B4 - Object-Oriented Programming

B-OOP-400

Raytracer

How to create your shared library

Raytracer

binary name: raytracer
build tool: CMakeLists.txt

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

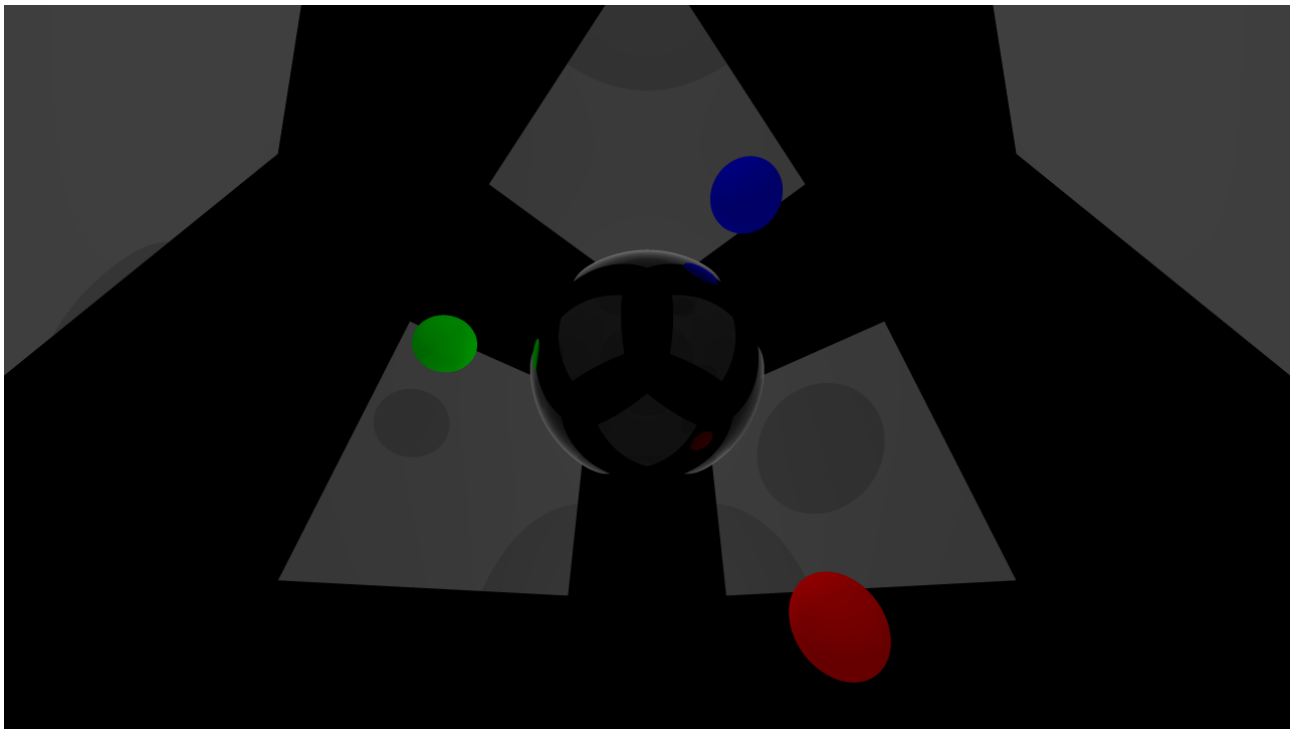

Figure 1: Raytracer_image

How to create your shared library

What is a shared library ?

A shared library is a library that can be linked to a program at run time. It is loaded into memory when the program is executed and then shared between the program and other programs that need it.

Why create a shared library ?

For our Raytracer project we had to implement lights (ambient light, directional light), primitives (sphere, plane, cone…) and materials (flat color…) in dynamic libraries (.so) to be able to create scenes and create objects in this scenes with configuration files. So the program will read the configuration files in order to create the scenes with the objects in a dynamic way. If you want to add a new object, you just have to create a new dynamic library and add it to the configuration file.

