meshes and texture are included in the source code.

2D array texture

012_ArrayTextureAlienRain.vert

```
#version 450 core

layout (location = 0) in int alien_index;

out VS_OUT
{
    flat int alien;
    vec2 tc;
} vs_out;

struct droplet_t
{
    float x_offset;
    float y_offset;
    float orientation;
    float unused;
};

layout (std140) uniform droplets
{
    droplet_t droplet[256];
};

void main(void)
{
    const vec2[4] position = vec2[4](vec2(-0.5, -0.5),
                                     vec2( 0.5, -0.5),
                                     vec2(-0.5,  0.5),
                                     vec2( 0.5,  0.5));
    vs_out.tc = position[gl_VertexID].xy + vec2(0.5);
    float co = cos(droplet[alien_index].orientation);
    float so = sin(droplet[alien_index].orientation);
    mat2 rot = mat2(vec2(co, so),
                    vec2(-so, co));
    vec2 pos = 0.25 * rot * position[gl_VertexID];
    gl_Position = vec4(pos.x + droplet[alien_index].x_offset,
                       pos.y + droplet[alien_index].y_offset,
                       0.5, 1.0);
    vs_out.alien = alien_index % 64;
}
```

012_ArrayTextureAlienRain.frag

```
#version 450 core

layout (location = 0) out vec4 color;

in VS_OUT
{
    flat int alien;
    vec2 tc;
} fs_in;

uniform sampler2DArray texAliens;

void main(void)
{
    color = texture(texAliens, vec3(fs_in.tc, float(fs_in.alien)));
}
```

Scene_012_ArrayTextureAlienRain.h

```
...
private:
    Game *game;
    Shader shader;
    GLuint vao;
    GLuint texAliens;
    GLuint rainBuffer;
    float dropletXOffset[256];
    float dropletRotSpeed[256];
    float dropletFallSpeed[256];

    Matrix4 transform[4];
    Matrix4 projection;
};

static unsigned int seed = 0x13371337;
static inline float randomFloat()
{
    float res;
    unsigned int tmp;

    seed *= 16807;
    tmp = seed ^ (seed >> 4) ^ (seed << 15);
    *((unsigned int *) &res) = (tmp >> 9) | 0x3F800000;

    return (res - 1.0f);
}
```

Scene_012_ArrayTextureAlienRain.cpp

```
...
void Scene_012_ArrayTextureAlienRain::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    Assets::loadTextureKtx("assets/textures/aliens.ktx", "aliens");
    texAliens = Assets::getTextureKtx("aliens").id;
    glBindTexture(GL_TEXTURE_2D_ARRAY, texAliens);
    glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

    glGenBuffers(1, &rainBuffer);
    glBindBuffer(GL_UNIFORM_BUFFER, rainBuffer);
    glBufferData(GL_UNIFORM_BUFFER, 256 * sizeof(Vector4), NULL, GL_DYNAMIC_DRAW);

    for (int i = 0; i < 256; i++)
    {
        dropletXOffset[i] = randomFloat() * 2.0f - 1.0f;
        dropletRotSpeed[i] = (randomFloat() + 0.5f) * ((i & 1) ? -3.0f : 3.0f);
        dropletFallSpeed[i] = randomFloat() + 0.2f;
    }
    //projection = Matrix4::createPerspectiveFOV(70.0f, game->windowWidth, game->windowHeight, 0.1f, 1000.0f);
    shader = Assets::getShader(SHADER_ID(SHADER_NAME));

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    shader.use();
}
...
void Scene_012_ArrayTextureAlienRain::draw()
{
    float t = Timer::getTimeSinceStart();
    glBindBufferBase(GL_UNIFORM_BUFFER, 0, rainBuffer);
    Vector4 * droplet = (Vector4 *)glMapBufferRange(GL_UNIFORM_BUFFER, 0, 256 * sizeof(Vector4), GL_MAP_WRITE_BIT |
GL_MAP_INVALIDATE_BUFFER_BIT);

    for (int i = 0; i < 256; i++)
    {
        droplet[i][0] = dropletXOffset[i];
        droplet[i][1] = 2.0f - fmodf((t + float(i)) * dropletFallSpeed[i], 4.31f);
        droplet[i][2] = t * dropletRotSpeed[i];
        droplet[i][3] = 0.0f;
    }
    glUnmapBuffer(GL_UNIFORM_BUFFER);

    int alienIndex;
    for (alienIndex = 0; alienIndex < 256; alienIndex++)
    {
        glVertexAttribI1i(0, alienIndex);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    }
}
```

