МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Web-додаток для редагування тривимірних моделей»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ-81-9 Харченко Кирило Олександрович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____ «__» _____2022 р.

Науковий керівник _____ к.т.н., доц., Нагорний В. В.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2022

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. кафедри ІТ

_____ В. В. Шендрик
«__» _____ 2022 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Харченко Кирило Олександрович

1 Тема роботи *Web-додаток для редагування тривимірних моделей*

керівник роботи *Нагорний Володимир В'ячеславович, к.т.н., доцент*,

затверджені наказом по університету від « 24 » квітня 2022 р. № 0301-VI

2 Строк подання студентом роботи « 14 » червня 2022 р.

3 Вхідні дані до роботи технічне завдання на розробку web-додатку для редагування тривимірних моделей

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, моделювання та проєктування, розробка додатку

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) ___актуальність, мета та задачі, аналіз аналогів, засоби розробки, функціональне моделювання, діаграма варіантів використання, головна сторінка додатку, висновки

6. Консультанти розділів роботи:

Розділ	**Консультант**	**Підпис, дата**	
		Завдання видав	**Завдання прийняв**

7.Дата видачі завдання_____05.10.2021_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Оформлення планування робіт	02.05.2022-07.05.2022	
2	Оформлення технічного завдання	07.05.2022-14.05.2022	
3	Проведення аналізу технічної області	14.05.2022-18.05.2022	
4	Проведення проєктування	18.05.2022-25.06.2022	
5	Розробка застосунку	25.05.2022-07.06.2022	
6	Тестування застосунку	07.06.2022-08.06.2022	
7	Оформлення пояснювальної записки	08.06.2022-14.06.2022	

Студент _____ Харченко К.О.
(підпис)

Керівник роботи _____ к.т.н., доц. Нагорний В.В.
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Web-додаток для редагування тривимірних моделей».

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 15 найменувань, додатків. Загальний обсяг роботи − 120 сторінок, у тому числі 36 сторінок основного тексту, 2 сторінки списку використаних джерел, 82 сторінок додатків.

Кваліфікаційну роботу бакалавра присвячено розробці Web-додатку для редагування тривимірних моделей. В роботі проведено огляд додатків аналогів та виявлення особливостей, аналіз наявних технологій, сформовано мету та основні вимоги до реалізації проєкту. У роботі виконано планування робіт та функціональне моделювання. Результатом проведеної роботи є створення web-додаток для редагування 3d моделей. Практичне значення роботи полягає у розробці web-додатку для редагування тривимірних моделей, що дозволить завантажувати моделі в форматах .STL, .PLY, переглядати модель та виконувати спрощення моделі, розділяти трикутники на більшу кількість.

Ключові слова: 3D модель, WebGL, WebAssembly, Rust, редактор.

ЗМІСТ

ВСТУП

На сьогоднішні багато рішень, що були доступні як лише рішення на локальному комп’ютері переходять на хмарні технології або доступні в браузері. Додатки, що працюють з 3d графікою не виняток. Останнім часом стали розвиватися технології, що дозволяють використовувати API відеокарти з браузера - як наслідок API WebGL[1], також з’явився WebAssembly (WASM)[2], що призначений для більш ефективного виконання коду ніж JavaScript[3] на віртуальній машині (в браузері або на сервері)[4].

Реалізація даного проєкту дозволить створити Web-додаток для редагування тривимірних моделей. Також дозволить зрозуміти доцільність та ефективність використання технологій (WebGL, WASM). Метою дипломного проєкту є розробка Web-додатку для редагування тривимірних моделей.

Для досягнення поставленої мети необхідно виконати наступні задачі:

– визначити актуальність роботи, дослідити предметну область;

– провести аналіз наявних публікацій та аналогічних проєктів для виявлення особливостей;

– обрати та проаналізувати технології для розробки проєкту;

– сформулювати постановку задачі та основні вимоги для реалізації проєкту;

– провести структурно-функціональний аналіз виконання проєкту, розробити контекстну діаграму, діаграму декомпозиції та варіантів використання;

– розробити архітектуру додатка;

– створити інтерфейс для проєкту;

– виконати тестування додатка.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень для Web-додатку

Багато додатків почали переходити на хмарні рішення та реалізують інтерфейс у браузері, що дозволяє легше сприяти поширенню та простоті у використанні. Останнім часом почали розвиватися технології, що дозволяють розширити можливості браузера – WebGL, WASM.

WebAssembly (WASM) – мова програмування низького рівня. Становить собою асемблер для браузерів з компактним бінарним форматом, котрий працює з продуктивністю у порівнянні з нативною, та служить для компіляції для таких мов програмування, як: C/C++, Rust, C#, Go, дозволяючи останнім виконувати код в мережі. WASM призначений для доповнення JavaScript – дозволяє обмінюватися даними.

WebGL (Web Graphics Library) – це API JavaScript для відтворення високопродуктивної інтерактивної 3d та 2d-графіки в будь-якому сумісному веббраузері без використання плагінів. WebGL робить це, представляючи API, який точно відповідає OpenGL ES 2.0, який можна використовувати в елементах HTML5 <canvas>. Ця відповідність дає можливість API використовувати переваги апаратного графічного прискорення, що надається пристроєм користувача.

JavaScript (JS) – це легка, інтерпретована або на льоту скомпільована мова програмування з першокласними функціями. Це мова сценаріїв для вебсторінок, багато середовищ, які не є браузерами, також використовують його, наприклад Node.js, Apache CouchDB та Adobe Acrobat. JavaScript – це заснована на прототипах, багатопарадигмальна, однопотокова динамічна мова, яка підтримує об’єктноорієнтований, імперативний та декларативний стилі.

HTML (HyperText Markup Language) [5] є основним будівельним блоком Інтернету. Він визначає значення та структуру вебконтенту. Інші технології, крім HTML, зазвичай використовуються для опису зовнішнього вигляду/презентації

вебсторінки (CSS) або функціональності/поведінки (JavaScript).

1.2 Аналіз існуючих продуктів-аналогів

Через те, що браузери були створені для відображення html документів, то кількість додатків, що дозволяють працювати з 3d графікою не так багато та наявний функціонал у вебверсій набагато менший ніж в аналогів для десктопу. Для визначення вимог майбутнього програмного продукту було проведено дослідження наявних аналогів додатків для роботи з 3d моделями в браузері, а саме: “SketchUp”[6], “Tinkercad”[7], “Clara.io”[8].

“SketchUp” дозволяє створювати 2d примітиви (коло багатокутники) та за допомогою операції “Extrude” (рис. 1.1) витягувати форму моделі по осі. В редакторі можливо виконувати базові операції з моделями (переміщення, поворот) та налаштовувати шлях анімації для камери.

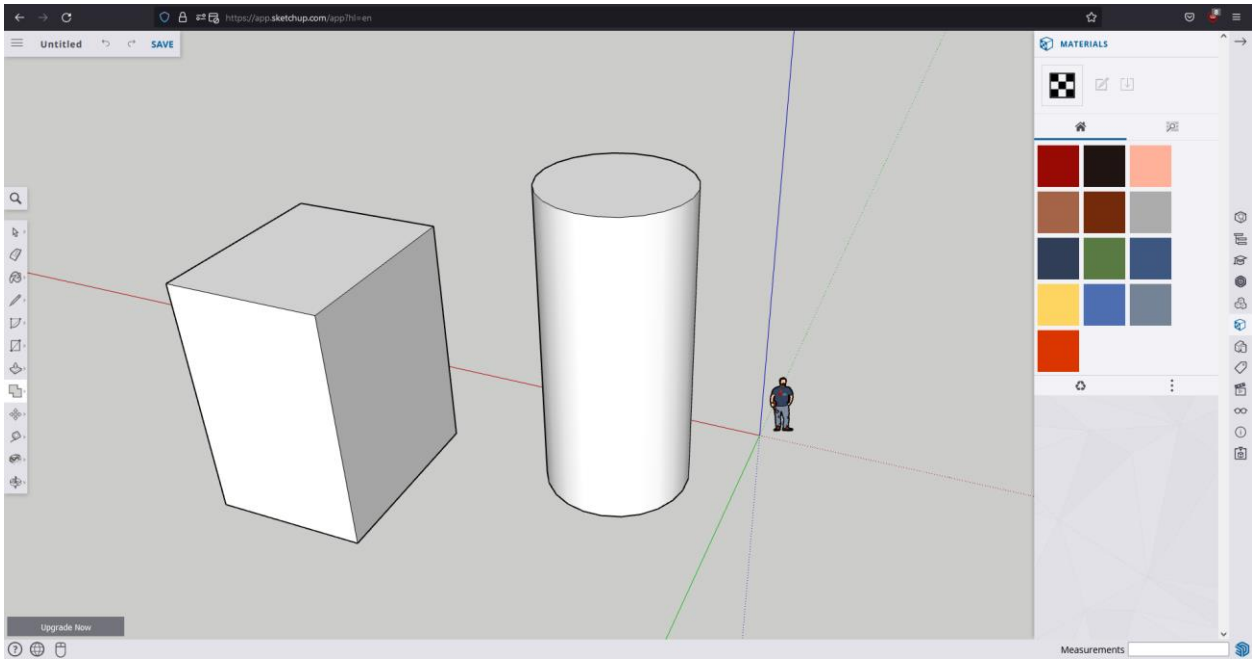


Рисунок 1.1 – Редактор “SketchUp”

Можливо також виконувати булеві операції між моделями (конструктивна блокова геометрія), але цей функціонал реалізовано в платній версії (рис. 1.2), також можливо активувати режим доповненої реальності.

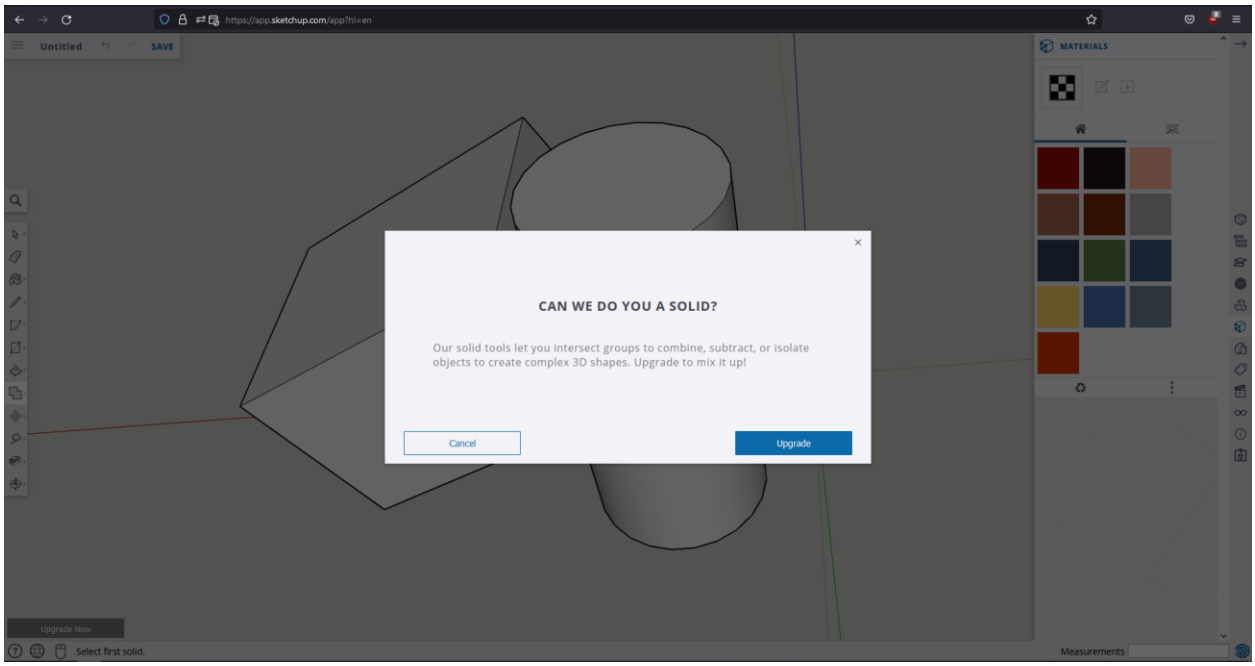


Рисунок 1.2 – Конструктивна блокова геометрія в редакторі “SketchUp”

“Tinkercad” дозволяє виконувати базові операції з моделями та об’єднувати моделі (рис. 1.3). Має симуляцію для електронних схем.

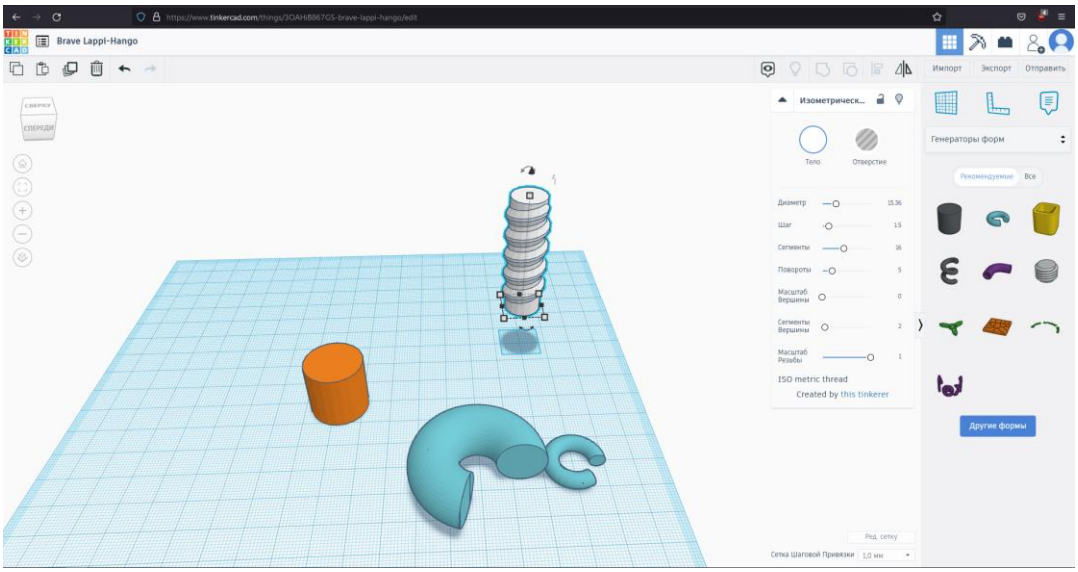


Рисунок 1.3 Редактор “Tinkercad”

Окрім режиму проєктування має режим кодування (рис. 1.4) – за допомогою блоків можна створювати адаптивні, динамічні 3d форми та анімацію для 3d об’єктів.

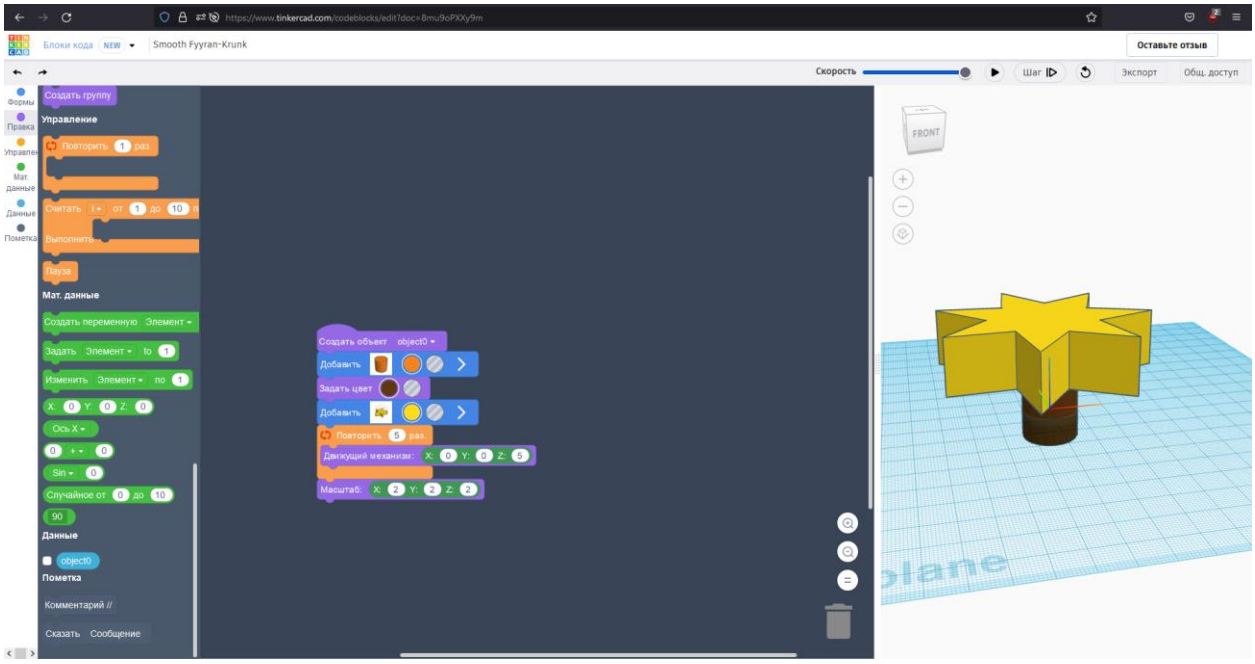


Рисунок 1.4 Режим кодування в редакторі “Tinkercad”

“Clara.io” хмарний програмний продукт для 3d моделювання (рис. 1.5), анімації та візуалізації (рис. 1.6). Має платний обліковий запис для підвищення процесорного часу для обробки моделей та збільшення простору.

