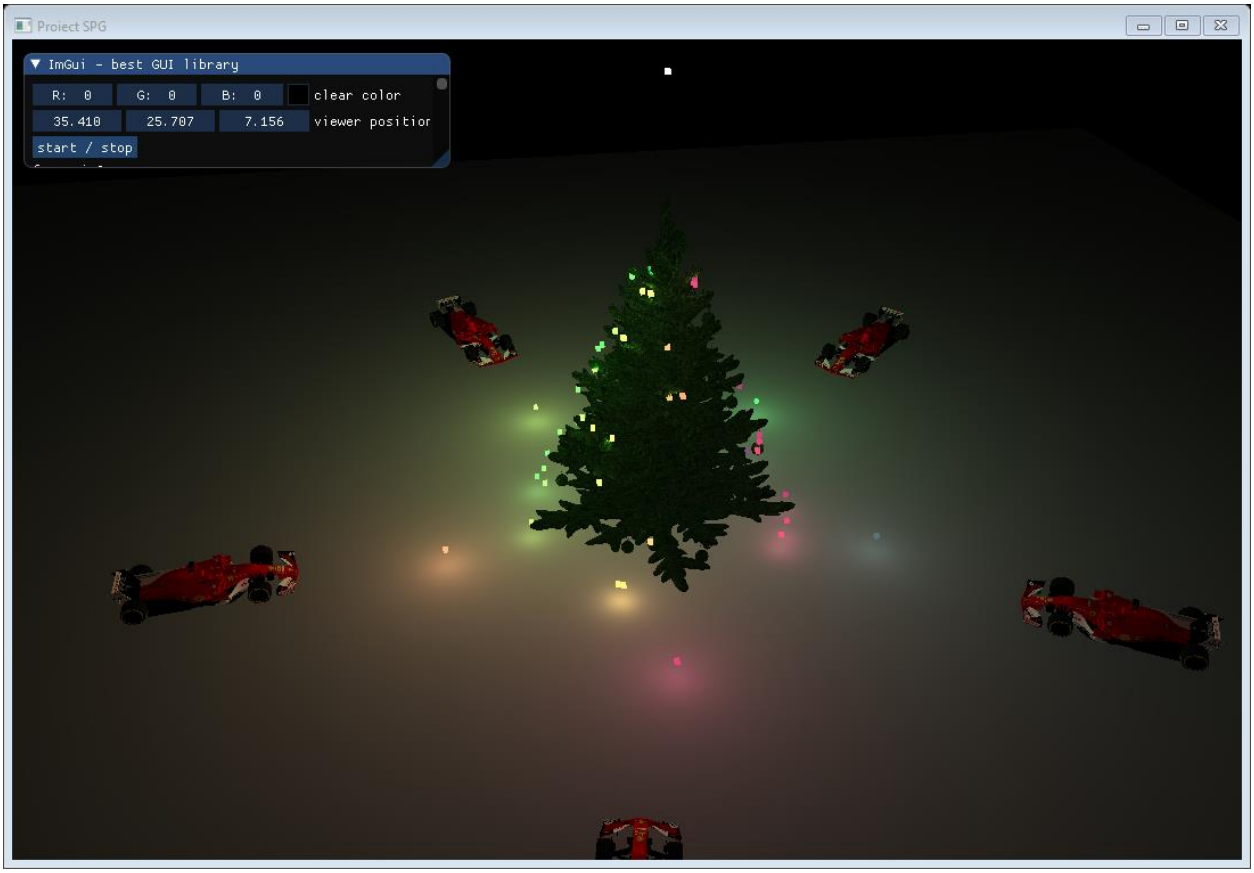
Luminițe



Abstract:

Acest proiect reprezintă o aplicație interactivă 3D creată pentru materia Sisteme de Prelucrare Grafică. Scopul acesteia este înțelegerea detaliată a materiei prin interactiunea usoară și directă cu conceptele predate la curs și laborator, cu toate că această aplicație prezintă și unele concepte care nu au fost incluse în materia predată.

