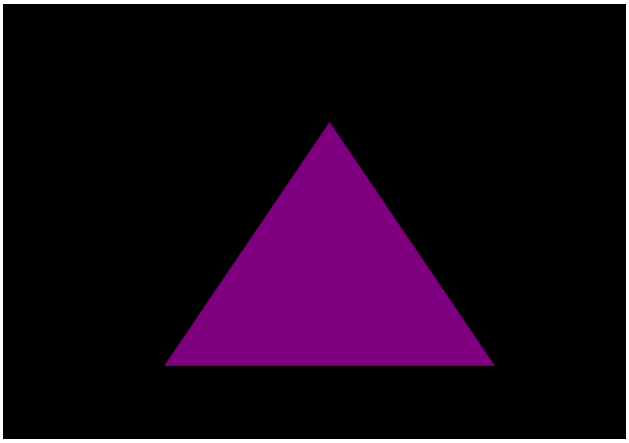
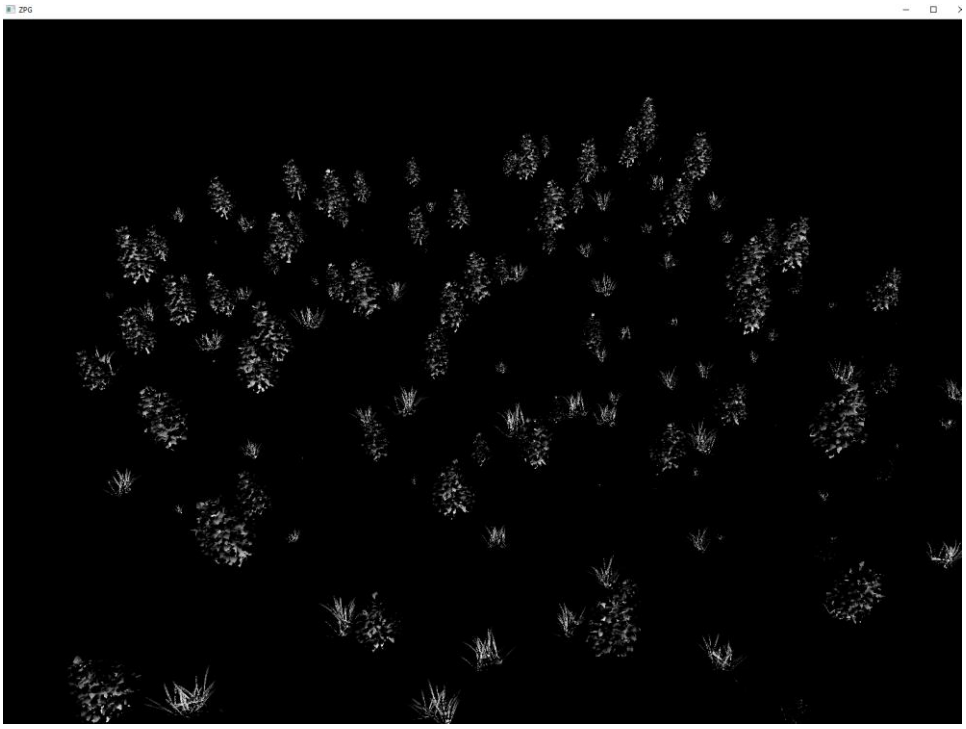
1) Přepínání scén za běhu (Ano/Ne) (kde a jak)