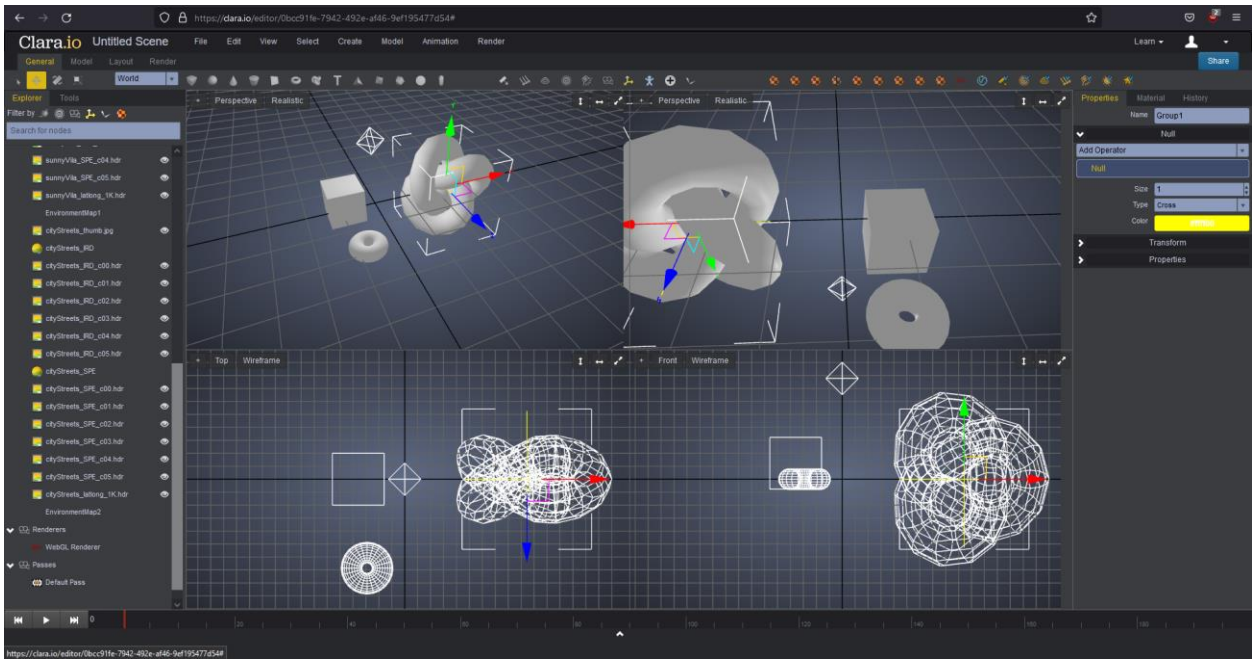


Рисунок 1.5 Редактор “Clara.io”

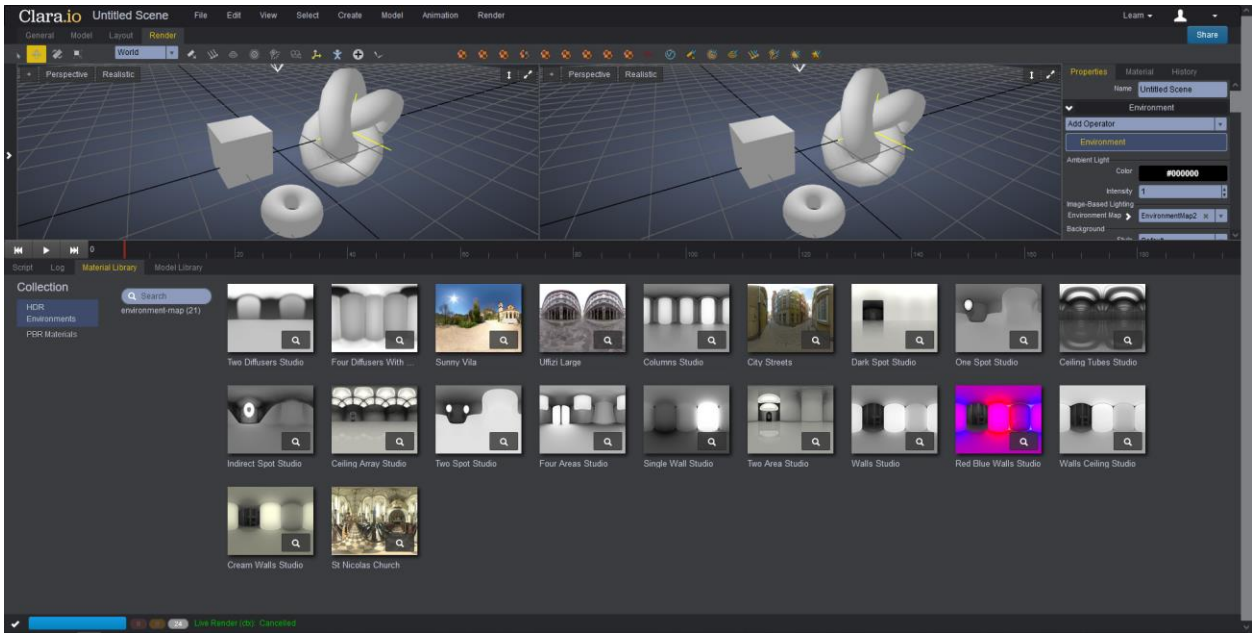


Рисунок 1.6 Вікно візуалізації в редакторі “Clara.io”

Дозволяє змінити топологію 3d моделі: тріангуляція, “smooth”, “bend”, “twist”, “taper” (рис. 1.7). Також можливо перейти на рівень редагування полігонів, граней, вершин.

Рисунок 1.7 Розширені налаштування моделі в редакторі “Clara.io”

Аналоги продуктів для браузера мають набагато менший функціонал у порівнянні з варіантами на десктопі. Це обумовлено тим, що для обробки великих моделей потрібно багато процесорного часу, тому потужні редактори використовують хмарні технології для обчислень.

Представлені аналоги виступають як більш прості версії рішень на десктопі. До найбільш функціонального продукту з безплатним доступом можна назвати “Clara.io”, тому що вона може не тільки виконувати прості операції над 3d моделями, а ще редагувати модель на рівні вершин, створювати анімацію та візуалізувати сцену з матеріалами.

Після детального аналізу аналогів онлайн редакторів 3d моделей було визначено їх переваги та недоліки. Його результати представлені в таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця характеристик онлайн редакторів.

Параметр	“SketchUp”	“Tinkercad”	“Clara.io”	Власне рішення
Базові операції з моделями (переміщення, поворот)	+	+	+	+
Платний доступ до розширення функціоналу	+	-	+	-
Розділення трикутників 3d моделей	-	-	+	+
Спрощення 3d моделей	-	-	-	+
Можливість редагування моделі на рівні вершин/граней/полігонів	-	-	+	-
Можливість виконання булевих операцій над моделями	+ (платна версія)	-	-	-
Можливість зберігання сцен/моделей користувача	+	+	+	+

Дані з таблиці 1.1 надають змогу під час розробки звернути увагу на цікаві функціональні доповнення, які можна використати, і недоліки, які варто подолати. З функціональних доповнень варто виділити такі можливості редагування топології 3d моделей: “smooth” (згладження моделі), “bend” (згинає модель), “twist” (скручення моделі), “taper” (звуження моделі).

1.3 Постановка задачі

Метою дипломного проєкту є розробка Web-додатку для редагування тривимірних моделей. Також потрібно створити простий та інтуїтивно зрозумілий інтерфейс та зрозуміти доцільність та ефективність використання технологій (WebGL, WASM).

Основні вимоги для створюваного програмного продукту є наступними:

– організувати імпорт/експорт моделей з форматів: .STL, .PLY;

– забезпечити операції розділення трикутників та спрощення;

– розробити інтерфейс для авторизації/реєстрації користувачів;

– розробити інтерфейс для головної сторінки з головними налаштуваннями.

Для досягнення мети проєкту необхідно зробити:

– визначити актуальність роботи, дослідити предметну область;

– провести аналіз наявних публікацій та аналогічних проєктів для виявлення особливостей;

– сформулювати постановку задачі та основні вимоги для реалізації проєкту;

– провести структурно-функціональний аналіз виконання проєкту, розробити контекстну діаграму, діаграму декомпозиції та розробити модель варіантів використання;

– обрати та проаналізувати технології для розробки проєкту;

– розробити архітектуру додатка;

– створити інтерфейс для проєкту;

– дослідити та розробити алгоритми розділення трикутників та спрощення;

– виконати тестування додатка.

Для більш детального опису технічне завдання на розробку продукту проєкту наводиться у додатку А.

1.4 Вибір засобів реалізації

Для реалізації будуть використані такі технології:

– мова програмування Rust та компіляція в WASM для запуску окремих модулів в браузері;

– мова програмування JavaScript для виклику WASM модулів;

– API WebGL для взаємодії відеокарти та JavaScript;

WebAssembly (WASM) – мова програмування низького рівня. Являє собою асемблер для браузерів з компактним бінарним форматом, котрий працює з продуктивністю у порівнянні з нативною, та служить для компіляції для таких мов програмування, як: C/C++, Rust, C#, Go, дозволяючи останнім виконувати код в мережі. WASM призначений для доповнення JavaScript – дозволяє обмінюватися даними.

Rust[9] – нова мова програмування загального призначення схожа з C/C++. Особливістю цієї мови є гарантії безпеки для пам’яті та паралелізму – завдяки механізму ‘володіння’ та афінних типів, що дозволяє обходитися без збирання сміття. Rust має продуктивність порівняну з C/C++ та має офіційний компілятор в WASM.

WebGL – API для 3d графіки в браузері, частина коду може бути виконана безпосередньо на відеокартах.

Більшість коду може бути написана на Rust та скомпільована у WASM і потім викликана з JavaScript коду. Але на цей момент підтримка компіляції в WASM для багатьох мов програмування експериментальна і є труднощі з доступом до файлової системи, тому що WASM виступає, як окрема платформа, а браузери обмежені до доступу ресурсів ОС з точки зору безпеки. Тому Rust та компіляція в WASM буде використана для алгоритмів обробки 3d моделі та коду рендеру, щоб підвищити продуктивність. JavaScript буде використано для

взаємодії з файловою системою та, як проміжний шар для взаємодії ОС та модулів WASM. В якості мови для бекенду буде використано мову Rust.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ

2.1 Проєктування інформаційної системи

Моделювання складається з етапів, які пов’язані між собою. Процес моделювання починається з абстрактної концептуальної схеми, після неї створюються логічна та фізична моделі.

Функціональне моделювання Web-додатку для редагування тривимірних моделей за допомогою IDEF0

Для діаграми IDEF0 було визначено:

– вхідні дані: файли в форматі .STL, .PLY;

– вихідні дані: файли в форматі .STL, .PLY, збережені файли в форматі .STL, .PLY для користувача;

– управління: початкові налаштування 3d моделей, дії користувача;

– механізми: онлайн редактор 3d моделей, апаратне та технічне забезпечення.

Функціональне моделювання онлайн редактору 3d моделей в нотації IDEF0 представлене на рисунку 2.1.

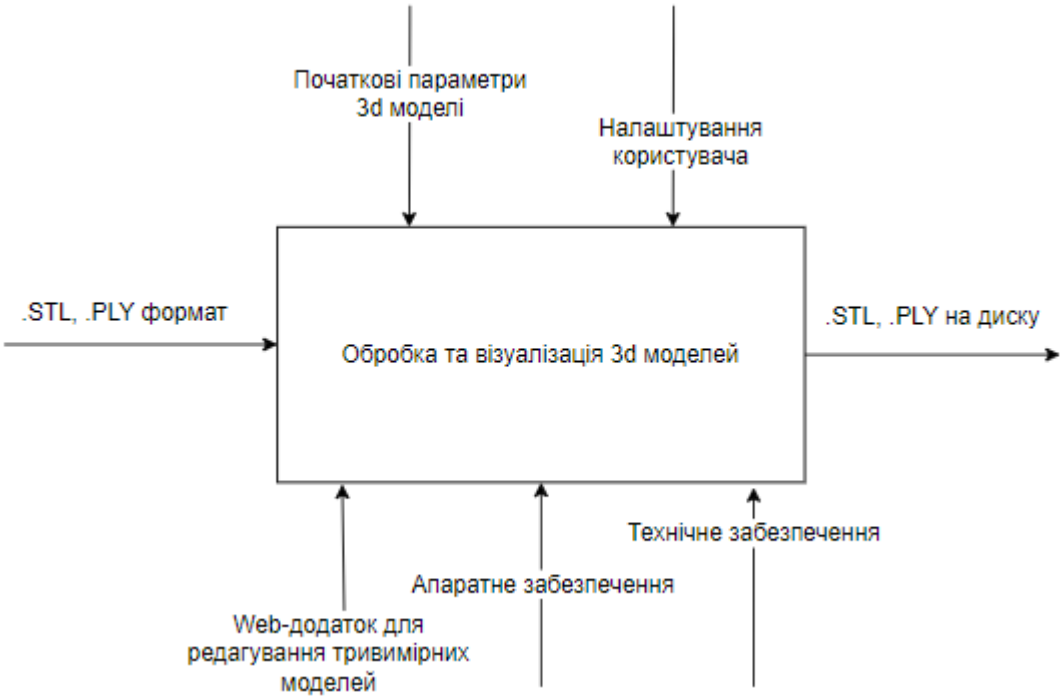


Рисунок 2.1 – IDEF0

Декомпозиція функціональної моделі збереження файлу на диск користувача представлена на рисунку 2.2.

Декомпозиція функціональної моделі збереження файлу на диск користувача включає підпроцеси:

– парсинг моделей з вхідних файлів;
– візуалізація;
– спрощення, розділення трикутників;
– збереження моделі на диск користувача.

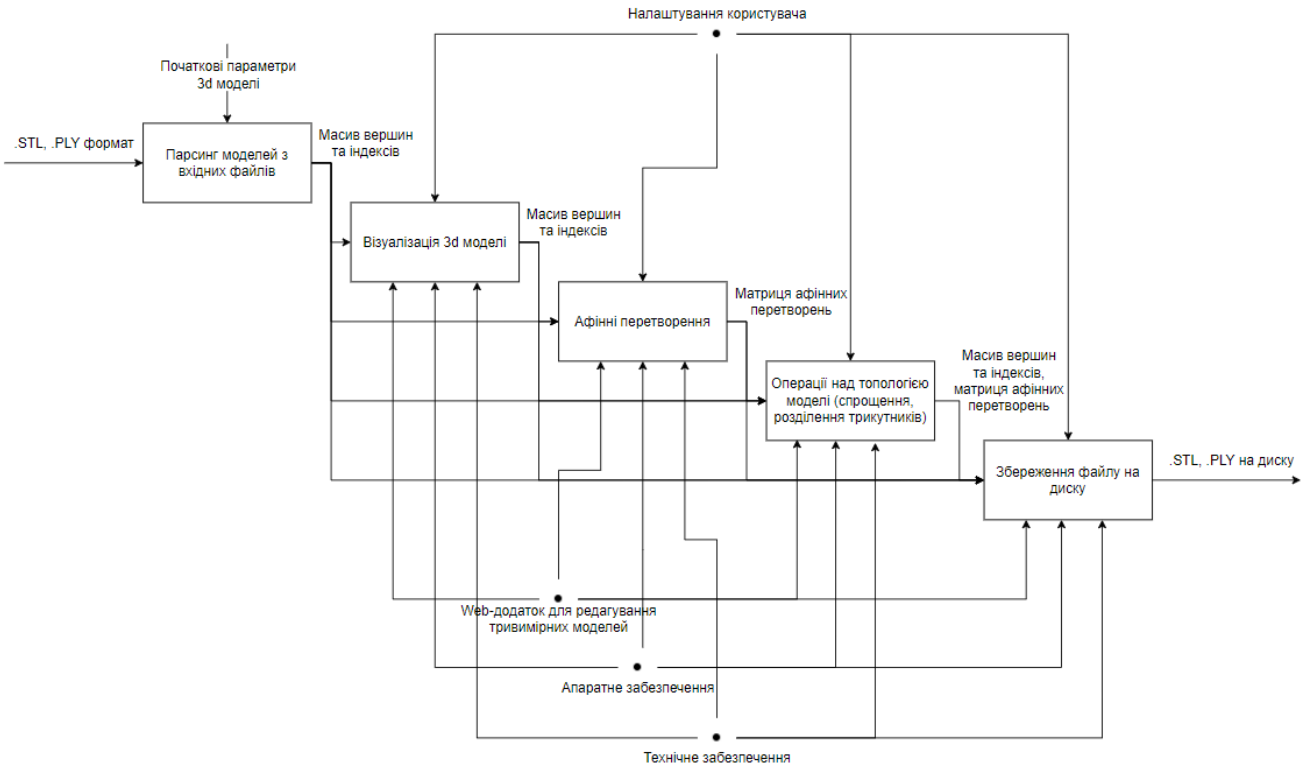


Рисунок 2.2 – Декомпозиція збереження файлу на диск користувача

Діаграма варіантів використання (use-case diagram) – використовується для зображення сценаріїв використання системи та користувачів системи, які використовують її функції.

Для досягнення цілей функціонування спочатку будується модель у формі діаграми варіантів використання, яка описує функціональне призначення системи. Діаграма варіантів використання є вихідною концептуальною моделлю системи в процесі її проєктування та розробки.

Для Web-додатку для редагування тривимірних моделей варіанти використання:

- імпорт моделей;
- експорт моделей;
- перегляд моделі;
- збереження моделей на сервері;
- спрощення моделі;
- розділення трикутників моделі.

Актори на моделі: користувач.

Діаграма варіантів використання в нотації UML представлена на рисунку 2.3.

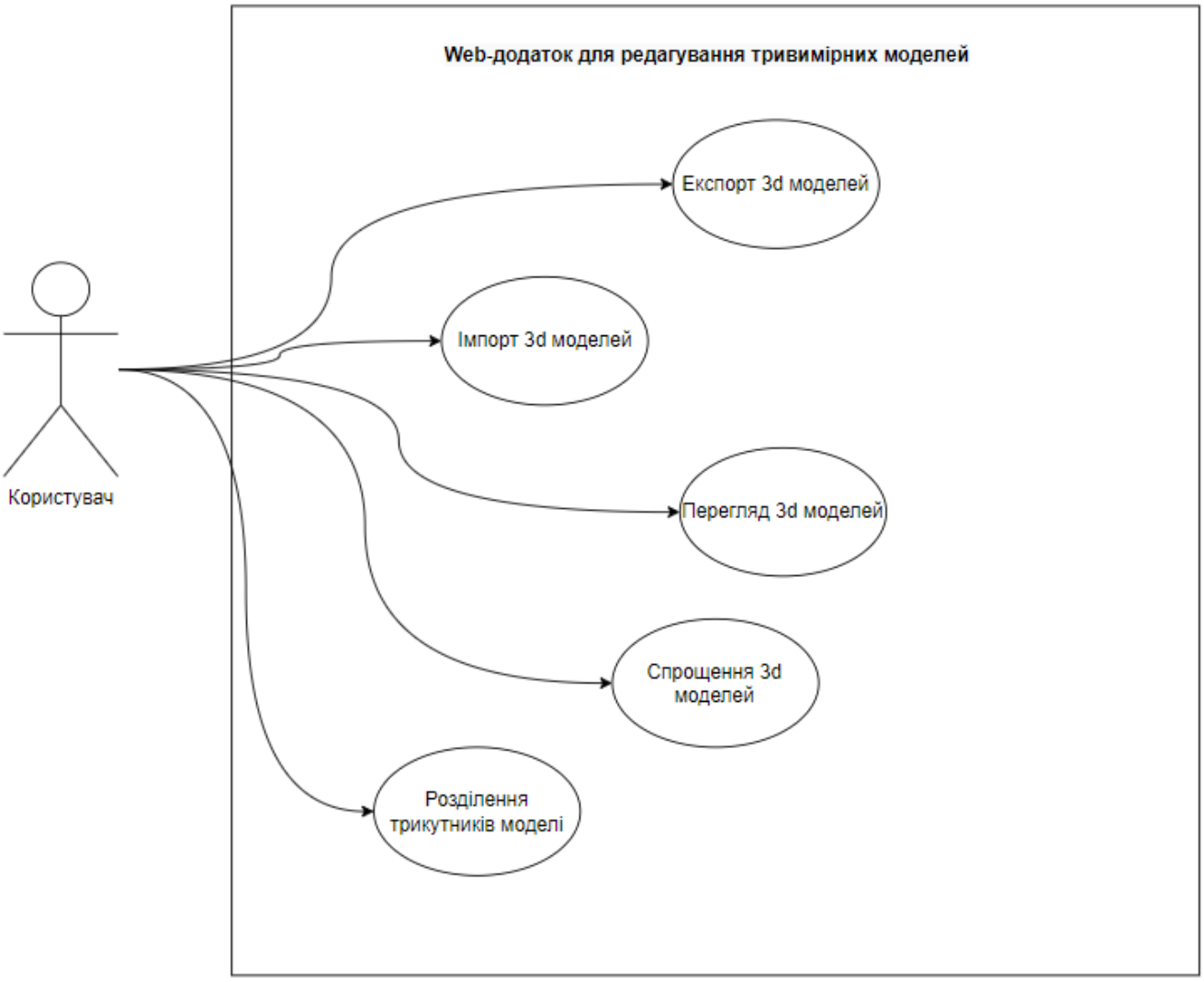


Рисунок 2.3 – Діаграма варіантів використання

3 РОЗРОБКА ДОДАТКУ

3.1 Архітектура програмного додатку

Основна частина застосунку буде написана на мові Rust та скомпільована в модулі WASM після чого ці модулі будуть викликано з сценаріїв JavaScript. Для графічного інтерфейсу буде використана бібліотека “egui”[14]. Код буде повністю виконуватися на стороні клієнта – вхідною точкою буде index.html, що завантажує модулі WASM.