PDF to configuration files

Create your shared library for the Raytracer project

First step

First you will have to create a folder according to what you want to add in primitives for shapes, lights for light and materials for a material.

Second step

Then add several files to this folder like the CMakeLists.txt, which will allow you to compile your object in (.so), you will just have to change the name of the libraries you want to give it.

```
file(GLOB SRC_OBJECT_TO_CREATE  *.cpp)

add_library(name_library SHARED ${SRC_OBJECT_TO_CREATE } ${SRC_SHARED})

target_include_directories(name_library PUBLIC ${CMAKE_CURRENT_SOURCE_DIR}/../../
    shared/)

set_target_properties(name_library PROPERTIES PREFIX "")

set_target_properties(name_library PROPERTIES LIBRARY_OUTPUT_DIRECTORY ${
    PROJECT_SOURCE_DIR}/plugins)

target_link_libraries(name_library)
```

Third step

Create the entrypoint of your object, for example for a sphere you will have to create a sphere.cpp file. In this file you will have to create a function that will return a pointer to your object.

```
extern "C" std::unique_ptr<RayTracer::IObject> getInstance()
{
    return std::make_unique<RayTracer::ClassName>();
}

extern "C" std::unique_ptr<RayTracer::pluginType_t> getType()
{
    return std::make_unique<RayTracer::pluginType_t>(RayTracer::pluginType_t::OBJECT)
        ;
}

extern "C" std::unique_ptr<std::string> getName()
{
    return std::make_unique<std::string>("object_name");
}
```

This example is for a primitive, for a light you will have to change the pluginType_t to LIGHT and for a material to MATERIAL.

Fourth step

Then you will have to create a header file for your object, for example for a sphere you will have to create a sphere.hpp file and a sphere.cpp file. In the header file you will have to declare the class and the methods that will be used in the cpp file. and add the methods you want from the AObject class.
You obviously can override the methods that you can find in IOject.hpp, ILight.hpp and IMaterial.hpp.

```
namespace RayTracer {
    class ObjectName : public AObject {
        public:

            ObjectName();

            bool hits(const RayTracer::Ray &ray, RayHit &hit) const override;

            virtual double getRadius() const noexcept;

            void setRadius(double radius) noexcept;
        private:
            double _radius;
    };
}
```

You must have to override the methods

Fifth step

Then you will have to create a header file for your object you have to create the methods that you have declared in the header file. create a cpp file for your object, for example for a sphere you will have to create a sphere.cpp file. You obviously can override the methods that you can find in IObject.hpp, ILight.hpp and IMaterial.hpp.

```
RayTracer::ObjectName::ObjectName()
    : _radius(1)
{
}

bool RayTracer::ObjectName::hits(const RayTracer::Ray &ray, RayHit &hit) const
{
    // Calculation of the intersection between the ray and the object
}

double RayTracer::ObjectName::getRadius() const noexcept
{
    return _radius;
}

void RayTracer::ObjectName::setRadius(double radius) noexcept
{
    _radius = radius;
}
```

For the primitives don't forget to normalize your vector

Sixth step

Then you will have to create your object you have to do a last things. If you have to had a variable for a primitives you will have to add it in ObjectBuilder.cpp.

```
void RayTracer::ObjectBuilder::set(const std::string &name, const double &value)
{
    if (name == "radius")
        _object->setRadius(value);
    else if (name == "height")
        _object->setHeight(value);
    else
        throw RayTracer::BuilderError("ObjectBuilder: " + name + " is not a valid
            setter or doesn't take a double");
}
```

For a primitives if you want to have an other parameter that you want to take in the configuration file you will have to add it in this method if the variables is an double. In this ObjectBuilder.cpp file you can add double, vector…

Don't forget to add your object in the CmakeList.txt of the project

PROJECT CARRIED OUT BY:

- Lucas HAUSZLER
- Axel IDOUX
- Thibault GUYONY
- Bastien GERARD