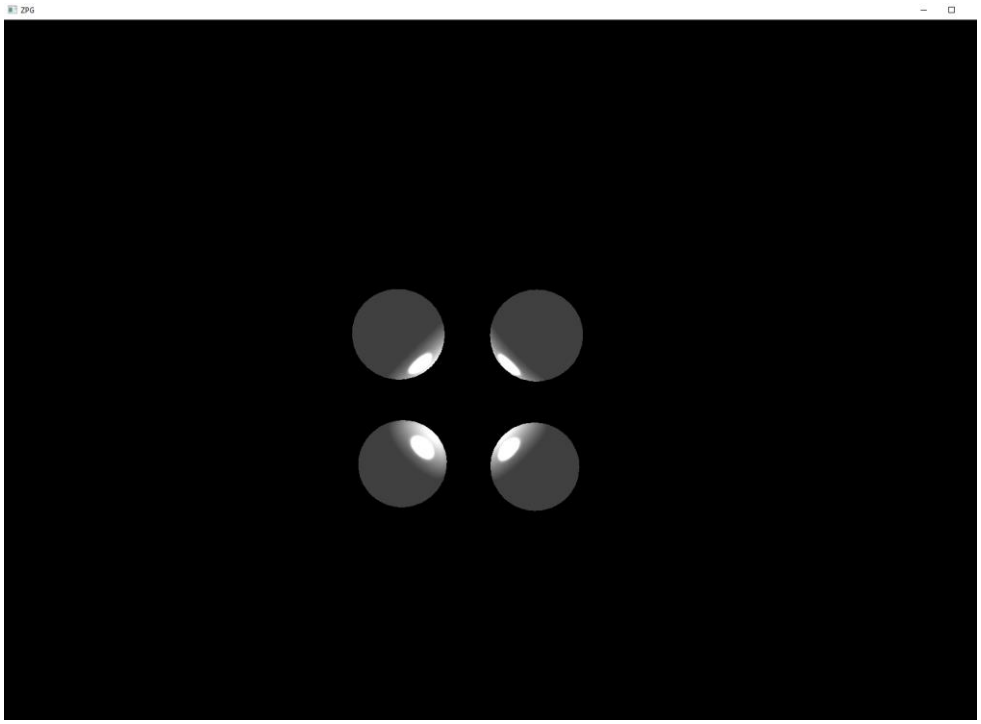
   a) Základní scéna – Ano



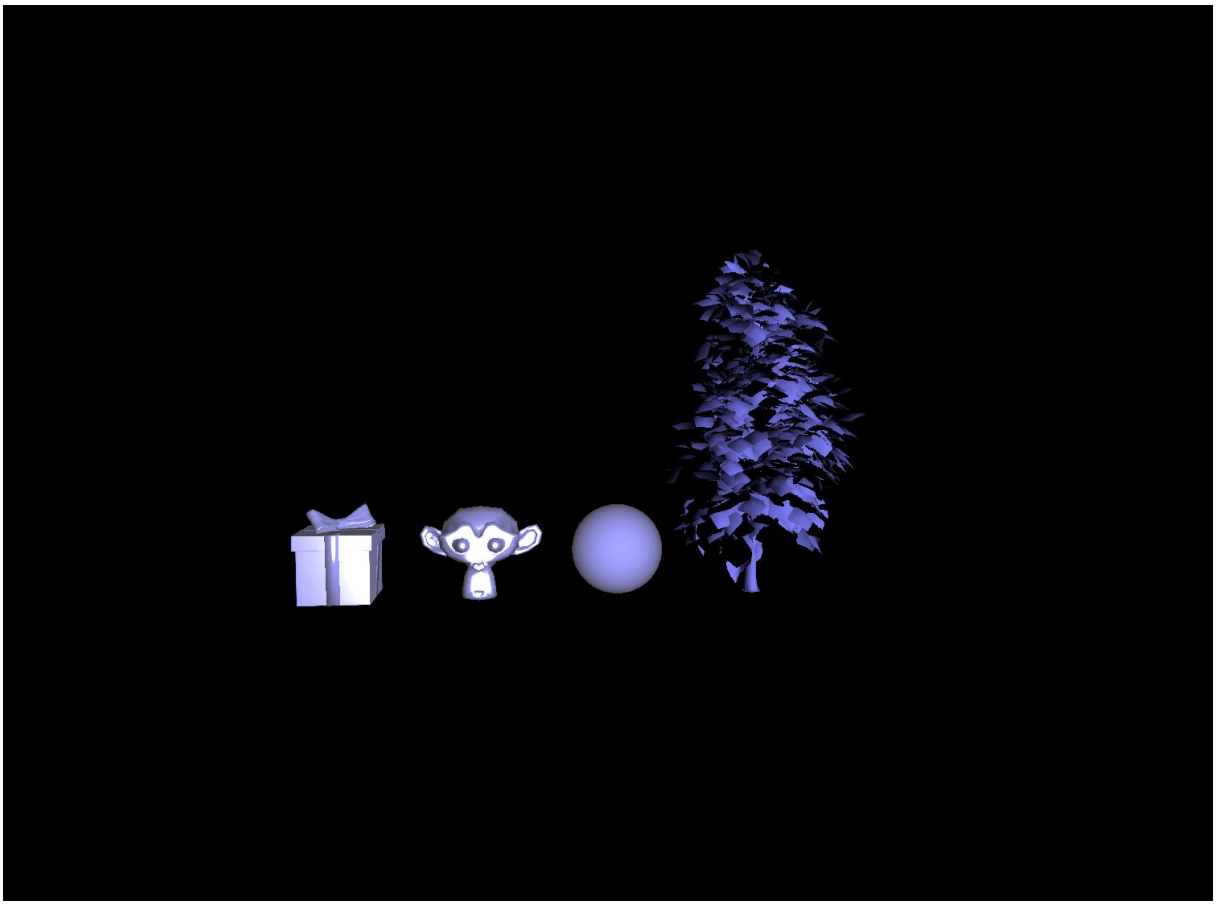
   b) Scéna s lesem – Ano



c) Scéna se čtyřmi kuličkami – Ano



d) Scéna pro demonstraci použití všech shaderů (Konstantní, Lambert, Phong a Blinn) – Ano (Gift – Phong, Suzi – Blinn, Sphere – Lambert, Tree - konstantní)