Інтерфейс має бути простим та інтуїтивно зрозумілим (рис. 3.1, 3.2).

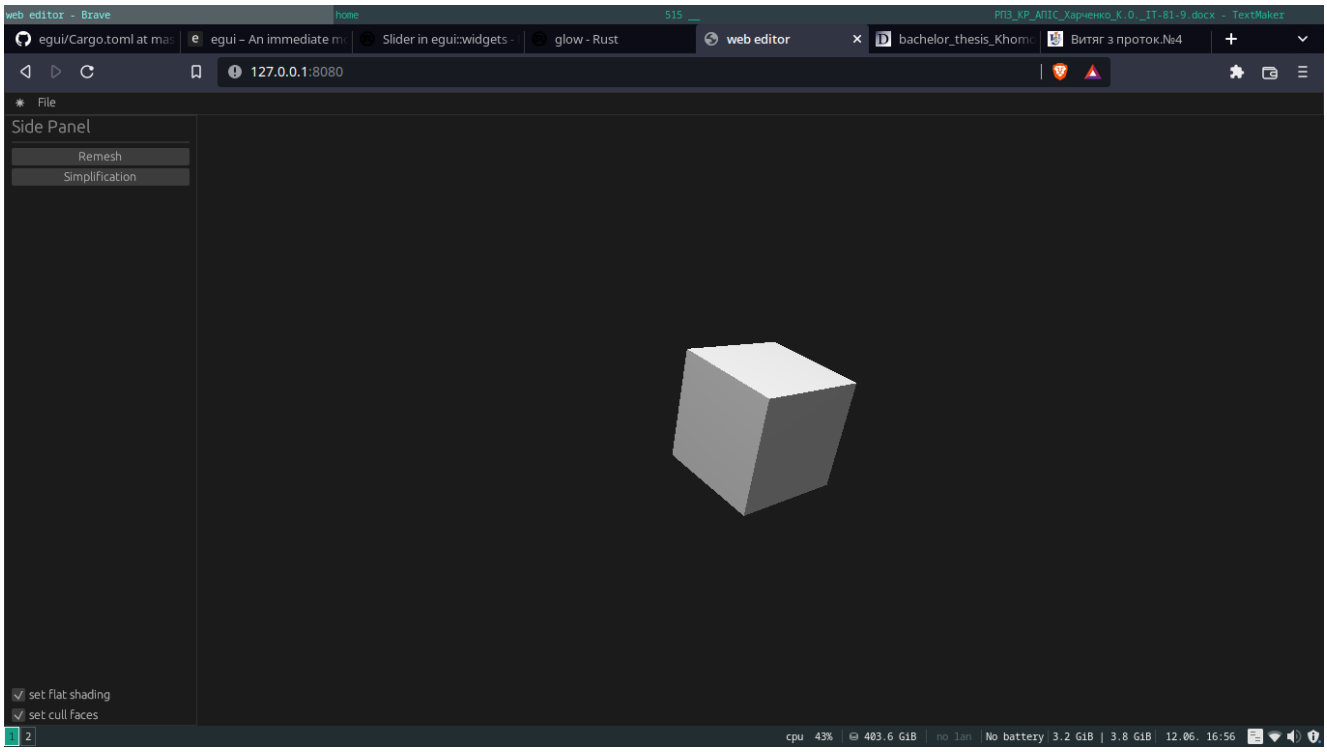


Рисунок 3.1 – Інтерфейс (темна тема)

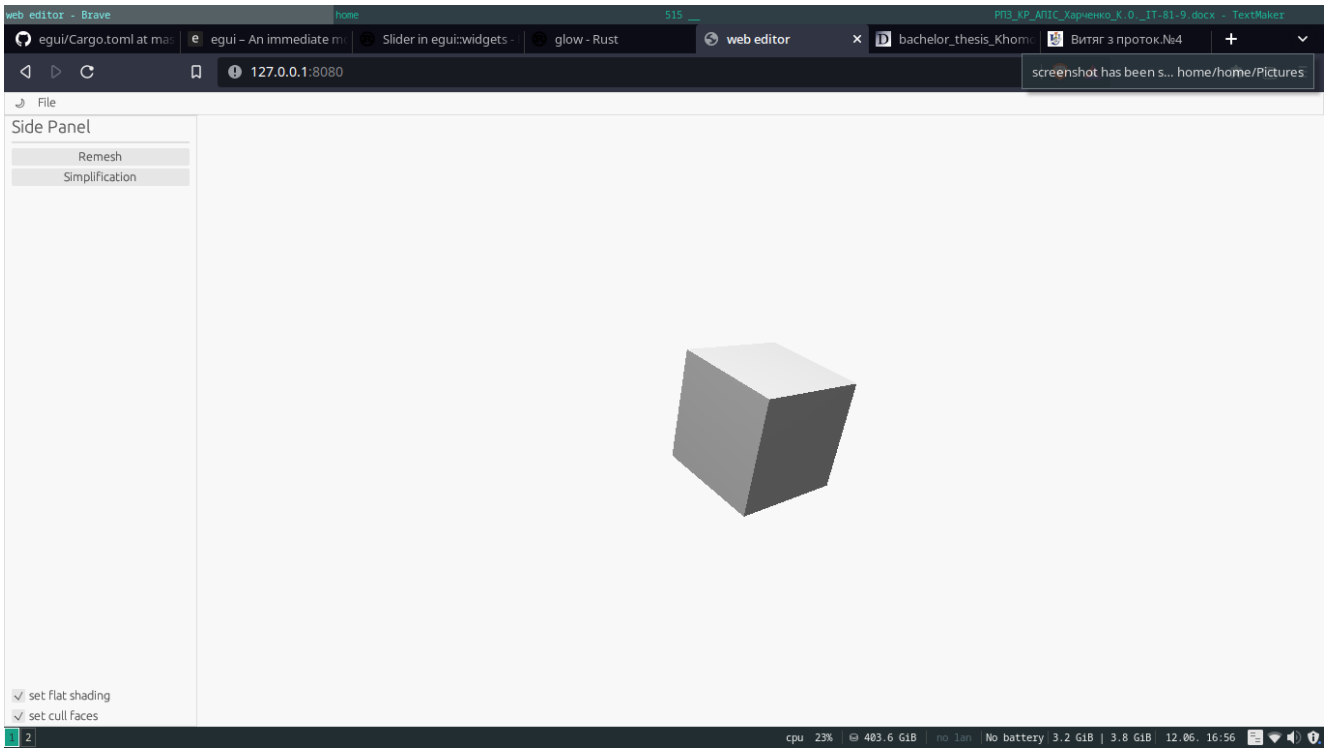


Рисунок 3.2 – Інтерфейс (біла тема)

3.2 Програмна реалізація

Для початку треба додати можливість відображення геометрії за допомогою WebGL. Для цього потрібно створити буфера на відеокарті та шейдери.

Лістинг коду вершинного шейдера:

```
layout (location = 0) in vec3 in_position;
layout (location = 1) in vec3 in_normal;

out vec3 vs_out_pos;
out vec3 vs_out_unproject_pos;
out vec3 vs_out_normal;

uniform mat4 u_model;
uniform mat4 u_view;
uniform mat4 u_proj;

void main() {
    vs_out_pos = vec3(u_view * u_model * vec4(in_position.xyz,
1.0));
```

```
    vs_out_normal  =  mat3(transpose(inverse(u_view  *  u_model)))  *
in_normal;
    gl_Position = u_proj * u_view * u_model * vec4(in_position.xyz,
1.0);
}
```

Лістинг коду фрагментного шейдера:

```
precision mediump float;

in vec3 vs_out_pos;
in vec3 vs_out_normal;

out vec4 out_color;

uniform vec3 u_light_pos;
uniform vec3 u_camera_pos;
uniform vec4 u_color;

uniform int u_is_flat_shading;

void main() {
    vec3 normal;
    if (u_is_flat_shading == 0) {
        normal = normalize(vs_out_normal);
    } else {
        normal          =          normalize(cross(dFdx(vs_out_pos),
dFdy(vs_out_pos)));
    }

    vec3 light_dir = normalize(u_light_pos - vs_out_pos);
    vec3 light_color = vec3(1.0, 1.0, 1.0);

    vec3 view_dir = normalize(u_camera_pos - vs_out_pos);
    vec3 reflect_dir = reflect(-light_dir, normal);

    float ambient_strength = 0.1;
    vec3 ambient = ambient_strength * light_color;

    float diff = max(dot(normal, light_dir), 0.0);
    vec3 diffuse = diff * light_color;

    float specular_strength = 0.5;
    float spec = pow(max(dot(view_dir, reflect_dir), 0.0), 32.0);
    vec3 specular = specular_strength * spec * light_color;

    vec3 color = (ambient + diffuse + specular) * u_color.rgb;

    out_color = vec4(color, u_color.a);
}
```

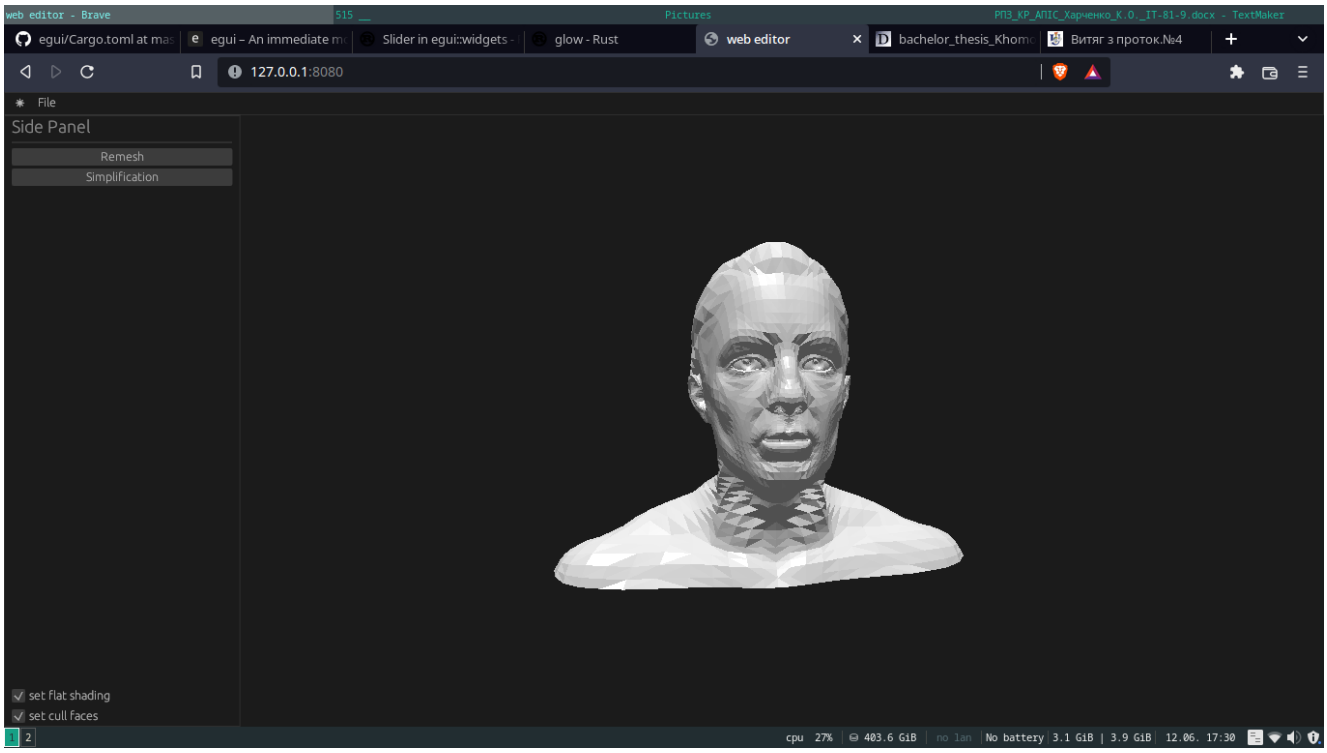


Рисунок 3.3 – Приклад освітлення для нормалей трикутників (flat shading)

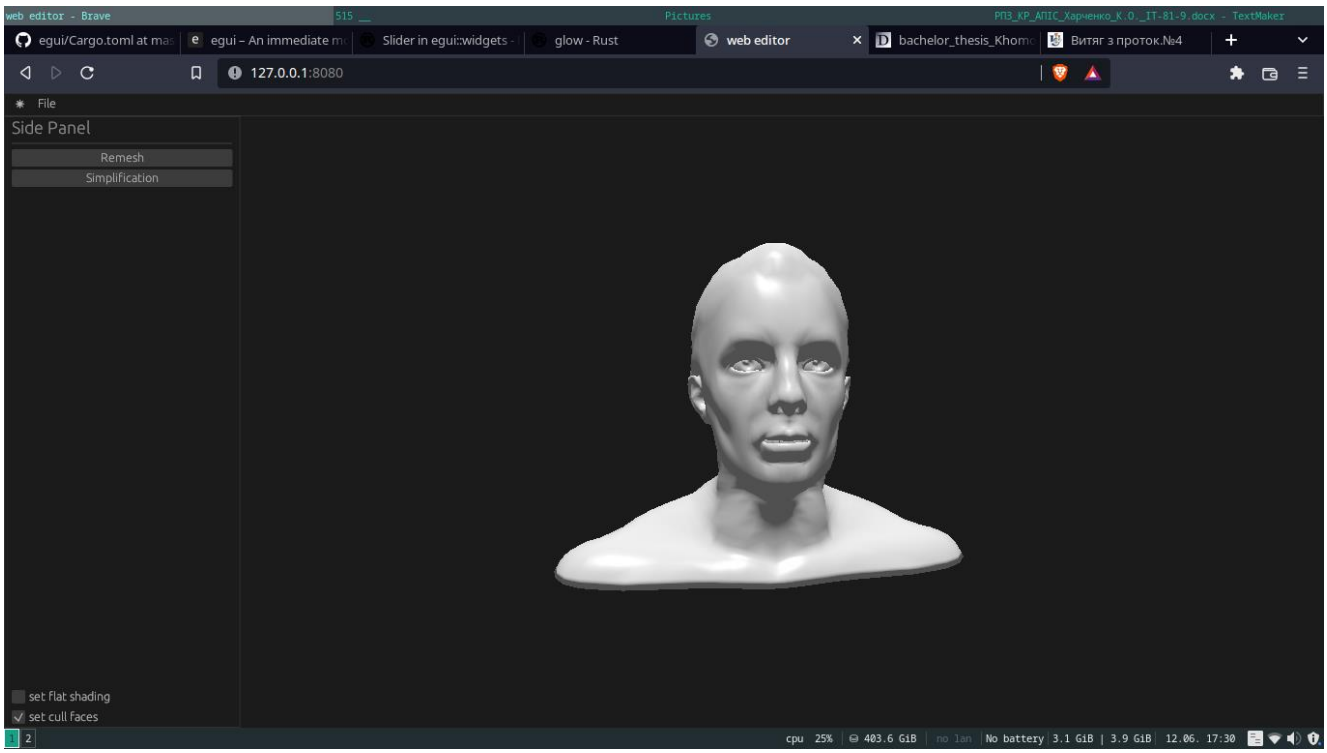


Рисунок 3.4 – Приклад освітлення для нормалей вершин

Також потрібно створити камеру для навігації та обробляти події з пристроїв вводу/виводу. Для того, щоб користувач міг завантажити моделі з/до

застосунку було використано парсери для форматів .STL та .PLY. Для цього було створено меню “File” (рис 3.5) з якого можна завантажити моделі. Також для завантаження моделей в редактор працює функція “Drag-and-drop”.