Dependențele proiectului:

- OpenGL 4 (Aveți nevoie de o placă video și un driver corespunzător)
- Visual Studio (Solutia creată este pentru VS 2019 si folosește C++17, însă cu niște modificări poate fi folosită și în cadrul unei versiuni mai vechi)
- GLFW (Bibliotecă folosită pentru crearea ferestrei și lucrul cu mouse-ul si tastatura)
- spdlog (Bibliotecă folosită pentru afișarea mesajelor de log)
- Assimp (Bibliotecă folosită pentru importarea modelelor 3D)
- stb_image (Bibliotecă folosită pentru citirea imaginilor / texturilor)
- ImGui (Bibliotecă folosită pentru elementele de interfață cu utilizatorul)
- glm (Bibloteca folosită pentru calcul matriceal)
- glew (Bibliotecă utilizată pentru a încărca funcțiile moderne din driver-ul de OpenGL)

Tot proiectul este present pe pagina de github https://github.com/ppetrica/ProiectSPG. Acolo se poate urmări tot parcursul proiectului de la primele linii de cod. Pentru a compila proiectul trebuie să clonati repositorul:

```
git clone https://github.com/ppetrica/ProiectSPG.git --recursive
```

În așa fel veți clona și dependențele (spdlog și imgui) incluse ca submodule ale proiectului. Va trebui să instalați și biblioteca Assimp. În cadrul acestui proiect s-a folosit tool-ul **vcpkg** pentru instalarea acestei biblioteci, pentru mai multe instructiuni, vizitați pagina de GitHub a bibliotecii Assimp.

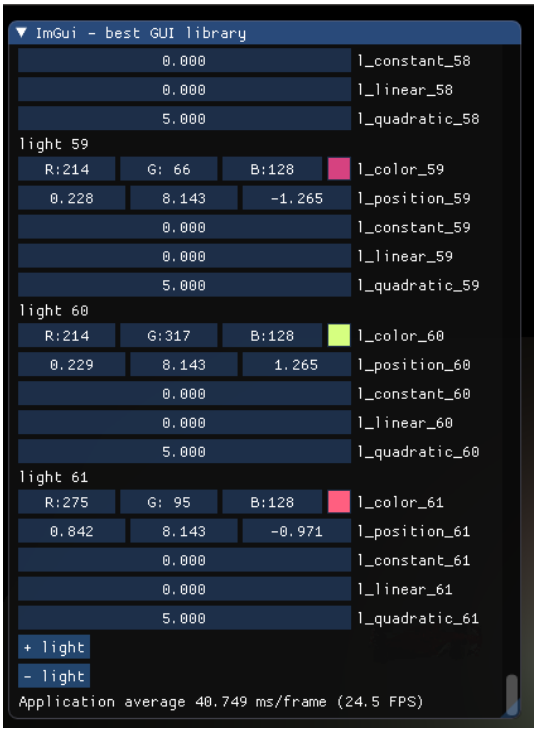
Proiectul suportă momentan doar compilare pentru platforma **x86**. După ce toate dependențele au fost instalate, se poate deschide proiectul în Visual Studio și compilat în mod obișnuit. Pentru o performanță decentă, compilați proiectul în modul **Release**.

Controale

Pentru a vă mișca în scenă utilizați tastele *WASD* în mod obișnuit, pentru a inainta în sus în lungul axei Y se va folosi tasta *Space* , iar pentru a coborî în lungul axei Y se va folosi tasta *X*. Se poate modifica orientarea camerei prin mișcarea mouse-ului.

La pornirea aplicației va trebui sa modificați directia de privire puțin în jos pentru a vizualiza scena.

De asemenea, în partea dreaptă a ferestrei veți observa o serie de controale grafice care pot fi utilizate pentru modificarea scenei:



- Culorarea de fundal a aplicației
- Poziția utilizatorului
- Oprirea / pornirea animației
- Poziția, culoarea și intensitatea fiecărei surse de lumină
- Poziția, scalarea si rotația fiecarei mașini din scenă
- Adăugarea a noi surse de lumină
- Ștergerea surselor de lumină existente

La apăsarea tastei *Escape*, mouse-ul nu va mai fi folosit pentru a controla directia de vizualicare a utilizatorului, ci va putea fi folosit pentru a apăsa pe controalele din interfață / pentru a închide aplicația. La apăsarea repetată se va trece din nou la folosirea mouse-ului pentru setarea direcției de vizualizare.

Scena

În scenă sunt amplasate mai multe obiecte 3D, dintre care unele sunt staționare, iar altele sunt animate.

