



UNIVERSITY OF SILESIA
IN KATOWICE



Real-time Graphics Project



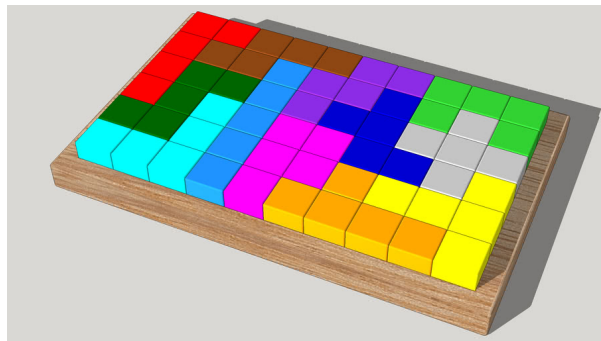
The aim of the project

For the RTG module, we have been asked to carry out a project in order to validate the module, this project must bring together various notions studied during the semester.

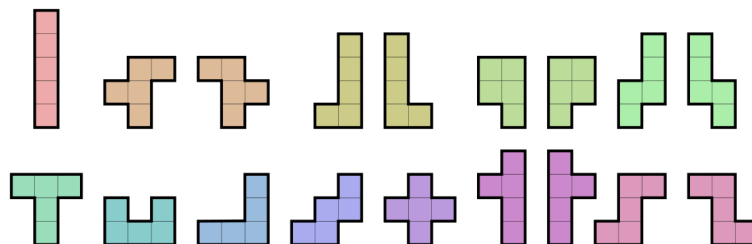
This project will be a **Tetris** game, with the possibility of moving the arena and zoom it in order to have a better view.

Tetris

The game Tetris was inspired by the game *Pentominos*, but the principle of this game being very closed consisting in putting the pieces of the puzzle in the right direction in order to constitute a full and closed circuit.



Pajitnov has taken the same concept and used the *Pentominos* pieces to integrate them into a game aligned in a vertical zone where the pieces fall from the top to the bottom.



The first version of the game was released on Electronika 60, but it was on the IBM PC that the game succeeded in being exported and attracting the attention of investors outside the Soviet Union.



The aim of the project is to recreate the **Tetris** game in a context with the same rules as the genuine.

Presentation video

A [video](#) has been prepared to demonstrate how the game works.

Software description

In order to design this game, many choices were made to have a cleaner and more efficient code.

- Abstraction
- OOP Functions
- Objects
- Shaders
- Game
- States
- Keys
- Sound

Abstraction

This C++ related feature allows us to group together a group of features belonging to a specific object (class), so they will be more easy to identify.

```
// Headers
#include "Game.h"

Game::Game(){
```

```

        // Constructor
    }

    Game::~Game(){
        // Destructor
    }

    Game::Init(){
        // Class function
    }

```

OOP Functions

Object-oriented programming is an increasingly popular method of computer programming. Organized around objects, or data, object-oriented programming offers many advantages.

```

// Function to get vertex buffer objects
GLuint Cube::GetVbo() const
{
    return vbo;
}

// Getting automatically vbos onto the cube class
void Shader::Render(const Camera& camera, Cube* cubes, GLuint cubes_count)
{
    ...

    for (GLuint i = 0; i < cubes_count; i++)
    {
        ...

        // Binding VBO
        glBindBuffer(GL_ARRAY_BUFFER, cubes[i].GetVbo());

        ...
    }

    ...
}

```

Objects

Each object has an impact and are complementary to each other, for example, the "Game" object forms a group of objects such as "Cube", "Shader", "Camera" etc... because this class contains the main functions to configure and set up the game, and like each object it includes functions that are specific to it but which in this case depend on functions belonging to other objects.

Shaders

Shaders are short programs that are loaded into the GPU. These programs each treat a specific step of the queue, they're independent and only communicate with each other through their inputs/outputs.

Some programs need to be executed on the GPU to take full advantage of these performances, that's why we need to link our program to GLSL files in our case a Vertex Shader and Fragment Shader file (knowing that other types of shaders exist).

Vertex shader

It runs first, receives attributes and then calculate (manipulate) the position of each individual vertex.

The model, view, and projection matrices are three different matrices. The model is used to move from the local coordinate space of an object to world space, the view from world space to camera space, the projection from camera to screen, if you compound these three matrices, you can use the same output to go from object space to display space, enabling you to figure out what to pass to the next phase of a pipeline programmable from the incoming vertices locations.

And we can manage this thanks to uniforms that make the link between our GLSL and C++ code.

```
#version 460 core

// Index attribution
layout(location = 0) in vec3 in_vertex_pos;
layout(location = 1) in vec3 in_vertex_color;

uniform mat4 view_projection;
uniform mat4 model_matrix;

out vec3 fragment_color;

void main() {
    gl_Position = view_projection * model_matrix * vec4(in_vertex_pos, 1);
    fragment_color = in_vertex_color;
}
```