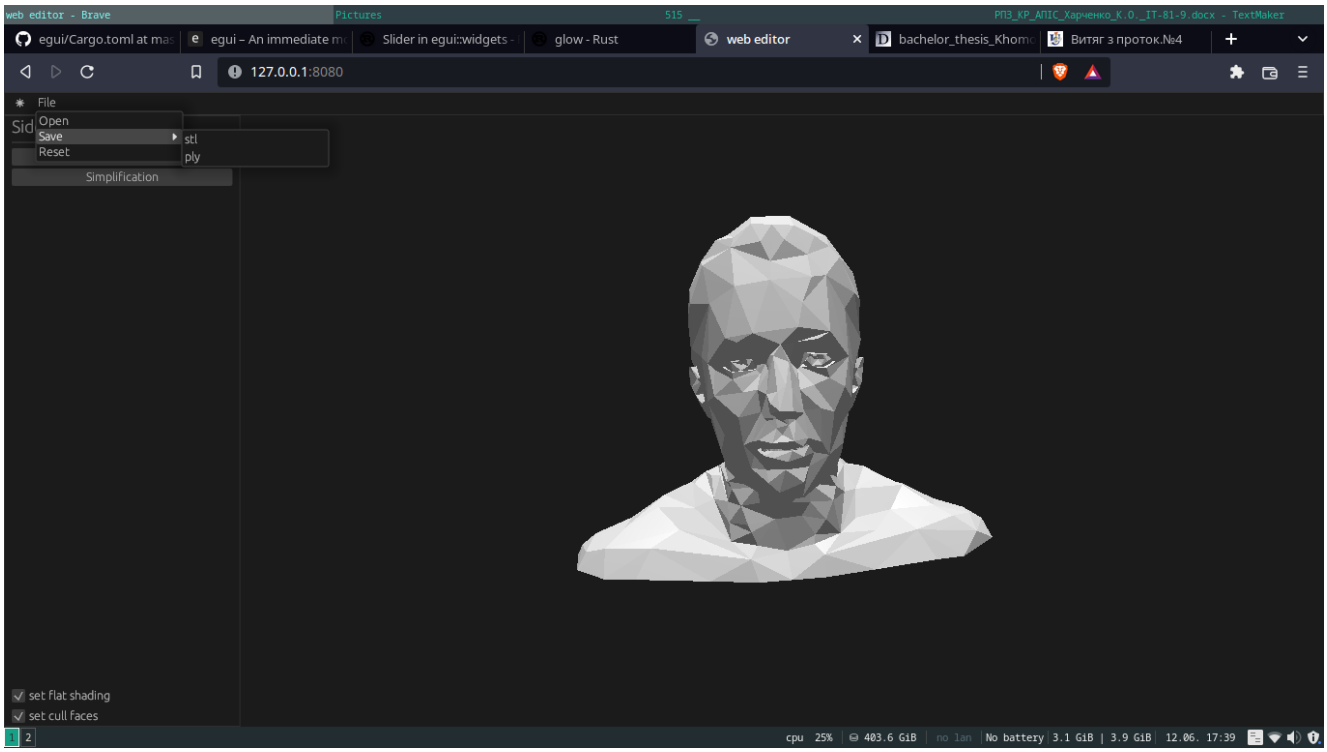


Рисунок 3.5 – Приклад верхнього меню для завантаження моделей

Основною проблемою реалізації експорту/імпорту є те що JavaScript не може взаємодіяти з файловою системою напряму – це зроблено для безпеки користувачів. Тому для реалізації механізму експорту/імпорту було використано можливості html для завантаження файлів з/до застосунку за допомогою посилань.

Лістинг коду для збереження файлів на диску користувача:

```
fn save_file_binary(filename: &str, content: Vec<u8>) -> bool {
    let window = web_sys::window();
    if window.is_none() {
        return false;
    }
    let window = window.unwrap();

    let document = window.document();
```

```
    if document.is_none() {
        return false;
    }
    let document = document.unwrap();

    let element = document.create_element("a");
    if element.is_err() {
        return false;
    }
    let element = element.unwrap();

    let a_element =
element.dyn_into::<web_sys::HtmlAnchorElement>();
    if a_element.is_err() {
        return false;
    }
    let a_element = a_element.unwrap();

    let body = document.body();
    if body.is_none() {
        return false;
    }
    let body = body.unwrap();

    let append_child = body.append_child(&a_element);
    if append_child.is_err() {
        return false;
    }

    let uint8arr = js_sys::Uint8Array::new(&unsafe {
js_sys::Uint8Array::view(&content) }.into());
    let blob_content = js_sys::Array::new();
    blob_content.push(&uint8arr.buffer());

    let blob =
web_sys::Blob::new_with_u8_array_sequence_and_options(
        &blob_content,
        web_sys::BlobPropertyBag::new().type_("application/octet-
stream")
    );
    if blob.is_err() {
        return false;
    }
    let blob = blob.unwrap();

    let url = web_sys::Url::create_object_url_with_blob(&blob);
    if url.is_err() {
        return false;
    }
    let url = url.unwrap();

    a_element.set_href(&url);
```

```
    a_element.set_download(filename);
    a_element.click();
    a_element.remove();

    true
}
```

В якості алгоритму для ремешу було використано лінійний алгоритм, який додає 3 додаткові трикутники замість наявного (рис. 3.6, 3.7).

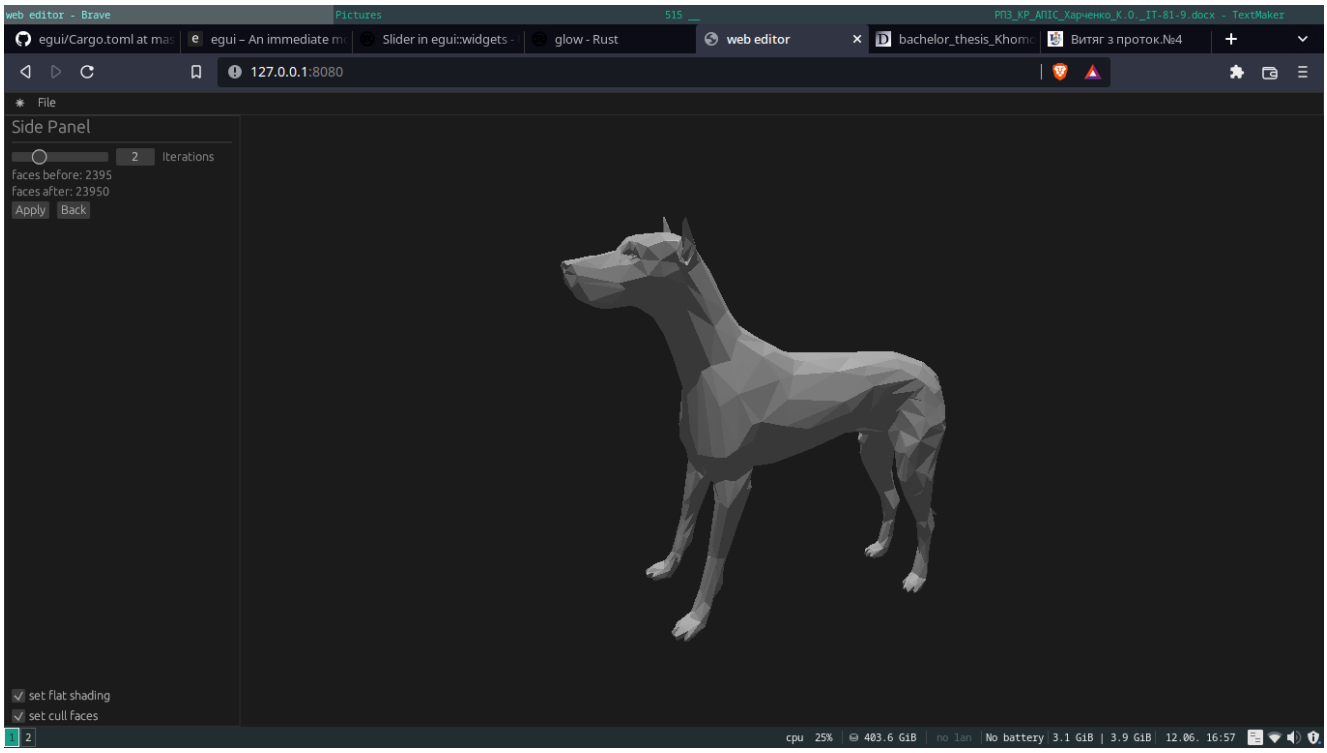


Рисунок 3.6 – Приклад меню розділення трикутників моделі

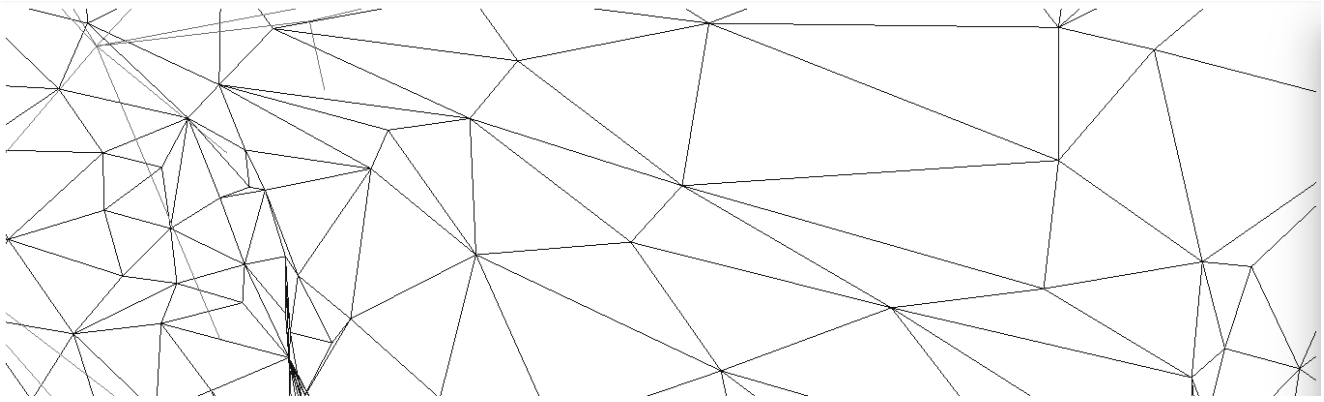


Рисунок 3.7 – Приклад сітки до розділення трикутників

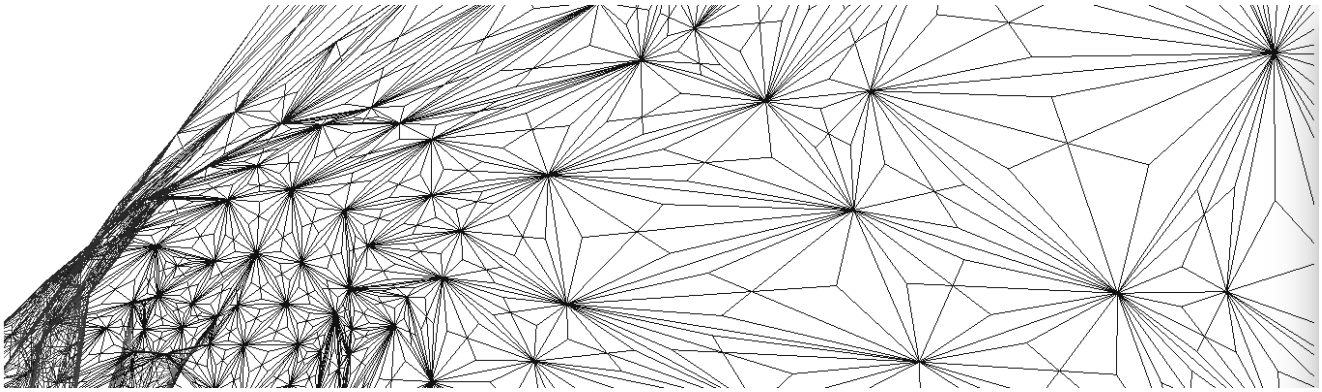


Рисунок 3.8 – Приклад сітки після розділення трикутників

Для спрощення (децимації) мешу було взято метрику квадратичної помилки та приклад реалізації на мові C++[15].