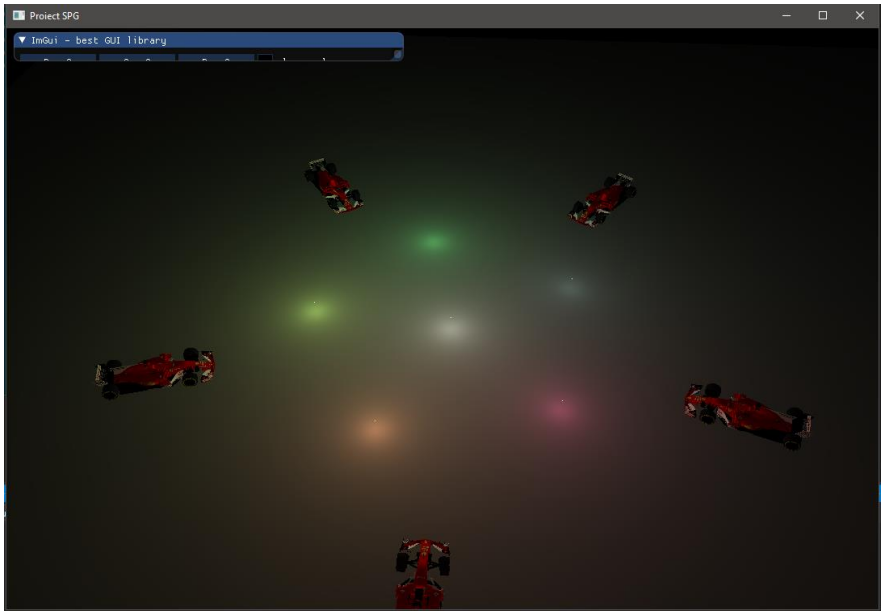
În centrul scenei se observă un brad, care este împodobit programatic cu surse de lumină punctiformă. Culorile acestora sunt calculate folosind funcțiile armonice, astfel toate luminițele de pe o parte a bradului vor avea o culoare asemănătoare. Modelul bradului este urmatorul: https://free3d.com/3d-model/christmas-tree-v2--519851.html. Afișarea acestuia este putin eronată, însă sunt deschis la contribuții / pull request-uri.

În jurul bradului sunt 5 surse de lumină care se rotesc în jurul acestuia în sens contrar acelor de ceasornic, culorile carora la fel au fost calculate folosind functiile armonice.

În spatele acestor surse de lumină se află un rand compus din 5 copii ale urmatorului model: https://free3d.com/3d-model/ferrari-formula-1-72527.html, care se rotesc în sens opus. Nu au fost utilizate texturile avansate pentru Normal Map / Specular Map si Height Map deoarece la încărcarea în Blender s-a constatat că nu aduceau o contribuție suficientă pentru acoperirea costurilor asociate implementării acestora.

În scenă se mai poate vedea o sursă de lumină albă care oscileaza in lungul axei Y, ce trece prin interiorul bradului. Aceasta se poate observa mai bine daca proiectul este compilat de be ramura „model” din git.

Dacă aveți probleme cu frame rate-ul in cadrul aplicației, puteți sa compilați versiunea de pe ramura “model” astfel încât aceasta nu conține bradul și consumă mult mai puține resurse.



Modul de funcționare

Triunghi

Proiectul utilizează biblioteca **GLFW** pentru a crea o **fereastră** și un **context** OpenGL care poate fi utilizat de aplicație pentru a desena în aceasta. GLFW ne asigură o interfață simplificată și cross platform pentru funcțiile native ale sistemului de operare Windows in cadrul platformei **Win32.** Odată ce fereastra este creată și contextul respectiv este atașat aplicației, putem folosi biblioteca glew pentru a încărca toate funcțtiile OpenGL pe care le avem disponibile pentru **placa video** în **driver**-ul asociat ei. În continuare putem folosi funcțiile **OpenGL 4** pentru a desena pe ecran.