Fragment shader

It runs second, it sets the color of each individual fragment of the geometry.

```
#version 460 core

in vec3 fragment_color;

out vec3 color;

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

Game

The goal of the game is to line up a complete row in order to make it disappear, otherwise the arena will fill up and cause the player to lose.

States

Collision

In this game, thanks to the concept of collision we will prevent each cube to pass through the other after the detection

Rotation

This state will allow the cube to change form because the principle of Tetris is to be able to change the orientation of the shape so that it can enter the associated empty space

Falling

The cubes fall to land on the field so that they are stacked on top of each other.

Disappearing

As mentioned before, once a horizontal line is formed, the cubes will (the line) disappear.

Keys

- Event Processing
- Keyboard Callback
- Keyboard Press Handler


Event Processing

GLFW needs to interrogate the window system in order to furnish entries to the application.

Keyboard Callback

Its main task is to attach each event to a specific key, for example when the A key is pressed, the event is recorded and the function `KeyPressHandler();` is called.

Keyboard Press Handler

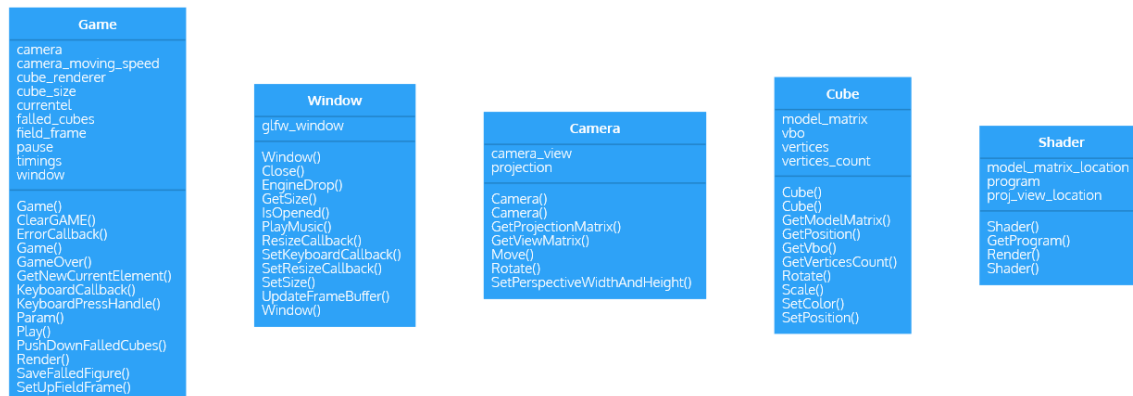
This function links each click to the event it is supposed to trigger, e.g. if you press , the current element will go to the right.

Sound

To be able to play audio (background music, sound effects linked to the actions) I chose to use irrKlang which is an intuitive library.

Its startup requires the creation of an instance of sound engine, after that it is necessary thanks to the function `Play2D()` to pass the audio file and the wished parameters like the autoplay or a looped playing.











Class Diagram



User guide

Make sure you're using a QWERTY keyboard in order to match with these keys

Control keys

	Move camera left		Zoom in camera
	Move camera right		Zoom out camera
	Rotate clockwise		Bring to the bottom
	Move to the right		Move to the left
	Pause / Resume the game		Exit the game

Libraries

- **OpenGL:** is a standardised set of functions for computing 2D or 3D images launched by Silicon Graphics in 1992. This programming interface is available on many platforms where it is used for applications ranging from video games to CAD to modelling. *[from Wikipedia]*
 - **GLEW:** groups all the functionality of the OpenGL kernel into a single file. This makes it easy to find the information required for each extension at a glance. In addition, the library allows extensions to be loaded and is fully compatible with OpenGL extensions. To initialise GLEW, a valid OpenGL rendering context must first be created.
 - **GLFW:** is an Open Source, multi-platform library for OpenGL, OpenGL ES and Vulkan development on the desktop. It provides a simple API for creating windows, contexts and surfaces, receiving input and events. *[from glfw.org]*
 - **GLM:** is a C++ mathematical library for 3D applications or games based on the GLSL specifications. Therefore it offers a syntax of mathematical operators very close to those found in GLSL.
 - **irrKlang:** is a C++ library for audio management, it allows to play mp3, ogg and wav sounds in a 3D space.
-

Resources

- [learnopengl](#)
 - [Sonar System](#)
 - [The Cherno](#)
 - [open.gl](#)
 - [docs.gl](#)
 - [Carbon](#)
 - [Adobe XD](#)
-

Files Architecture

```
RTG_Final_Project
├── doc
│   ├── assets (resources for docs)
│   ├── Case Study.pdf (automations of the game)
│   ├── Project DEMO.mp4 (video presentation of the game)
│   ├── Real-time Graphics Project.md (report in Markdown made with Typora)
│   ├── Real-time Graphics Project.md (report in PDF)
│   └── Real-time Graphics Project.html (report in HTML)
└── src
    ├── bin
    │   └── x64
    │       └── glew32.dll
```

```
| | | └─ glfw3.dll
| | | └─ ikpMP3.dll
| | | └─ irrKlang.dll
| | | └─ SilesianTetris.exe (executable file)
| └─ Game
| | └─ Shaders
| | | └─ FragmentShader.cpp
| | | └─ VertexShader.cpp
| | └─ Camera.cpp
| | └─ Camera.h
| | └─ Coordinates.h
| | └─ Cube.cpp
| | └─ Cube.h
| | └─ CurrentElement.h
| | └─ Game.cpp
| | └─ Game.h
| | └─ main.cpp
| | └─ Shader.cpp
| | └─ Shader.h
| | └─ Window.cpp
| | └─ Window.h
| └─ Include
| | └─ audio (audio tracks for the game)
| | | └─ selection.mp3
| | | └─ tetris.mp3
| | └─ GL
| | └─ GLFW
| | └─ GLM
| | └─ irrKlang
| └─ Lib
| | └─ x64
| | | └─ glew32.lib
| | | └─ glew32s.lib
| | | └─ glfw3.lib
| | | └─ glfw3dll.lib
| | | └─ irrKlang.lib
```