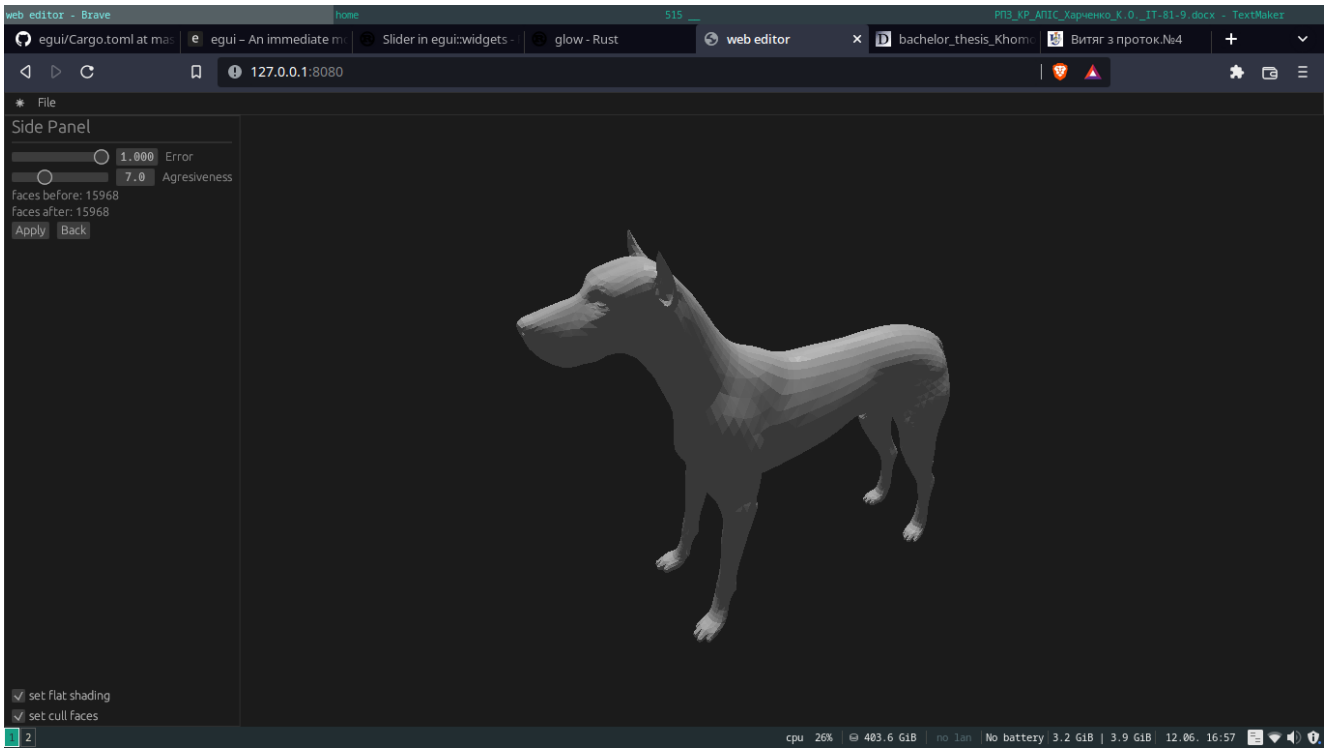


Рисунок 3.9 – Модель “собаки” до спрощення

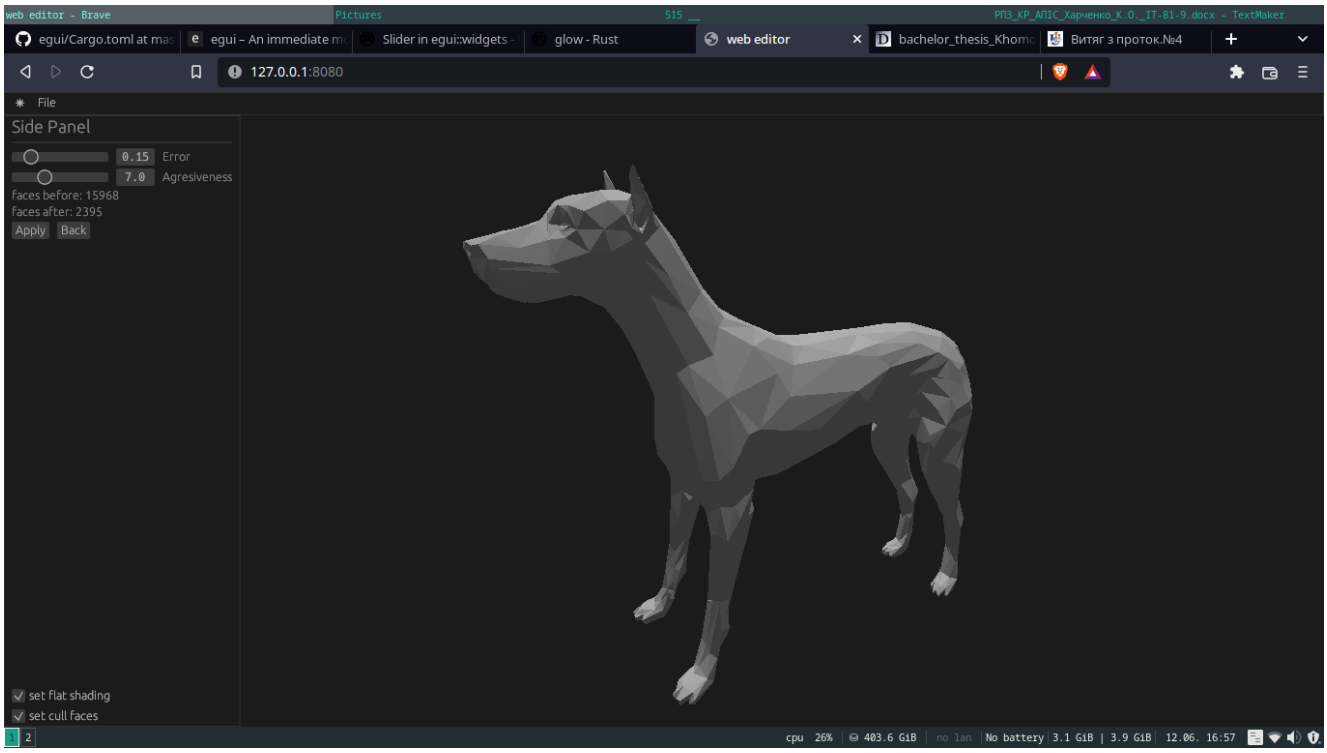


Рисунок 3.10 – Модель “собаки” після спрощення

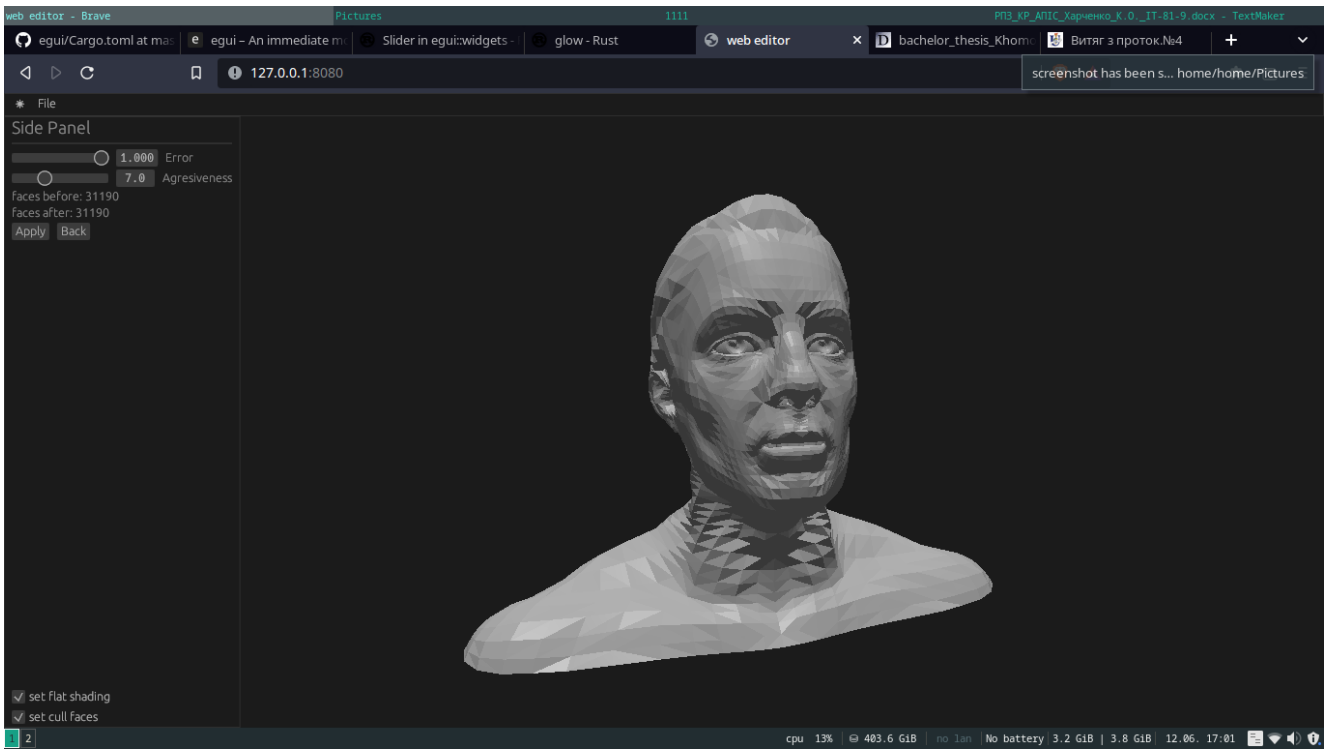


Рисунок 3.11 – Модель “голови” до спрощення

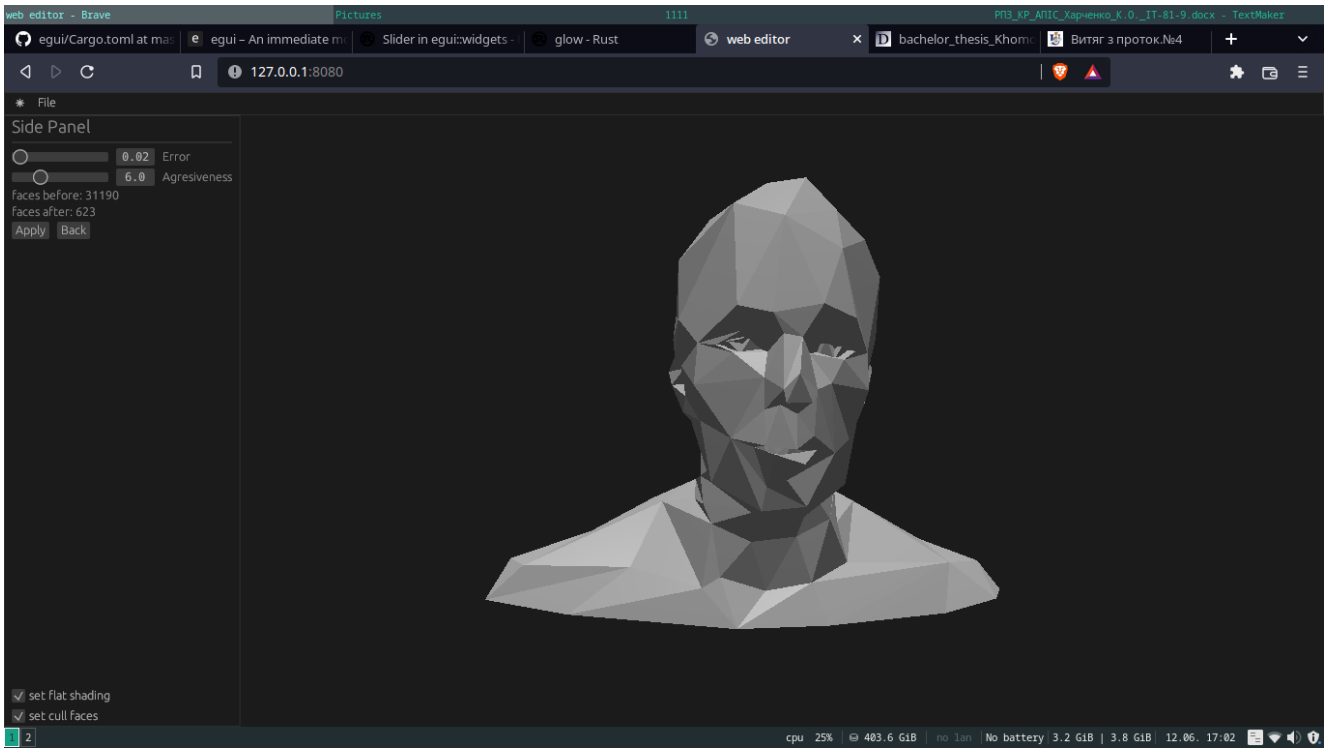


Рисунок 3.12 – Модель “голови” після спрощення

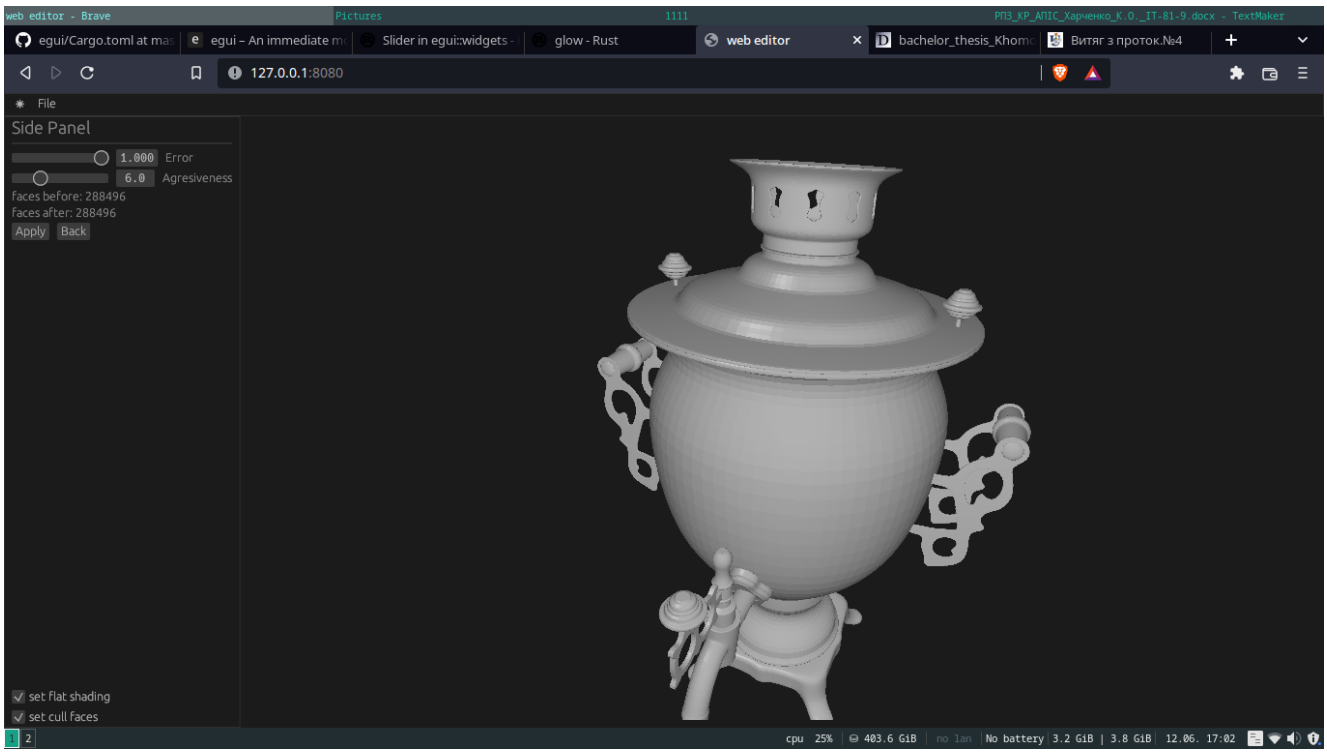


Рисунок 3.13 – Модель “самовару” до спрощення

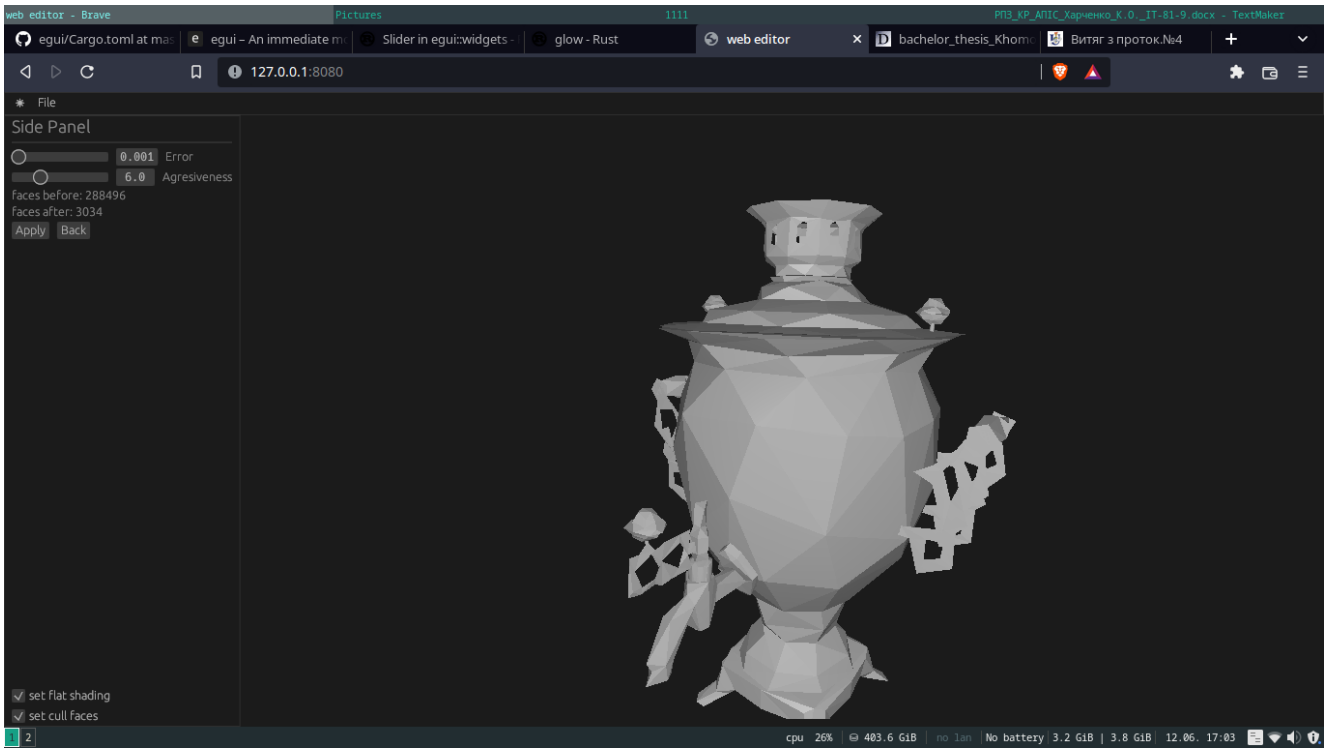


Рисунок 3.14 – Модель “самовару” після спрощення

Основною ідеєю алгоритму є мінімізація квадратичної помилки для вершин - обираємо пари вершини довжина котрих менша за певне значення та на основі квадратичної помилки з’єднуємо 2 вершини в одну. Під час з’єднання вершин може бути видалено від 1 до 2 трикутників.

3.3 Використання програмного додатку

Для початку роботи потрібно завантажити 3d модель у додаток у форматі .STL або .PLY (рис 3.15).

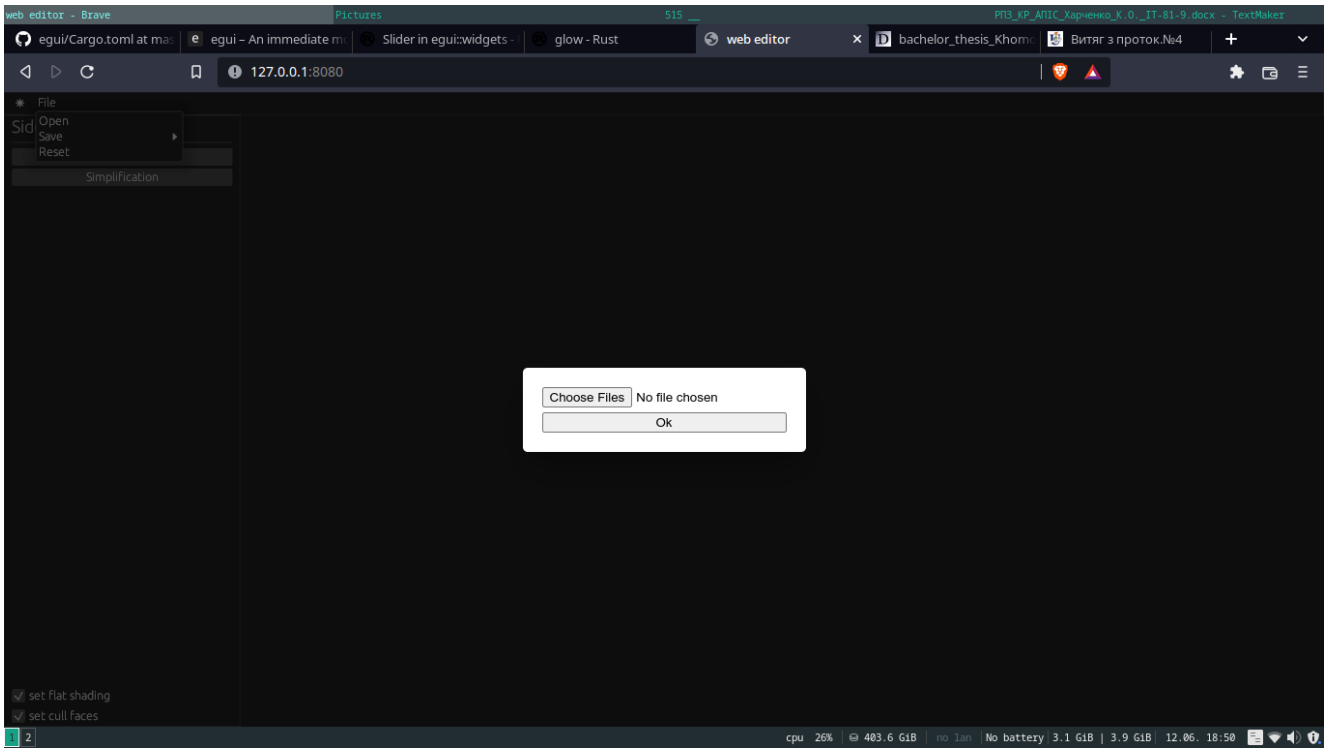


Рисунок 3.15 – Завантаження моделі

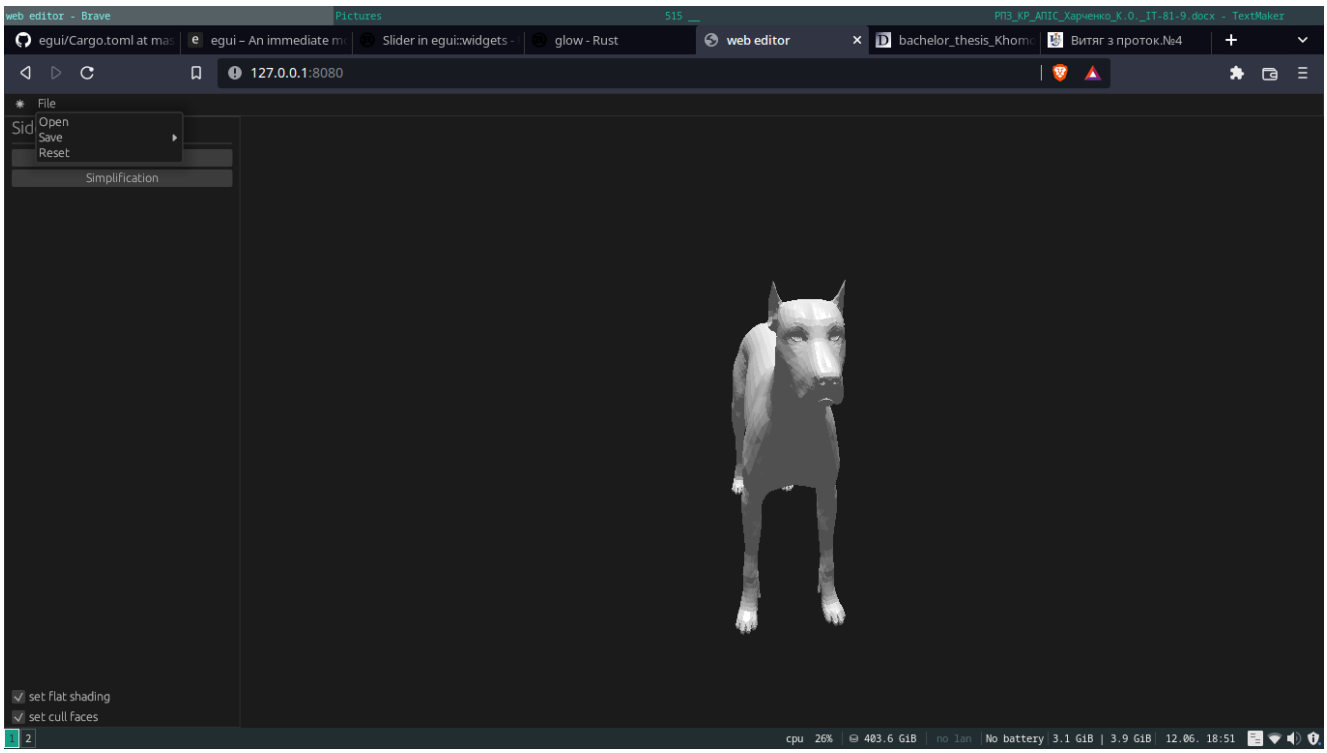


Рисунок 3.16 – Завантажена модель

Після завантаження моделей можливо перейти у меню спрощення (рис. 3.17) або розділення трикутників (рис. 3.18).