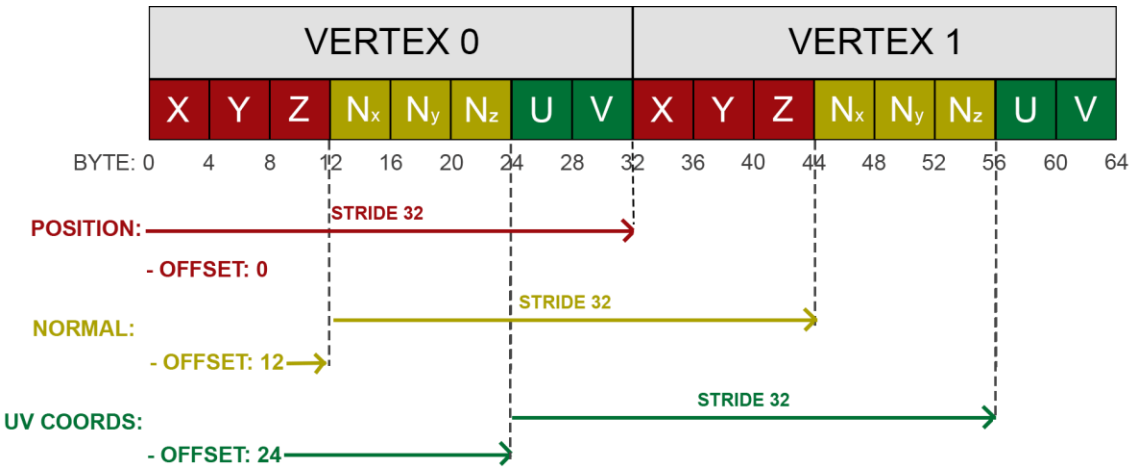
De asemenea, avem nevoie de 2 **shadere**, și anume **vertex** shader-ul și **fragment** shader-ul (pixel shader-ul din cadrul DirectX). Acestea sunt folosite pentru a executa cod pe placa video într-un mediu optimizat pentru calcule paralele și în special pentru **calcul matricial**.

Pentru a vedea o demonstrație a acestor pași și un triunghi desenat pe ecran, puteti folosi git pentru a va reîntoarce la primul commit al proiectului cu id-ul 01de195.

```
git checkout 01de195
```

Încărcarea modelelor

În cadrul acestui proiect a fost folosită biblioteca Assimp pentru încărcarea modelelor 3D. Structura unui vertex din cadrul aplicației este urmatoarea:



Aceasta corespunde cu urmatoare secventa de cod:

```
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, 0, sizeof(loader::vertex), NULL);

glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, 0, sizeof(loader::vertex),
                      (const void *)(sizeof(glm::vec3)));

glEnableVertexAttribArray(2);
glVertexAttribPointer(2, 2, GL_FLOAT, 0, sizeof(loader::vertex),

                      (const void *)(2 * sizeof(glm::vec3)));
```

Iluminarea

Iluminarea a fost realizata dupa modelul de iluminare Phong. În cadrul proiectului au fost utilizate doar surse de lumină punctiforme. Fiecare iecare sursă de lumină are asociată o **structură** în programul **C++**, respectiv o structură în cadrul **fragment shader**-ului:

```
struct point_light {
	vec3 position;
	vec3 color;

	vec3 ambient;

	float constant;
	float linear;
	float quadratic;
};
```

Urmatoarea functie este responsabilă de setarea si actualizarea continuă a valorilor utilizate in fragment shader:

```
    void update() {
        glUseProgram(program);

        glUniform3fv(position_loc, 1, glm::value_ptr(position));

        glUniform3fv(ambient_loc, 1, glm::value_ptr(ambient));

        glUniform3fv(color_loc, 1, glm::value_ptr(color));

        glUniform1f(constant_loc, constant);
        glUniform1f(linear_loc, linear);
        glUniform1f(quadratic_loc, quadratic);
    }
```