2) Světlo (Ano/Ne) (kde a jak máte naimplementováno, jak se updatuje změna světla) – Ano, pomocí classy Light, která dědí z Subject virtual classy.

```cpp
#pragma once
#include "Subject.h"
#include <glm/vec3.hpp>
#include <glm/vec4.hpp>
#include <vector>

class Light : public Subject
{
public:
    Light(const glm::vec3& position, const glm::vec4& color);

    void attach(Observer* observer) override;
    void detach(Observer* observer) override;
    void notify() override;
    glm::vec3 getPosition();
    glm::vec4 getColor();


private:
    glm::vec3 position;
    glm::vec4 color;
    std::vector<Observer*> observers;
};
```

Do scény je vkládáno světlo a kamera, kde se po initu světla a kamery, načtou všechny drawable objekty ve kterých je shaderprogram a shaderu se ve scéně pomoci notify řekne kde je světlo umístěno.

3) Základní třídy (ShaderProgram, DrowableObject, Camera, Controller) (Ano/Ne) (kdo zodpovídá za vykreslování, kde jsou uloženy modely, shadery atd.) – Ano

Kromě Controlleru, ShaderProgram zodpovídá za funkčnost shaderu a načtení shaderu pomocí shader loaderu, zasílá hodnoty z jiných class jako je např. kamera do fragment nebo vertex shaderu. ShaderProgram je Observer.

DrawableObject – classa, která se stará o spojení objektu a shaderu, do classy se posílají transformace které jsou následně předány ShaderProgramu, o vykreslení

se stará classa Model pomoci funkce draw

```
Model::Model(const float* model, size_t vertexCount, GLuint mode, int first, int count): mode(mode), first(first), count(count) {
    //vertex buffer object (VBO)
    VBO = 0;
    glGenBuffers(1, &VBO); // generate the VBO
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, vertexCount * sizeof(float), model, GL_STATIC_DRAW);

    //Vertex Array Object (VAO)
    VAO = 0;
    glGenVertexArrays(1, &VAO); //generate the VAO
    glBindVertexArray(VAO); //bind the VAO
    glEnableVertexAttribArray(0); //enable vertex attributes
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    //glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (GLvoid*)0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (GLvoid*)(3 * sizeof(float)));
}

void Model::draw()
{
    glBindVertexArray(VAO);
    glDrawArrays(mode, first, count);
}
```

Camera – classa zodpovídá za funkčnost kamery, je subject pro observer ProgramShader, kamera obsahuje funkčnost jako je pohyb a pohled do stran.

Shadery jsou uloženy v souborech .glsl v resource files. Modely používám z .h souboru ve složce models

```
#include "../Models/tree.h"
#include "../Models/bushes.h"
#include "../Models/gift.h"
#include "../Models/sphere.h"
#include "../Models/suzi_smooth.h"
```

4) Transformace (Composite pattern) (Ano/Ne) (Máte pro transformace základní třídy? Co a jak jste použili?) – Ano, virtuální classa TansformationComponent, ze které následně dědí třídy: Translate, Scale, Rotate, Transformation. Transformation je komposit a třídy Translate, Scale, Rotate reprezentují jednotlivé druhy transformací.
5) Základy OOP
    a) Encapsulation (zapouzdření) (Ano/Ne) (Kde a jak?) – Ano

    Např. v ShaderProgram

```
class ShaderProgram : public Observer
{
public:
    ShaderProgram(const char* vertexFilePath, const char* fragmentFilePath);
    void setTransformation(Transformation &transformation);
    void update(Subject* subject) override;
    void use();

private:
    GLuint shaderProgram;
    glm::mat4 M = glm::mat4(1.0f);
    GLint idModelTransform;
    GLint idModelView;
    GLint idModelProjection;
    GLint idLightColor;
    GLint idLightPosition;
};
```

b) Inheritance (dědičnost) (Ano/Ne) (Kde a jak?) – Ano

Zase např. v ShaderProgramu, dědí z Observeru

c) Polymorphism (polymorfismus neboli mnohotvárnost) (Ano/Ne) (Kde a jak?) – Ano, v transformacích, z TransformationComponent se následně dědí Translate, Rotate a Scale. Každá z těchto poděděných tříd má stejné metody. Ale transformují objekt jiným způsobem.

6) Vertex a fragment shadery prosím uložte do textových souboru a použijte přiložený ShaderLoader pro jejich načítání (Ano/Ne) (Kde a jak jste použili?) – Ano, uloženy v .glsl souborech načítané pomoci ShaderLoaderu které je spouštěn v ShaderProgramu