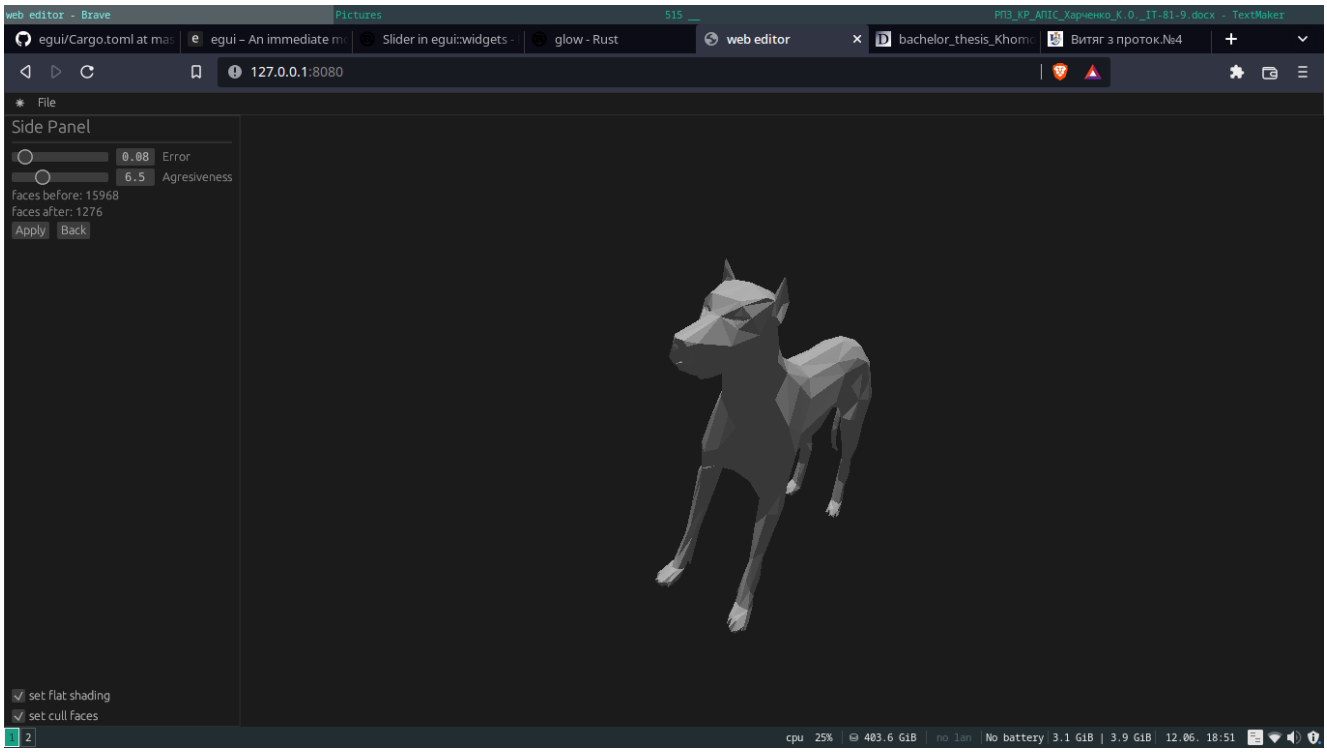


Рисунок 3.17 – Спрощена модель

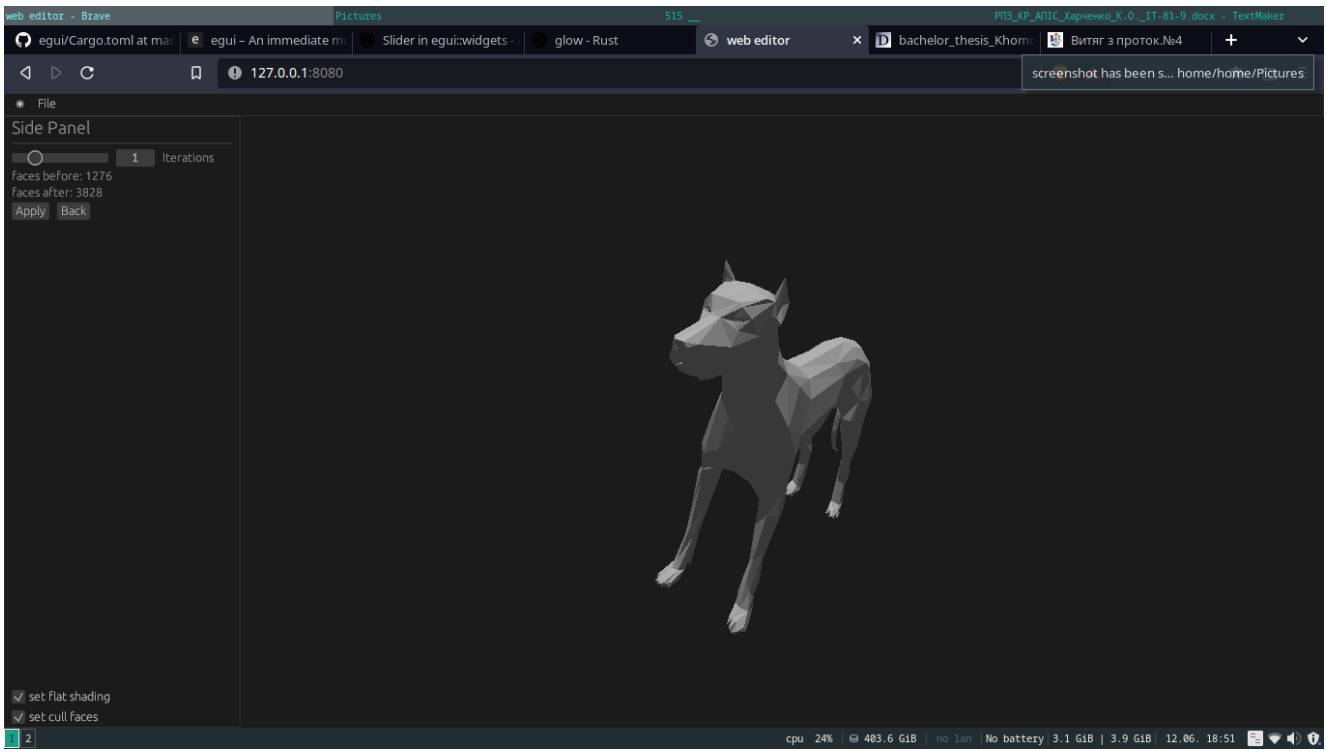


Рисунок 3.18 – Модель після розділення трикутників

Після закінчення редагування моделі можливо її зберегти у форматі .STL, .PLY (рис 3.20).

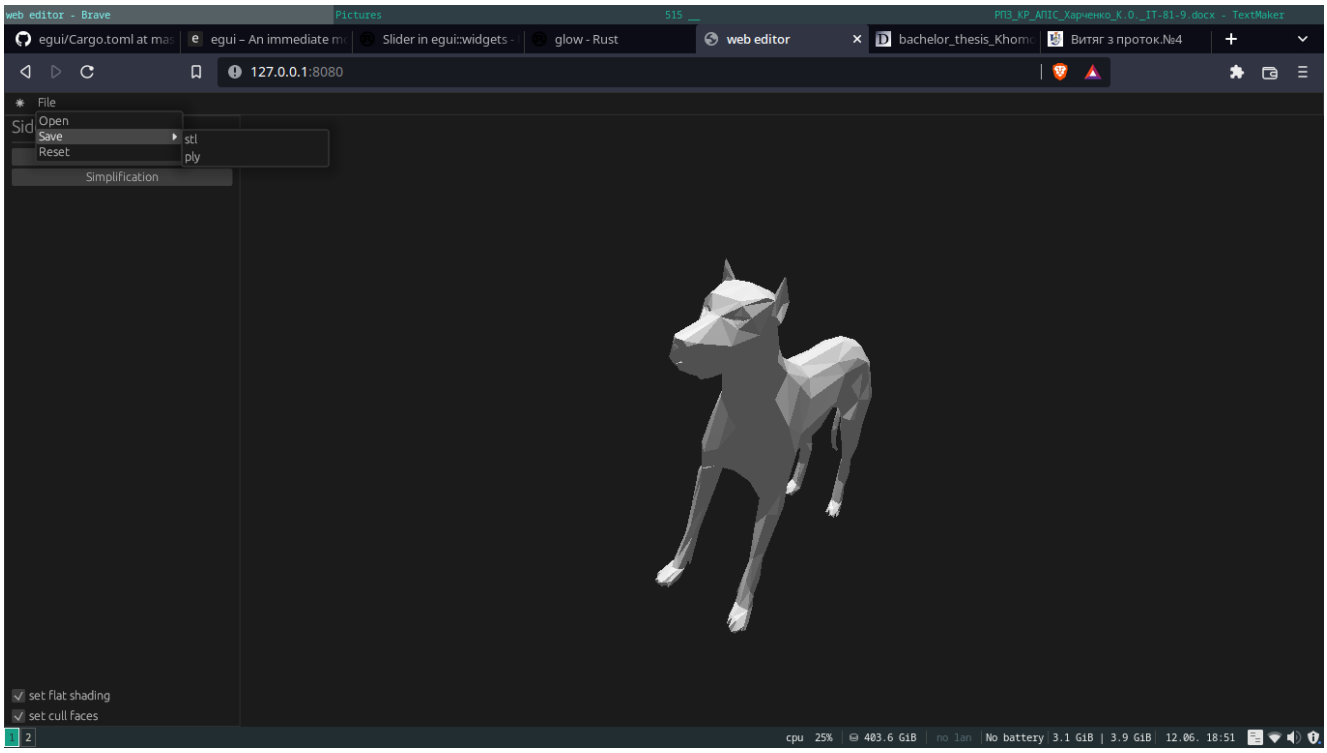


Рисунок 3.19 – Меню збереження моделі

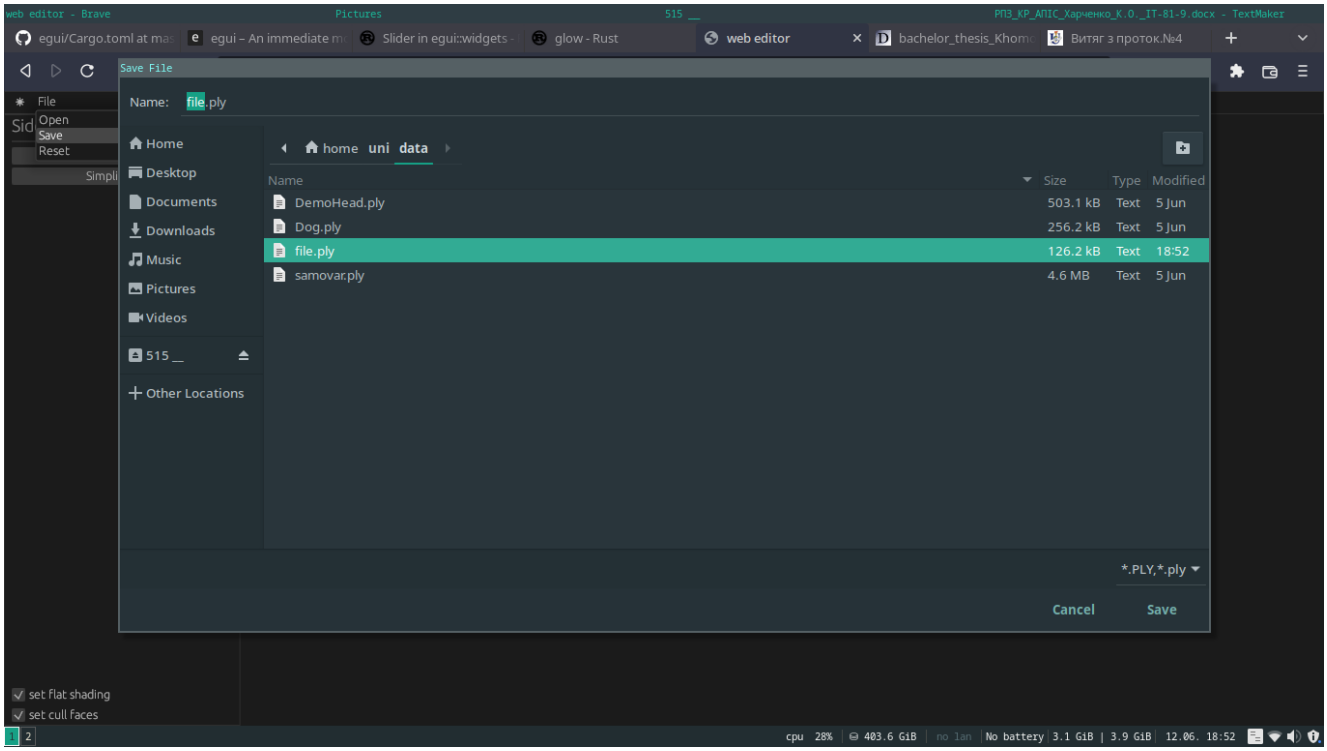


Рисунок 3.20 – Вікно збереження моделі

ВИСНОВКИ

В ході виконання дипломної роботи було проведено огляд додатків аналогів (“SketchUp”, “Tinkercad”, “Clara.io”) для виявлення особливостей. В результаті чого було сформовано порівняльну таблицю, що надає змогу під час розробки звернути увагу на цікаві функціональні доповнення, які можна використати, і недоліки, які варто подолати.

Було проведено аналіз наявних технологій. Під час чого виділено основні засоби розробки, а саме: WebGL – для роботи з тривимірною графікою, JavaScript – для зв’язку з модулями WASM, WASM – мова програмування для доповнення JavaScript та реалізації алгоритмів для підвищення продуктивності, Rust – як основна мова для написання застосунку, що буде компілюватися у WASM.

Було сформовано мету та основні вимоги для реалізації проєкту, розроблено технічне завдання. У технічному завданні було виділено: мету, призначення, цільову аудиторію, загальні та апаратні вимоги, описано етапи розроблення застосунку.

Також було виконано планування робіт, що дозволяє обсяг роботи та наявні ризики. Під час планування робіт створено матрицю відповідальності, календарний графік, матрицю ризиків та таблицю реагування на ризики.

Проведено функціональне моделювання онлайн редактору 3d моделей в нотації IDEF0, декомпозицію функціональної моделі та діаграму варіантів використання.

Було створено web-додаток для редагування 3d моделей, що дозволяє завантажувати моделі в форматах .STL, .PLY, переглядати модель та виконувати спрощення моделі, розділяти трикутники на більшу кількість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WebAssembly [Електронний ресурс] – Режим доступу до ресурсу: https://webassembly.org/ (дата звернення: 12.06.2022).

2. WebGL [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/WebGL_API (дата звернення: 12.06.2022).

3. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 12.06.2022).

4. WebAssembly Is Fast: A Real-World Benchmark of WebAssembly vs. ES6 [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@torch2424/webassembly-is-fast-a-real-world-benchmark-of-webassembly-vs-es6-d85a23f8e193 (дата звернення: 12.06.2022).

5. HTML: HyperText Markup Language [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 12.06.2022).

6. SketchUp [Електронний ресурс] – Режим доступу до ресурсу: https://www.sketchup.com/ru (дата звернення: 12.06.2022).

7. Tinkercad [Електронний ресурс] – Режим доступу до ресурсу: https://www.tinkercad.com/ (дата звернення: 12.06.2022).

8. Clara.io [Електронний ресурс] – Режим доступу до ресурсу: https://clara.io/scenes (дата звернення: 12.06.2022).

9. Rust [Електронний ресурс] – Режим доступу до ресурсу: https://www.rust-lang.org/ (дата звернення: 12.06.2022).

10. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com/ (дата звернення: 12.06.2022).

11. Gallant G. WebAssembly in Action. Manning Publications Company, 2019. 425 p.

12. Nichols C., Klabnik S. Rust Programming Language. No Starch Press, Incorporated, 2019. 560 p.

13. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. Upper Saddle River, New Jersey : Addison-Wesley, 2013.

14. GitHub - emilk/egui: egui: an easy-to-use immediate mode GUI in Rust that runs on both web and native. *GitHub*. URL: https://github.com/emilk/egui (дата звернення: 12.06.2022).

15. GitHub - sp4cerat/Fast-Quadric-Mesh-Simplification: mesh triangle reduction using quadrics. *GitHub*. URL: https://github.com/sp4cerat/Fast-Quadric-Mesh-Simplification (дата звернення: 12.06.2022).

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ «Web-додатку для редагування тривимірних моделей»

1. Призначення й мета «Web-додатку для редагування тривимірних моделей»

1.1 Призначення створення «Web-додатку для редагування тривимірних моделей»

Web-додатку редагування тривимірних моделей призначений для перегляду 3d моделей в браузері та виконання операцій спрощення та розділення трикутників.

1.2 Мета створення «Web-додатку для редагування тривимірних моделей»

Метою дипломного проєкту є розробка Web-додатку для редагування тривимірних моделей.

1.3 Цільова аудиторія

Цільовою аудиторією даного проєкту є користувачі котрі зацікавлені в Web-додатку для редагування тривимірних моделей.

2. Вимоги до проєкту

2.1 Загальні вимоги до редактору

Створення операцій спрощення геометрії та розділення трикутників на меншу к-сть. Можливість імпорту/експорту моделей в формати .STL, .PLY.

2.2 Вимоги до апаратного забезпечення

Для використання Web-додатку для редагування тривимірних моделей та стабільної роботи, апаратне забезпечення повинно задовольняти наступні рекомендовані вимоги:

- Процесор: 64-bit Intel, AMD;
- Відеокарта GeForce GTX 950;
- Оперативна пам’ять: 6 GB;
- Браузер: Google Chrome (версія 57), Mozilla Firefox (версія 52), Safari (версія 11), Microsoft Edge (версія 16).

3. Склад і зміст робіт зі створення Web-додатку для редагування тривимірних моделей

Детальний опис етапів створення Web-додатку для редагування тривимірних моделей в таблиці А.2.

Таблиця А.2 – Етапи розробки проєкту.

№	Склад і зміст робіт	Строк розробки
1	Розробка архітектури додатку	4 дні
2	Розробка інтерфейсу	5 днів
3	Розробка модулю графіки	5 днів

Продовження таблиці А.2

№	Склад і зміст робіт	Строк

		розробки
4	Розробка модулю для роботи з файлами із модулів WASM	10 днів
5	Розробка модулю імпорту/експорту з форматів stl, ply	5 днів
6	Розробка алгоритмів ремешу та децимації	14 днів
8	Розробка модулю для серверної частини	10 днів
9	Тестування	14 днів
	Загальна тривалість	67 днів

ДОДАТОК Б – ПЛАНУВАННЯ РОБІТ НА РОЗРОБКУ «Web-додатку для редагування тривимірних моделей»

Метою дипломного проєкту є розробка Web-додатку для редагування тривимірних моделей. Реалізація даного проєкту дозволить створити Web-додаток для редагування тривимірних моделей. Також дозволить зрозуміти доцільність та ефективність використання технологій (WebGL, WASM).