Optimization: vertex processing and drawing commands

Instanced grass blades

013_InstancedGrassBlades.vert

```
#version 450 core

// Incoming per vertex position
in vec4 vVertex;

// Output varyings
out vec4 color;

uniform mat4 mvp_matrix;
uniform float time;

layout (binding = 0) uniform sampler1D grasspalette_texture;
layout (binding = 1) uniform sampler2D length_texture;
layout (binding = 2) uniform sampler2D orientation_texture;
layout (binding = 3) uniform sampler2D grasscolor_texture;
layout (binding = 4) uniform sampler2D bend_texture;

int random(int seed, int iterations)
{
    int value = seed;
    int n;

    for (n = 0; n < iterations; n++) {
        value = ((value >> 7) ^ (value << 9)) * 15485863;
    }

    return value;
}

vec4 random_vector(int seed)
{
    int r = random(gl_InstanceID, 4);
    int g = random(r, 2);
    int b = random(g, 2);
    int a = random(b, 2);

    return vec4(float(r & 0x3FF) / 1024.0,
                float(g & 0x3FF) / 1024.0,
                float(b & 0x3FF) / 1024.0,
                float(a & 0x3FF) / 1024.0);
}

mat4 construct_rotation_matrix(float angle)
{
    float st = sin(angle);
    float ct = cos(angle);

    return mat4(vec4(ct, 0.0, st, 0.0),
                vec4(0.0, 1.0, 0.0, 0.0),
                vec4(-st, 0.0, ct, 0.0),
                vec4(0.0, 0.0, 0.0, 1.0));
}

void main(void)
{
    vec4 offset = vec4(float(gl_InstanceID >> 10) - 512.0,
                       0.0f,
                       float(gl_InstanceID & 0x3FF) - 512.0,
                       0.0f);
    int number1 = random(gl_InstanceID, 3);
    int number2 = random(number1, 2);
    offset += vec4(float(number1 & 0xFF) / 256.0,
                   0.0f,
                   float(number2 & 0xFF) / 256.0,
                   0.0f);

    vec2 texcoord = offset.xz / 1024.0 + vec2(0.5);

    float bend_factor = texture(bend_texture, texcoord).r * 2.0;
    float bend_amount = cos(vVertex.y);

    // Make grass angle change with texture move
    vec2 texcoord_angle = texcoord + time * 0.05;
    float angle = texture(orientation_texture, texcoord_angle).r * 2.0 * 3.141592;

    mat4 rot = construct_rotation_matrix(angle);
    vec4 position = (rot * (vVertex + vec4(0.0, 0.0, bend_amount * bend_factor, 0.0))) + offset;

    position *= vec4(1.0, texture(length_texture, texcoord).r * 0.9 + 0.3, 1.0, 1.0);

    gl_Position =  mvp_matrix * position;
    color = texture(grasspalette_texture, texture(grasscolor_texture, texcoord).r) +
            vec4(random_vector(gl_InstanceID).xyz * vec3(0.1, 0.5, 0.1), 1.0);
}
```

013_InstancedGrassBlades.frag

```
#version 450 core

in vec4 color;
out vec4 output_color;

void main(void)
{
    output_color = color;
}
```

Scene_013_InstancedGrassBlades.h

```
...
private:
    Game *game;
    Shader shader;
    GLuint vao;
    GLuint buffer;
    GLuint texGrassColor;
    GLuint texGrassLength;
    GLuint texGrassOrientation;
    GLuint texGrassBend;

    Matrix4 view;
    Matrix4 projection;
};

static unsigned int seed = 0x13371337;
static inline float randomFloat()
{
    float res;
    unsigned int tmp;

    seed *= 16807;
    tmp = seed ^ (seed >> 4) ^ (seed << 15);
    *((unsigned int *) &res) = (tmp >> 9) | 0x3F800000;

    return (res - 1.0f);
}
```