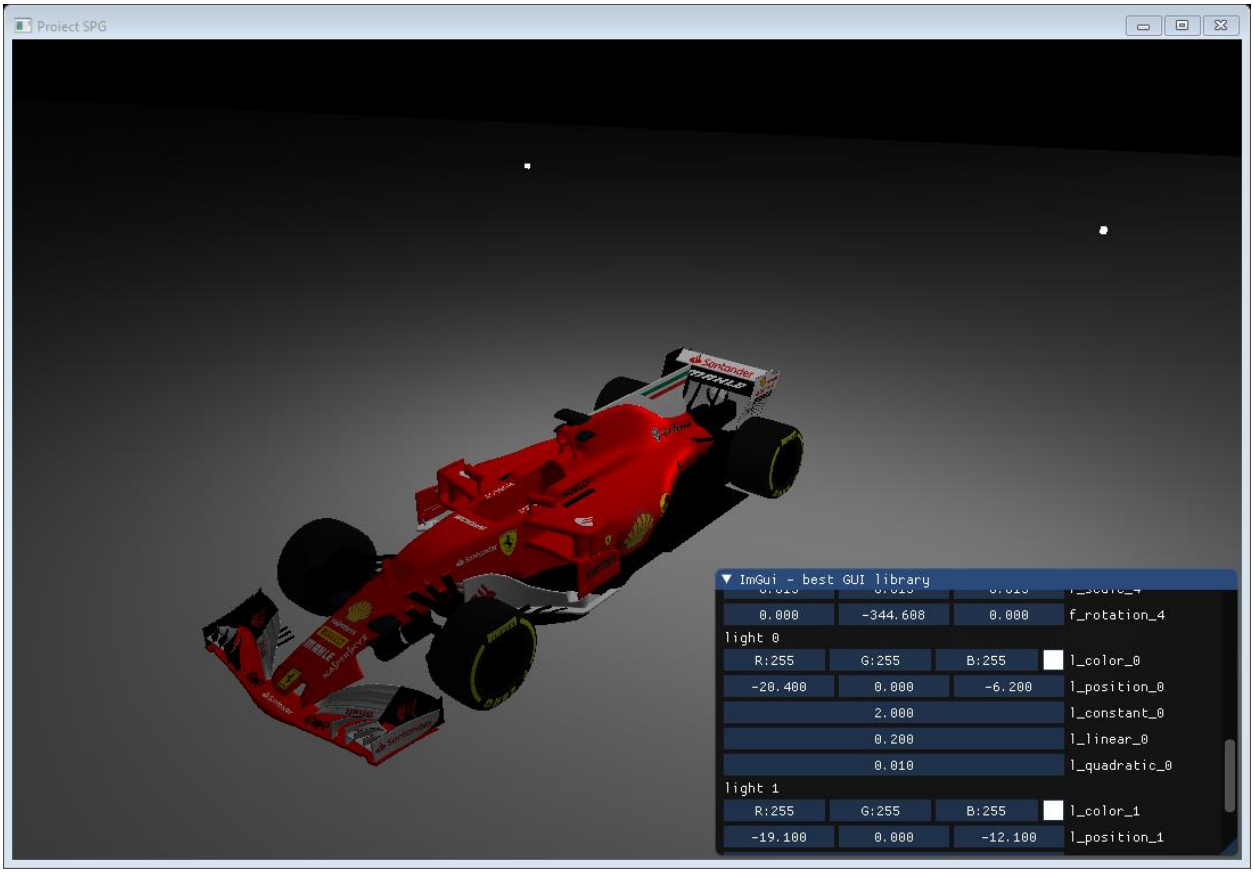
Locatiile variabilelor respective, sunt setate in constructor in functie de denumirea variabilei de tip **point_light** din shader:

```
light (uint32_t program, std::string name, glm::vec3 position = glm::vec3(0.0f),
glm::vec3 color = glm::vec3(1.0f))
        : program(program), name(name), position(position), ambient(0.3f), color(color),
constant(2.0f), linear(0.2f), quadratic(0.01f) {
        position_loc = get_location(program, (name + ".position").c_str());

        ambient_loc = get_location(program, (name + ".ambient").c_str());

        constant_loc = get_location(program, (name + ".constant").c_str());
        linear_loc = get_location(program, (name + ".linear").c_str());
        quadratic_loc = get_location(program, (name + ".quadratic").c_str());

        color_loc = get_location(program, (name + ".color").c_str());

        light_color_location = get_location(program, "u_light_color");
    }

// Instantierea unui obiect de tip light
light l{ program, "u_light[0]", glm::vec3{0.0f, 0.2f, 0.0f} };

// In interiorul fragment shader va corespunde cu primul element din acest vector
uniform point_light u_light[256];
```

În continuare aceste lumini sunt folosite pentru a calcula un factor de iluminare al unui obiect, iar apoi sunt combinate cu culoarea citita din textura asociata obiectului:

```
vec3 light = vec3(0.0f);
for (int i = 0; i < u_n_lights; ++i) {
	light += calculate_point_light(u_light[i], u_viewpos, pos, normal) * u_light[i].color;
}
vec3 color = min(light * texture(u_tex, uv_coords).rgb, 1.0f);
```

Astfel intensitățile luminilor aflate la o distanță mică de obiect sunt combinate pentru a obține următorul rezultat:



Concluzie:

Acest proiect reprezintă un tool ce poate fi ușor extins pentru a putea studia cu ușurință variația parametrilor de luminozitate a luminilor. Folosind ca bază acest proiect, se pot crea animații și efecte luminoase interesante. Cineva care ar replica acest proiect ar aprofunda cunoștințe în grafiă 3D, iluminare, OpenGL, algebra lineară și C++.

Autor: Petrica Petru 1306B **Contact:** petrica.petru99@gmail.com