Для досягнення мети проєкту необхідно зробити:

– визначити актуальність роботи, дослідити предметну область;

– провести аналіз наявних публікацій та аналогічних проєктів для виявлення особливостей;

– обрати та проаналізувати технології для розробки проєкту;

– сформулювати постановку задачі та основні вимоги для реалізації проєкту;

– провести структурно-функціональний аналіз виконання проєкту, розробити контекстну діаграму, діаграму декомпозиції та варіантів використання;

– розробити архітектуру додатка;

– створити інтерфейс для проєкту;

– виконати тестування додатка.

Реалізація даного проєкту дозволить використовувати механізми для роботи з топологією 3d моделей в браузері. Також дозволить зрозуміти доцільність та ефективність використання технологій (WebGL, WASM).

Деталізація мети проєкту методом SMART.

Технологія SMART – метод опису мети, що містить в собі: конкретність, вимірність, досягнень, важливість і визначеність за терміном. Результати деталізації мети даного проєкту представлено в таблиці 1.

Таблиця Б.1. Деталізація мети проєкту методу SMART.

Specific	Створити онлайн додаток для редагування 3d моделей
Measurable	Створений додаток за допомогою якого можна виконувати редагування 3d моделей
Achievable	Створити редактор з потужним функціоналом та продуктивністю порівняною з версією для локального комп’ютера
Relevant	Для ремешу та децимації 3d моделей, зрозуміти ефективність використання технологій (WebGL, WASM)
Time-framed	До кінця 4 курсу (06.06.2022)

Планування змісту робіт. WBS- це графічне подання згрупованих елементів проєкту у вигляді пакетів робіт, які ієрархічно пов’язані з продуктом проєкту. WBS необхідна для забезпечення ефективного управління проєктом, визначення і структурування переліку робіт, створення структури звітності, розуміння задач виконавцем.

WBS є базовим засобом для створення організаційної структури (OBS) і системи управління проєктом, оскільки дозволяє виявити проблеми організації робіт, визначення ієрархії проєктних завдань (етапів робіт), підзадач і пакетів робіт на всіх подальших фазах життєвого циклу проєкту.

WBS дозволяє розподілити відповідальність за досягнення цілей проєкту між його виконавцями та тим самим гарантувати, що всі роботи за проєктом мають відповідальних і не випадуть з поля зору. WBS забезпечує членам команди розуміння загальних цілей і завдань за проєктом. Важливо врахувати всі роботи в проєкті і кількість рівнів деталізації визначається достатністю для планування та моніторингу робіт проєкту.

WBS повинна давати команді управління проєктом та замовнику чітку картину кінцевого продукту проєкту та всіх процесів, за допомогою яких він буде створений.

Основне призначення WBS- структури - визначити зміст проєкту через декомпозицію його продуктів. Кожна WBS-структура є інструментом, який допомагає керівникові проєкту здійснити декомпозицію робіт проєкту до рівня, необхідного для досягнення його цілей.

Планування змісту структури робіт даного IT-проєкту (WBS) здійснювалося за допомогою програми WBS Schedule. На рисунку Б.1 представлено WBS - структура робіт проєкту.

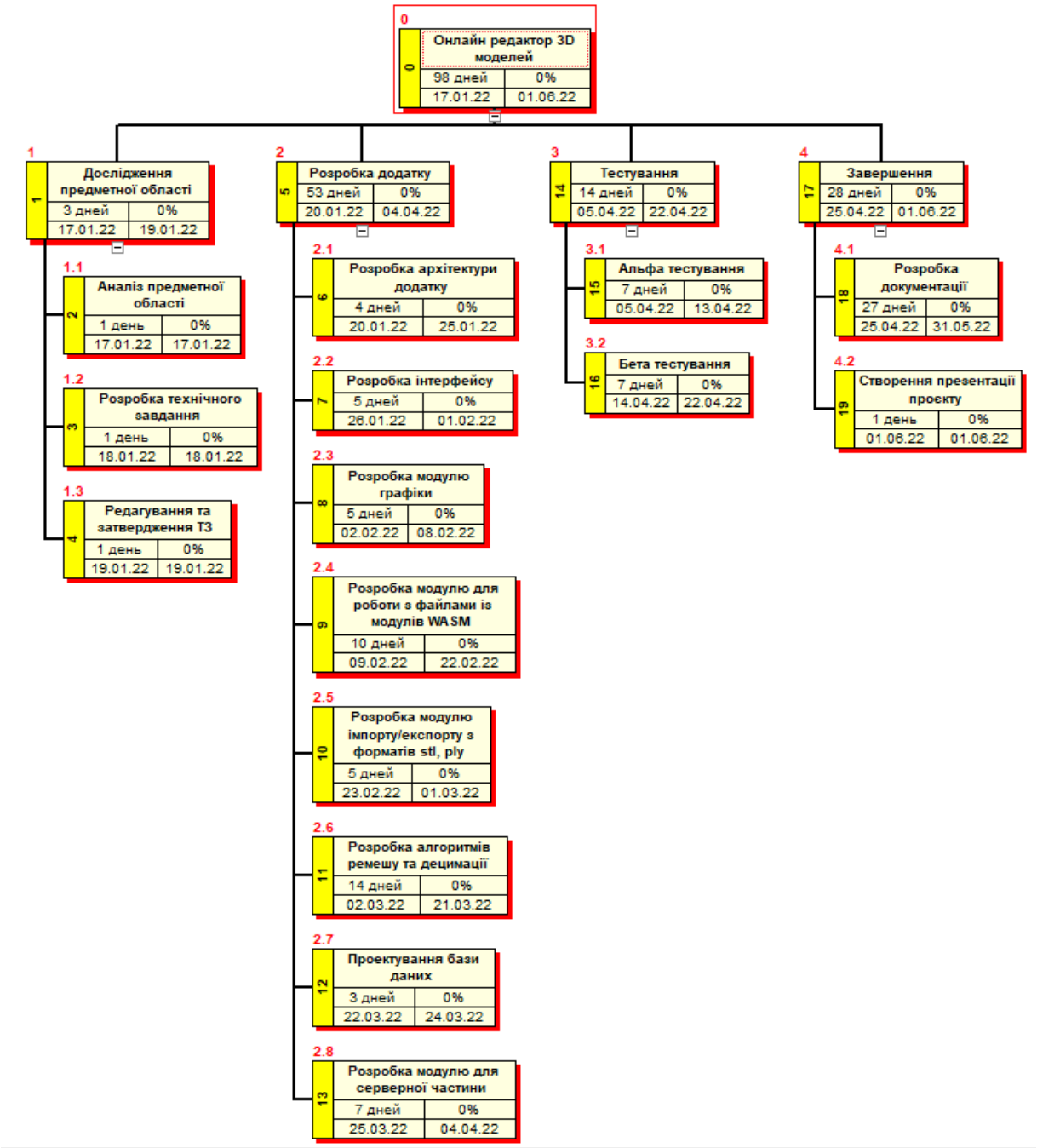


Рисунок Б.1 – WBS структура

Планування структури виконавців. OBS - графічне відображення учасників проєкту (фізичних та юридичних осіб) та їхніх відповідальних осіб, залучених до реалізації проєкту. Елементами OBS можуть бути: окремі виконавці (керівники, фахівці, службовці), організації, структурні підрозділи і служби, у яких зайнята та або інша кількість фахівців, що виконують певні функціональні обов'язки, зовнішні постачальники обладнання, послуг та інші організації.

OBS є найважливішим механізмом управління проєктами. OBS дає можливість реалізувати всі пакети робіт, передбачені WBS-структурою, а також функції, процеси й операції, необхідні для досягнення поставлених перед проєктом цілей. OBS є основою формування і здійснення діяльності команди проєкту.

Слід відрізняти організаційну структуру організації та організаційну структуру проєкту. На етапі планування, коли розробляють OBS-структуру проєкту, досить часто невідомо, які конкретні організації та їхні відповідальні особи будуть залучені до проєкту. Відповідь на це запитання буде отримана тільки після проведення відповідних тендерів на виконання робіт. Тому попередньо в OBS-структуру вводять умовні позначення виконавців та їх відповідальних осіб, які потім змінюють на конкретні дійсні назви та прізвища.

Планування змісту структури організації (OBS) здійснювалося за допомогою програми WBS Schedule. На рисунку Б.2 представлено OBS - структура робіт проєкту.

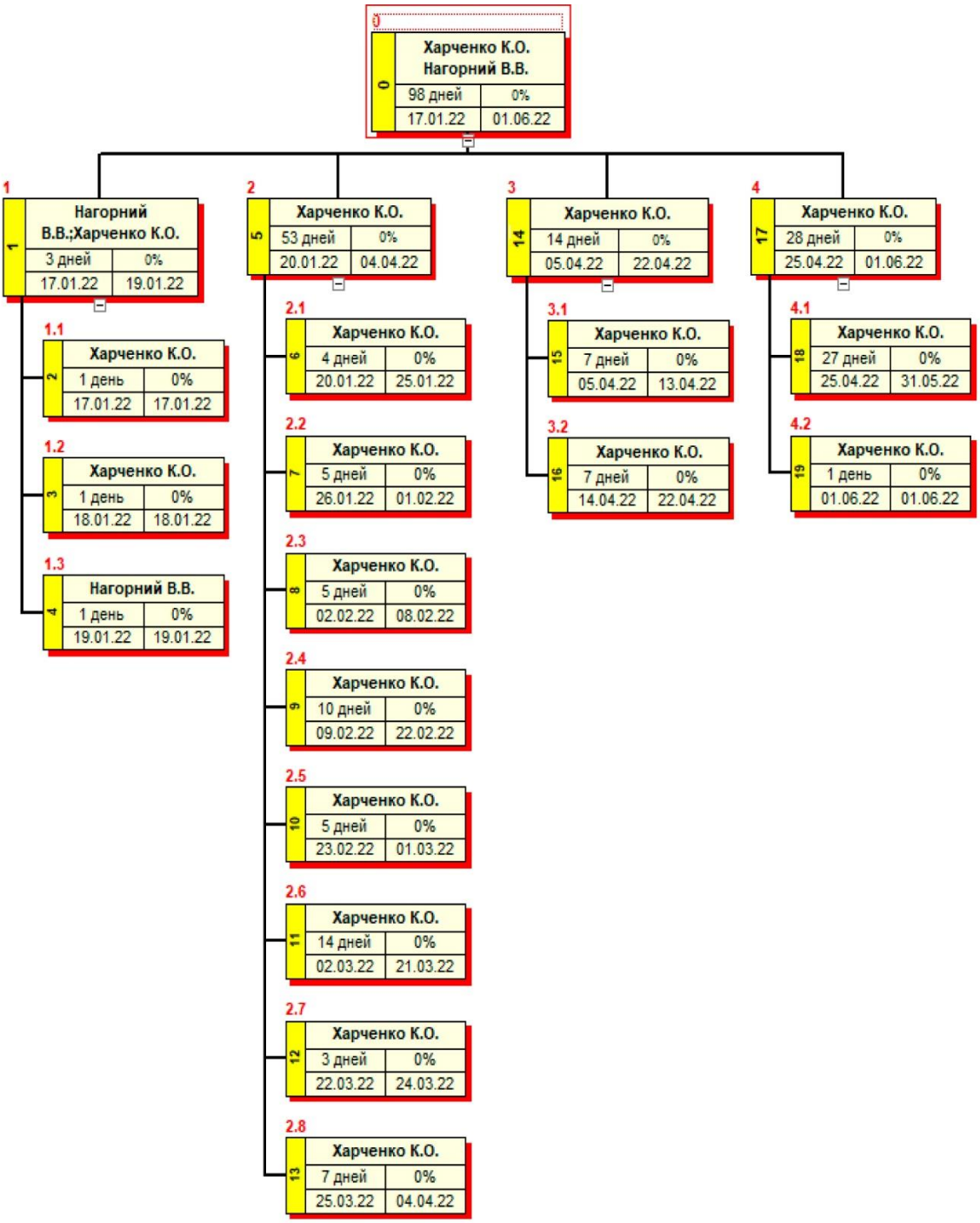


Рисунок Б.2 – OBS структура

Представлення списку виконавців. Для управління проєктами важливо знати, хто за яку роботу в проєкті відповідає. Для цього використовують такий інструмент, як матрицю відповідальності (або лінійний графік відповідальності) Її будують на підставі розроблених WBS та OBS структур.

Зазвичай OBS структуру представляють у вигляді горизонтальних рядків матриці, а WBS- структуру - у вигляді вертикальних стовпчиків.

Основні правила складання RAM: за одним пакетом робіт не може бути закріплено кілька відповідальних, але один відповідальний може відповідати за декілька робіт. Для кожного елемента WBS-структури визначають особу, яка відповідатиме за її виконання. Тобто матриця повинна давати відповідь на запитання: хто відповідає за виконання кожного елемента WBS-структури нижчого рівня. Потрібно пам'ятати, що за виконання одного WBS-елемента може відповідати лише одна особа. Але одна особа може відповідати за виконання декількох WBS-елементів.

Досить часто матрицю відповідальності також створюють для визначення розподілу адміністративних завдань управління. Для цього, крім відповідальних та виконавців у ній враховують інших учасників проєкту, яким доручають функції контролю, консультування, активізації виконання тощо. З цією метою в матриці використовують умовні позначення у вигляді літер. У міжнародних проєктах заведено використовувати літери.

Матриця відповідальності (RAM) будувалась за допомогою програми WBS Schedule. Матриця відповідальності представлена в таблиці Б.2.

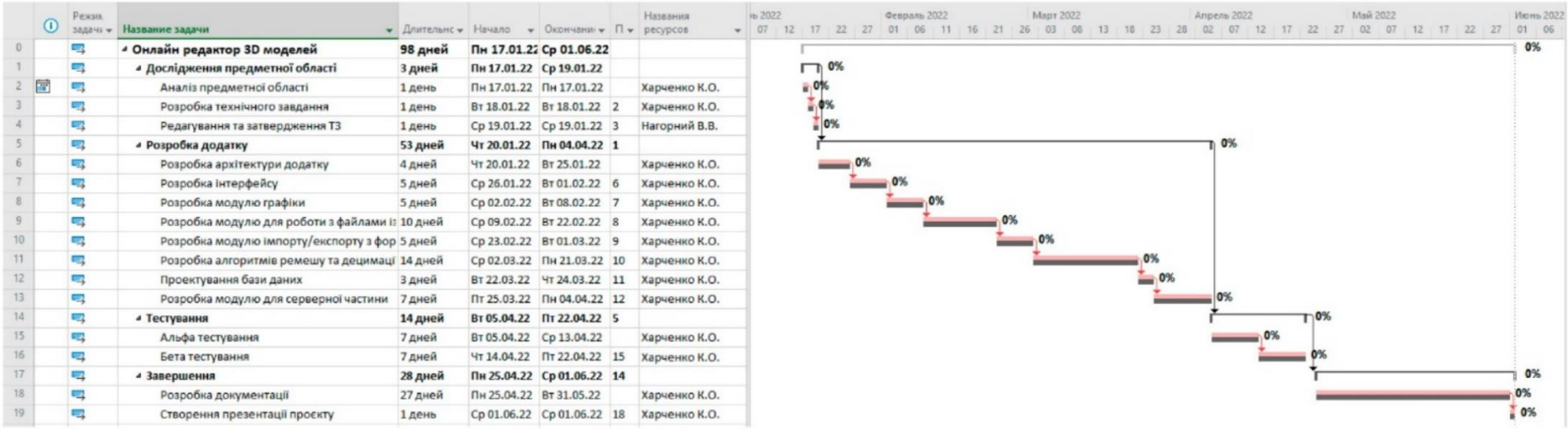
Таблиця Б.2 – Матриця відповідальності

WBS/OBS	Харченко К.О.	Нагорний В.В.
Розробка Web-додатоку для редагування тривимірних моделей		
1.1 Дослідження предметної області		
1.1 Аналіз предметної області	+	
1.2 Розробка технічного завдання	+	

Продовження таблиці Б.2

WBS/OBS	Харченко К.О.	Нагорний В.В.
Розробка Web-додатоку для редагування тривимірних моделей		
1.3 Редагування та затвердження ТЗ		+
2 Розробка додатку		
2.1 Розробка архітектури додатку	+	
2.2 Розробка інтерфейсу	+	
2.3 Розробка модулю графіки	+	
2.4 Розробка модулю для роботи з файлами із модулів WASM	+	
2.5 Розробка модулю імпорту/експорту з форматів stl, ply	+	
2.6 Розробка алгоритмів ремешу та децимації	+	
2.7 Розробка модулю для серверної частини	+	
3 Тестування		
3.1 Альфа тестування	+	
3.2 Бета тестування	+	
4 Завершення		
4.1 Розробка документації	+	
4.2 Створення презентації проєкту	+	

Діаграма Ганта. На діаграмі Ганта всі роботи за проєктом представлені у вигляді горизонтальних відрізків, паралельних осі часу. Використання моделі проєкту, побудованої в програмному середовищі MS Project, дозволяє контролювати й оптимізувати план виконання робіт, наочно відстежувати хід його виконання. Календарний графік представлено на рисунку Б.3.



	Режим задачи	Название задачи	Длительнс	Начало	Окончани	П	Названия ресурсов
0		**Онлайн редактор 3D моделей**	**98 дней**	**Пн 17.01.22**	**Ср 01.06.22**		
1		**Дослідження предметної області**	**3 дней**	**Пн 17.01.22**	**Ср 19.01.22**		
2		Аналіз предметної області	1 день	Пн 17.01.22	Пн 17.01.22		Харченко К.О.
3		Розробка технічного завдання	1 день	Вт 18.01.22	Вт 18.01.22	2	Харченко К.О.
4		Редагування та затвердження ТЗ	1 день	Ср 19.01.22	Ср 19.01.22	3	Нагорний В.В.
5		**Розробка додатку**	**53 дней**	**Чт 20.01.22**	**Пн 04.04.22**	**1**	
6		Розробка архітектури додатку	4 дней	Чт 20.01.22	Вт 25.01.22		Харченко К.О.
7		Розробка інтерфейсу	5 дней	Ср 26.01.22	Вт 01.02.22	6	Харченко К.О.
8		Розробка модулю графіки	5 дней	Ср 02.02.22	Вт 08.02.22	7	Харченко К.О.
9		Розробка модулю для роботи з файлами і:	10 дней	Ср 09.02.22	Вт 22.02.22	8	Харченко К.О.
10		Розробка модулю імпорту/експорту з фор	5 дней	Ср 23.02.22	Вт 01.03.22	9	Харченко К.О.
11		Розробка алгоритмів ремешу та децимаці	14 дней	Ср 02.03.22	Пн 21.03.22	10	Харченко К.О.
12		Проектування бази даних	3 дней	Вт 22.03.22	Чт 24.03.22	11	Харченко К.О.
13		Розробка модулю для серверної частини	7 дней	Пт 25.03.22	Пн 04.04.22	12	Харченко К.О.
14		**Тестування**	**14 дней**	**Вт 05.04.22**	**Пт 22.04.22**	**5**	
15		Альфа тестування	7 дней	Вт 05.04.22	Ср 13.04.22		Харченко К.О.
16		Бета тестування	7 дней	Чт 14.04.22	Пт 22.04.22	15	Харченко К.О.
17		**Завершення**	**28 дней**	**Пн 25.04.22**	**Ср 01.06.22**	**14**	
18		Розробка документації	27 дней	Пн 25.04.22	Вт 31.05.22		Харченко К.О.
19		Створення презентації проєкту	1 день	Ср 01.06.22	Ср 01.06.22	18	Харченко К.О.