Scene_013_InstancedGrassBlades.cpp

```
...
void Scene_013_InstancedGrassBlades::load() {
    std::srand((int) std::time(nullptr));
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    static const GLfloat grassBlade[] =
    {
        -0.3f, 0.0f,
        0.3f, 0.0f,
        -0.20f, 1.0f,
        0.1f, 1.3f,
        -0.05f, 2.3f,
        0.0f, 3.3f
    };

    glGenBuffers(1, &buffer);
    glBindBuffer(GL_ARRAY_BUFFER, buffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(grassBlade), grassBlade, GL_STATIC_DRAW);

    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, NULL);
    glEnableVertexAttribArray(0);

    glActiveTexture(GL_TEXTURE1);
    Assets::loadTextureKtx("assets/textures/grass_length.ktx", "grass_length");
    texGrassLength = Assets::getTextureKtx("grass_length").id;
    glBindTexture(GL_TEXTURE_2D, texGrassLength);

    glActiveTexture(GL_TEXTURE2);
    Assets::loadTextureKtx("assets/textures/grass_orientation.ktx", "grass_orientation");
    texGrassOrientation = Assets::getTextureKtx("grass_orientation").id;
    glBindTexture(GL_TEXTURE_2D, texGrassOrientation);

    glActiveTexture(GL_TEXTURE3);
    Assets::loadTextureKtx("assets/textures/grass_color.ktx", "grass_color");
    texGrassColor = Assets::getTextureKtx("grass_color").id;
    glBindTexture(GL_TEXTURE_2D, texGrassColor);

    glActiveTexture(GL_TEXTURE4);
    Assets::loadTextureKtx("assets/textures/grass_bend.ktx", "grass_bend");
    texGrassBend = Assets::getTextureKtx("grass_bend").id;
    glBindTexture(GL_TEXTURE_2D, texGrassBend);

    projection = Matrix4::createPerspectiveFOV(70.0f, game->windowWidth, game->windowHeight, 0.1f, 1000.0f);
    shader = Assets::getShader(SHADER_ID(SHADER_NAME));
    shader.use();
}
...
void Scene_013_InstancedGrassBlades::update(float dt) {
    float t = Timer::getTimeSinceStart() * 0.20f + 1.0f;
    float r = 20.0f;
    shader.setFloat("time", t);
    view = Matrix4::createTranslation(Vector3(0, -25, 0))
         * Matrix4::createLookAt(Vector3(Maths::sin(t) * r, 25.0f, Maths::cos(t) * r),
                                 Vector3(0.0f, 15.0f, 10.0f),
                                 Vector3::unitY);
}

void Scene_013_InstancedGrassBlades::draw()
{
    shader.setMatrix4("mvp_matrix", projection * view);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glBindVertexArray(vao);
    glDrawArraysInstanced(GL_TRIANGLE_STRIP, 0, 6, 1024 * 1024);
}
```

Indirect draw : astoids belt

014_IndirectDrawAstroids.vert

```
#version 410 core

layout (location = 0) in vec3 position_3;
layout (location = 1) in vec3 normal;

layout (location = 10) in uint draw_id;

out VS_OUT
{
    vec3 normal;
    vec4 color;
} vs_out;

uniform float time = 0.0;

uniform mat4 view_matrix;
uniform mat4 proj_matrix;
uniform mat4 viewproj_matrix;

const vec4 color0 = vec4(0.29, 0.21, 0.18, 1.0);
const vec4 color1 = vec4(0.58, 0.55, 0.51, 1.0);

void main(void)
{
    vec4 position = vec4(position_3, 1.0);
    mat4 m1;
    mat4 m2;
    mat4 m;
    float t = time * 0.1;
    float f = float(draw_id) / 30.0;

    float st = sin(t * 0.5 + f * 5.0);
    float ct = cos(t * 0.5 + f * 5.0);

    float j = fract(f);
    float d = cos(j * 3.14159);

    // Rotate around Y
    m1[0] = vec4(ct, 0.0, st, 0.0);
    m1[1] = vec4(0.0, 1.0, 0.0, 0.0);
    m1[2] = vec4(-st, 0.0, ct, 0.0);
    m1[3] = vec4(0.0, 0.0, 0.0, 1.0);

    // Translate in the XZ plane
    m1[0] = vec4(1.0, 0.0, 0.0, 0.0);
    m1[1] = vec4(0.0, 1.0, 0.0, 0.0);
    m1[2] = vec4(0.0, 0.0, 1.0, 0.0);
    m1[3] = vec4(260.0 + 30.0 * d, 5.0 * sin(f * 123.123), 0.0, 1.0);

    m = m * m1;

    // Rotate around X
    st = sin(t * 2.1 * (600.0 + f) * 0.01);
    ct = cos(t * 2.1 * (600.0 + f) * 0.01);

    m1[0] = vec4(ct, st, 0.0, 0.0);
    m1[1] = vec4(-st, ct, 0.0, 0.0);
    m1[2] = vec4(0.0, 0.0, 1.0, 0.0);
    m1[3] = vec4(0.0, 0.0, 0.0, 1.0);

    m = m * m1;

    // Rotate around Z
    st = sin(t * 1.7 * (700.0 + f) * 0.01);
    ct = cos(t * 1.7 * (700.0 + f) * 0.01);

    m1[0] = vec4(1.0, 0.0, 0.0, 0.0);
    m1[1] = vec4(0.0, ct, st, 0.0);
    m1[2] = vec4(0.0, -st, ct, 0.0);
    m1[3] = vec4(0.0, 0.0, 0.0, 1.0);

    m = m * m1;

    // Non-uniform scale
    float f1 = 0.65 + cos(f * 1.1) * 0.2;
    float f2 = 0.65 + cos(f * 1.1) * 0.2;
    float f3 = 0.65 + cos(f * 1.3) * 0.2;

    m1[0] = vec4(f1, 0.0, 0.0, 0.0);
    m1[1] = vec4(0.0, f2, 0.0, 0.0);
    m1[2] = vec4(0.0, 0.0, f3, 0.0);
    m1[3] = vec4(0.0, 0.0, 0.0, 1.0);

    m = m * m1;

    gl_Position = viewproj_matrix * m * position;
    vs_out.normal = mat3(view_matrix * m) * normal;
    vs_out.color = mix(color0, color1, fract(j * 313.431));
}
```

014_IndirectDrawAstroids.frag

```
#version 410 core

out vec4 color;

in VS_OUT
{
    vec3 normal;
    vec4 color;
} fs_in;

void main(void)
{
    vec3 N = normalize(fs_in.normal);

    color = fs_in.color * abs(N.z);
}
```

Scene_014_IndirectDrawAstroids.h

```
...

struct DrawArraysIndirectCommand
{
    GLuint count;
    GLuint primCount;
    GLuint first;
    GLuint baseInstance;
};

enum class Mode
{
    MODE_FIRST,
    MODE_MULTIDRAW = 0,
    MODE_SEPARATE_DRAWS,
    MODE_MAX = MODE_SEPARATE_DRAWS
};

class Scene_014_IndirectDrawAstroids : public Scene
{

private:
    Game *game;
    Shader shader;

    MeshObject object;
    GLuint indirectDrawBuffer;
    GLuint drawIndexBuffer;
    Mode mode;
    bool paused;
    bool vsync;
    float t;

    Matrix4 view;
    Matrix4 projection;
};
```

Scene_014_IndirectDrawAstroids.cpp

```
...
void Scene_014_IndirectDrawAstroids::load() {
    Assets::loadShader(SHADER_VERT(SHADER_NAME), SHADER_FRAG(SHADER_NAME), "", "", "", SHADER_ID(SHADER_NAME));

    object.load("assets/meshes/asteroids.sbm");
    mode = Mode::MODE_MULTIDRAW;
    paused = false;
    vsync = false;
    t = 0;

    glGenBuffers(1, &indirectDrawBuffer);
    glBindBuffer(GL_DRAW_INDIRECT_BUFFER, indirectDrawBuffer);
    glBufferData(GL_DRAW_INDIRECT_BUFFER,
                 NUM_DRAWS * sizeof(DrawArraysIndirectCommand),
                 NULL,
                 GL_STATIC_DRAW);

    DrawArraysIndirectCommand * cmd = (DrawArraysIndirectCommand*) glMapBufferRange(GL_DRAW_INDIRECT_BUFFER,
                                                                                    0,
                                                                                    NUM_DRAWS * sizeof(DrawArraysIndirectCommand),
                                                                                    GL_MAP_WRITE_BIT | GL_MAP_INVALIDATE_BUFFER_BIT);

    for (int i = 0; i < NUM_DRAWS; i++)
    {
        object.getSubObjectInfo(i % object.getSubObjectCount(),
                                 cmd[i].first,
                                 cmd[i].count);
        cmd[i].primCount = 1;
        cmd[i].baseInstance = i;
    }

    glUnmapBuffer(GL_DRAW_INDIRECT_BUFFER);

    glBindVertexArray(object.getVao());
    glGenBuffers(1, &drawIndexBuffer);
    glBindBuffer(GL_ARRAY_BUFFER, drawIndexBuffer);
    glBufferData(GL_ARRAY_BUFFER,
                 NUM_DRAWS * sizeof(GLuint),
                 NULL,
                 GL_STATIC_DRAW);

    GLuint * drawIndex =
        (GLuint *)glMapBufferRange(GL_ARRAY_BUFFER,
                                   0,
                                   NUM_DRAWS * sizeof(GLuint),
                                   GL_MAP_WRITE_BIT |
                                   GL_MAP_INVALIDATE_BUFFER_BIT);

    for (int i = 0; i < NUM_DRAWS; i++)
    {
        drawIndex[i] = i;
    }

    glUnmapBuffer(GL_ARRAY_BUFFER);

    glVertexAttribIPointer(10, 1, GL_UNSIGNED_INT, 0, NULL);
    glVertexAttribDivisor(10, 1);
    glEnableVertexAttribArray(10);

    projection = Matrix4::createPerspectiveFOV(50.0f, game->windowWidth, game->windowHeight, 1.0f, 2000.0f);
    shader = Assets::getShader(SHADER_ID(SHADER_NAME));
    shader.use();
}
...
void Scene_014_IndirectDrawAstroids::update(float dt) {
    int i;
    static const float one = 1.0f;
    static const float black[] = { 0.0f, 0.0f, 0.0f, 0.0f };

    t += dt;

    view = Matrix4::createLookAt(Vector3(100.0f * Maths::cos(t * 0.023f), 100.0f * Maths::cos(t * 0.023f), 300.0f * Maths::sin(t * 0.037f) - 600.0f),
                                 Vector3(0.0f, 0.0f, 260.0f),
                                 Vector3::normalize(Vector3(0.1f - Maths::cos(t * 0.1f) * 0.3f, 1.0f, 0.0f)));

    shader.use();
    shader.setFloat("time", t);
    shader.setMatrix4("view_matrix", view);
    shader.setMatrix4("proj_matrix", projection);
    shader.setMatrix4("viewproj_matrix", projection * view);
}

void Scene_014_IndirectDrawAstroids::draw()
{
    static const float one = 1.0f;
    static const float black[] = { 0.0f, 0.0f, 0.0f, 0.0f };
    glClearBufferfv(GL_COLOR, 0, black);
    glClearBufferfv(GL_DEPTH, 0, &one);

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);

    glEnable(GL_CULL_FACE);

    glBindVertexArray(object.getVao());

    if (mode == Mode::MODE_MULTIDRAW)
    {
        glMultiDrawArraysIndirect(GL_TRIANGLES, NULL, NUM_DRAWS, 0);
    }
    else if (mode == Mode::MODE_SEPARATE_DRAWS)
    {
        for (int j = 0; j < NUM_DRAWS; j++)
        {
            GLuint first, count;
            object.getSubObjectInfo(j % object.getSubObjectCount(), first, count);
            glDrawArraysInstancedBaseInstance(GL_TRIANGLES,
                                              first,
                                              count,
                                              1, j);
        }
    }
}
```

Follow-up in next lecture !