Рисунок Б.3 – Календарний графік проєкту

Управління ризиками проєкту. Ідентифікація ризиків проєкту наведено в таблиці Б.3.

Таблиця Б.3 – Ідентифікація ризиків проєкту

№ ризику	Назва (опис) ризику
1	Використання нових технологій
2	Проблеми з підключенням до Інтернету
3	Втрата даних
4	Збій у роботі техніки розробника
5	Недостатня кваліфікація розробника
6	Неправильне проєктування архітектури додатка
7	Нечітко виділені вимоги
8	Неправильний розподіл часу
9	Зміна пріоритетів
10	Виявлення помилок під час тестування

В таблиці Б.4 визначено ймовірність, вплив і ранг кожного ризику.

Таблиця Б.4 – Ймовірність, вплив і ранг кожного ризику

№	Назва ризику	Ймовірність (0,1-0,9)	Вплив (0,05-0,8)	Ранг
1	Використання нових технологій	0,5	0,4	0,2
2	Проблеми з підключенням до Інтернету	0,3	0,3	0,09
3	Втрата даних	0,3	0,3	0,09
4	Збій у роботі техніки розробника	0,2	0,4	0,08
5	Недостатня кваліфікація розробника	0,3	0,6	0,18

Продовження таблиці Б.4

6	Неправильне проєктування архітектури додатка	0,3	0,6	0,18
7	Нечітко виділені вимоги	0,1	0,5	0,05
8	Неправильний розподіл часу	0,1	0,3	0,03
9	Зміна пріоритетів	0,1	0,5	0,05
10	Виявлення помилок під час тестування	0,6	0,4	0,24

Матриця ймовірності та впливу наведено в таблиці Б.5.

Таблиця Б.5 – Матриця ймовірність, вплив і ранг кожного ризику

Ймовірність	Вплив загрози(ризику)				
	Дуже малий 0,05	Малий 0,1	Середній 0,4	Великий 0,6	Дуже великий 0,8
0,6			R10(0,24)		
0,5			R1(0,2)		
0,4					
0,3			R2(0,09) R3(0,09) R4(0,08)	R5(0,18) R6(0,18)	
0,1			R8(0,03)	R7(0,05) R9(0,05)	

Таблиця Б.6 – Шкала оцінювання за рівнем впливу

№	Назва	Межі	Ризик, які входять (номера)
1	Прийнятні	0,005≤ R ≤0,05	8, 7, 9
2	Виправдані	0,05< R ≤0,14	2, 3, 4
3	Недопустимі	0,14< R ≤0,72	1, 5, 6, 10

Таблиця Б.7 – Таблиця реагування на ризики

ID	Статус ризику	Опис	Ймовірність	Вплив	Ранг ризику	План А	Тип стратегії реагування	План Б
1	Відкритий	Використання нових технологій	Велика	Середній	0,2	На початку проєктування дослідити матеріал	Зменшення	Замінити іншою технологією
2	Відкритий	Проблеми з підключенням до Інтернету	Низька	Середній	0.09	Підключити два провайдери інтернету	Ухилення	Залучити спеціаліста для вирішення проблем
3	Відкритий	Втрата даних	Низька	Середній	0.09	Залучити спеціаліста для вирішення проблем	Зменшення	-
4	Відкритий	Збій у роботі техніки розробника	Середня	Середній	0.08	Замінити техніку на резервну	Зменшення	-
5	Відкритий	Недостатня кваліфікація розробника	Низький	Великий	0.18	Підвищити кваліфікацію працівника	Зменшення	Долучити сторонього спеціаліста
6	Відкритий	Неправильне проєктування архітектури додатка	Низька	Великий	0.18	Приділити більше часу в плані	Ухилення	-
7	Відкритий	Нечітко виділені вимоги	Низький	Великий	0.05	Обговорити не зрозумілі пункти завдання	Зменшення	-
8	Відкритий	Неправильний розподіл часу	Низький	Великий	0.03	Перевизначити терміни виконання завдання	Зменшення	-
9	Відкритий	Зміна пріоритетів	Низький	Середній	0.05	-	Ухилення	-
10	Відкритий	Виявлення помилок під час тестування	Великий	Середній	0.24	Виправити помилки	Зменшення	Проігнорувати помилки якщо вони не значні

ДОДАТОК В

Лістинг коду файлу app.rs:

```
use std::sync::Arc;

use wasm_bindgen::JsCast;

use cgmath::*;
use egui::mutex::Mutex;
use egui_glow::glow;

use crate::camera::OrbitalCamera;
use crate::render::RenderScene;
use crate::mesh::IndexedMesh;
use crate::simplification::Simplify;
use crate::remesh::Remesher;

#[derive(Clone)]
pub struct Settings {
    pub is_cull_face: bool,
    pub is_flat_shading: bool,
    pub is_render_static: bool,
    pub is_render_temp: bool,

    pub light_pos: [f32; 3],
    pub scroll_sensitivity: f32,
    pub min_camera_dist: f32,
```

```
    pub simplification_error: f32,
    pub simplification_agr: f32,
    pub remesh_iterations: u32,

    pub total_num_faces: usize,
    pub total_num_faces_temp: usize,
}

impl Default for Settings {
    fn default() -> Self {
        Self {
            is_cull_face: true,
            is_flat_shading: true,
            is_render_static: true,
            is_render_temp: false,

            light_pos: [0.0, 5.0, 0.0],
            scroll_sensitivity: 0.001,
            min_camera_dist: 0.001,

            simplification_error: 1.0,
            simplification_agr: 7.0,
            remesh_iterations: 1,
            total_num_faces: 0,
            total_num_faces_temp: 0,
        }
    }
}
```

```
#[derive(PartialEq)]
enum PanelState {
    SelectionMenu,
    RemeshMenu,
    SimplificationMenu,
}

impl Default for PanelState {
    fn default() -> Self {
        PanelState::SelectionMenu
    }
}

pub struct WebEditor {
    render_scene_ref: Arc<Mutex<RenderScene>>,
    indexed_meshes: Vec<IndexedMesh>,
    indexed_meshes_temp: Vec<IndexedMesh>,

    settings: Settings,
    camera: OrbitalCamera,

    state: PanelState,

    receiver: Option<oneshot::Receiver<Vec<IndexedMesh>>>,
}

impl WebEditor {
    pub fn new(cc: &eframe::CreationContext<'_>) -> Self {
        let mut app = Self {
```

```
            render_scene_ref: Arc::new(Mutex::new(RenderScene::new(
                cc.gl.as_ref()
            ))),
            indexed_meshes: vec![],
            indexed_meshes_temp: vec![],

            settings: Settings::default(),
            camera: OrbitalCamera::default(),

            state: PanelState::default(),

            receiver: None,
        };

        app.push_indexed_mesh(cc.gl.as_ref(),
IndexedMesh::box3d(Vector3::new(1.0f32, 1.0, 1.0)));
        app
    }

    pub fn reset_all(&mut self, gl: &glow::Context) {
        self.render_scene_ref.lock().reset_buffers(gl);
        self.indexed_meshes.clear();
        self.settings.total_num_faces = 0;

        self.switch_to_selection_menu(gl);
    }
    pub fn switch_to_selection_menu(&mut self, gl: &glow::Context) {
        self.indexed_meshes_temp.clear();
        self.render_scene_ref.lock().reset_temp_buffers(gl);
```

```
        self.settings.total_num_faces_temp = 0;

        self.settings.is_render_static = true;
        self.settings.is_render_temp = false;

        self.state = PanelState::SelectionMenu;
    }
    pub fn apply_temp_mehes(&mut self, gl: &glow::Context) {
        self.indexed_meshes = self.indexed_meshes_temp.clone();
        self.render_scene_ref.lock().reset_static_and_create_static_meshes(gl,
&self.indexed_meshes);
        self.settings.total_num_faces = self.settings.total_num_faces_temp;
        self.settings.total_num_faces_temp = 0;
    }
    pub fn clone_static_to_temp(&mut self, gl: &glow::Context) {
        self.indexed_meshes_temp = self.indexed_meshes.clone();
        self.render_scene_ref.lock().reset_temp_and_create_temp_meshes(gl,
&self.indexed_meshes_temp);
        self.settings.total_num_faces_temp = self.settings.total_num_faces;
    }
    pub fn push_indexed_mesh(&mut self, gl: &glow::Context, mesh: IndexedMesh) {
        self.render_scene_ref.lock().push_static_mesh(gl, &mesh);
        self.indexed_meshes.push(mesh);
        self.settings.total_num_faces += self.indexed_meshes.last().unwrap().indices.len()
/ 3;
    }
    pub fn recalculate_camera_view(&mut self) {
        let mut center_point = Vector3::new(0.0f32, 0.0, 0.0);
        let (mut min, mut max) = (
```

```
            Vector3::new(std::f32::MAX, std::f32::MAX, std::f32::MAX),
            Vector3::new(std::f32::MIN, std::f32::MIN, std::f32::MIN)
        );

        for mesh in self.indexed_meshes.iter() {
            center_point += mesh.calculate_center_point() / self.indexed_meshes.len() as
f32;

            let (min_local, max_local) = mesh.calculate_aabb();
            min.x = min.x.min(min_local.x);
            min.y = min.y.min(min_local.y);
            min.z = min.z.min(min_local.z);

            max.x = max.x.max(max_local.x);
            max.y = max.y.max(max_local.y);
            max.z = max.z.max(max_local.z);
        }

        self.camera.center = center_point;

        let scene_dist = max - min;
        let max_scene_dist_half = scene_dist.magnitude() / 2.0;
        let tan_half = (self.camera.fov / 180.0 * std::f32::consts::PI).tan() / 2.0;
        self.camera.dist = max_scene_dist_half / tan_half;

        self.settings.scroll_sensitivity = max_scene_dist_half * 0.001;
    }
}
```

```
impl eframe::App for WebEditor {
    fn update(&mut self, ctx: &egui::Context, frame: &mut eframe::Frame) {
        egui::TopBottomPanel::top("top_panel").show(ctx, |ui| {
            egui::menu::bar(ui, |ui| {
                egui::widgets::global_dark_light_mode_switch(ui);
                ui.menu_button("File", |ui| {
                    if ui.button("Open").clicked() {

                        let (sender, receiver) = oneshot::channel::<Vec<IndexedMesh>>();
                        self.receiver = Some(receiver);

                        let task = rfd::AsyncFileDialog::new().pick_files();
                        wasm_bindgen_futures::spawn_local(async {
                            let files = task.await;

                            let mut loaded_indexed_meshes = vec![];
                            if let Some(files) = files {
                                for file in files {
                                    let bytes = file.read();

                                    let file_name = file.file_name();
                                    let ext = std::path::Path::new(&file_name)
                                        .extension()
                                        .and_then(std::ffi::OsStr::to_str);

                                    let bytes = std::io::Cursor::new(bytes.await);

                                    if let Some(ext) = ext {
                                        let mesh = Files::read_indexed_mesh(bytes, ext);
```

```
                    if let Ok(mesh) = mesh {
                        if !mesh.is_empty() {
                            loaded_indexed_meshes.push(mesh);
                        }
                    }
                }
            }

            let _err = sender.send(loaded_indexed_meshes);
        });

    }
    ui.menu_button("Save", |ui| {
        if ui.button("stl").clicked() {
            let mut stl_mesh = vec![];
            for mesh in self.indexed_meshes.iter() {
                for face_idxs in mesh.indices.windows(3).step_by(3) {
                    let v0 = mesh.positions[face_idxs[0] as usize];
                    let v1 = mesh.positions[face_idxs[1] as usize];
                    let v2 = mesh.positions[face_idxs[2] as usize];

                    let face_normal = (v1 - v0).cross(v2 - v0);

                    stl_mesh.push(
                        stl_io::Triangle {
                            normal: stl_io::Normal::new([face_normal.x, face_normal.y,
face_normal.z]),
```

```
                vertices:
                [
                    stl_io::Vertex::new([v0.x, v0.y, v0.z]),
                    stl_io::Vertex::new([v1.x, v1.y, v1.z]),
                    stl_io::Vertex::new([v2.x, v2.y, v2.z]),
                ]
            }
        );
    }
}

let mut binary_stl = Vec::<u8>::new();
let write_result = stl_io::write_stl(&mut binary_stl, stl_mesh.iter());
if !write_result.is_ok() {
    panic!("Error when create binary stl!");
}

let is_ok = Files::save_file_binary("file.stl", binary_stl);
if !is_ok {
    panic!("Error when save stl file!");
}
}
if ui.button("ply").clicked() {
use ply_rs::ply::{
    Ply, DefaultElement, Encoding,
    ElementDef, PropertyDef, PropertyType,
    ScalarType, Property, Addable
};
use ply_rs::writer::Writer;
```

```
                let mut binary_ply = Vec::<u8>::new();

                let mut ply = {
                    let mut ply = Ply::<DefaultElement>::new();
                    ply.header.encoding = Encoding::Ascii;
                    ply.header.comments.push("ply export from Web
Editor".to_string());

                    let mut vertex_element = ElementDef::new("vertex".to_string());
                    let v = PropertyDef::new("x".to_string(),
PropertyType::Scalar(ScalarType::Float));
                    vertex_element.properties.add(v);
                    let v = PropertyDef::new("y".to_string(),
PropertyType::Scalar(ScalarType::Float));
                    vertex_element.properties.add(v);
                    let v = PropertyDef::new("z".to_string(),
PropertyType::Scalar(ScalarType::Float));
                    vertex_element.properties.add(v);
                    ply.header.elements.add(vertex_element);

                    let mut face_element = ElementDef::new("face".to_string());
                    let face_type = PropertyType::List(ScalarType::UChar,
ScalarType::Int);
                    let v = PropertyDef::new("vertex_indices".to_string(), face_type);
                    face_element.properties.add(v);
                    ply.header.elements.add(face_element);

                    let mut vertices = Vec::new();
                    for mesh in self.indexed_meshes.iter() {
```

```
            for v in mesh.positions.iter() {

                let mut vertex = DefaultElement::new();
                vertex.insert("x".to_string(), Property::Float(v.x));
                vertex.insert("y".to_string(), Property::Float(v.y));
                vertex.insert("z".to_string(), Property::Float(v.z));

                vertices.push(vertex);
            }
        }
        ply.payload.insert("vertex".to_string(), vertices);

        let mut indices = Vec::new();
        for mesh in self.indexed_meshes.iter() {
            for face_idxs in mesh.indices.windows(3).step_by(3) {

                let mut index = DefaultElement::new();
                index.insert(
                    "vertex_indices".to_string(),
                    Property::ListInt([face_idxs[0] as i32, face_idxs[1] as i32,
face_idxs[2] as i32].into())
                );
                indices.push(index);
            }
        }
        ply.payload.insert("face".to_string(), indices);

        ply.make_consistent().unwrap();
        ply
```

```
                    };
                    let write_result = Writer::new().write_ply(&mut binary_ply, &mut
ply);
                    if !write_result.is_ok() {
                        panic!("Error when create binary ply!");
                    }

                    let is_ok = Files::save_file_binary("file.ply", binary_ply);
                    if !is_ok {
                        panic!("Error when save ply file!");
                    }
                }
            });
            if ui.button("Reset").clicked() {
                self.reset_all(frame.gl());
            }
        });
    });

    Files::check_dropped_files_then_preview_load(ctx, frame.gl(), self);
    if let Some(receiver) = self.receiver.as_ref() {
        match receiver.try_recv() {
            Ok(loaded_indexed_meshes) => {
                self.reset_all(frame.gl());
                for indexed_mesh in loaded_indexed_meshes {
                    self.push_indexed_mesh(frame.gl(), indexed_mesh);
                }
```

```
                self.recalculate_camera_view();
                self.receiver = None;
            }
            Err(oneshot::TryRecvError::Disconnected) => {
                self.receiver = None;
            }
            _ => {}
        }
    }

    egui::SidePanel::left("side_panel").resizable(false).show(ctx, |ui| {
        ui.heading("Side Panel");
        ui.separator();

        match self.state {
            PanelState::SelectionMenu => {

ui.with_layout(egui::Layout::top_down(egui::Align::Center).with_cross_justify(true),
|ui| {
                if ui.button("Remesh").on_hover_text("Remesh operation").clicked() {
                    self.clone_static_to_temp(frame.gl());
                    self.settings.is_render_static = false;
                    self.settings.is_render_temp = true;
                    self.state = PanelState::RemeshMenu;
                }
                if ui.button("Simplification").on_hover_text("Decimation
operation").clicked() {
                    self.clone_static_to_temp(frame.gl());
                    self.settings.is_render_static = false;
```

```
                    self.settings.is_render_temp = true;
                    self.state = PanelState::SimplificationMenu;
                }

                //let input = ui.input().clone();
                //input.ui(ui);
            });
        }
        PanelState::RemeshMenu => {
            let mut iter = self.settings.remesh_iterations;
            ui.add(egui::Slider::new(&mut iter, 1..=5).integer().text("Iterations"));

            if self.settings.remesh_iterations != iter {

                self.settings.total_num_faces_temp = 0;
                for (mesh, new_mesh) in self.indexed_meshes.iter().zip(self.indexed_meshes_temp.iter_mut()) {
                    *new_mesh = mesh.clone();

                    Remesher::split_faces(new_mesh, iter as usize);
                    self.settings.total_num_faces_temp += new_mesh.indices.len() / 3;
                }

                self.settings.remesh_iterations = iter;
                self.render_scene_ref.lock()
                    .reset_temp_and_create_temp_meshes(frame.gl(), &self.indexed_meshes_temp);
            }
```

```
                ui.label(&format!("faces before: {}", self.settings.total_num_faces));
                ui.label(&format!("faces after: {}", self.settings.total_num_faces_temp));

                ui.horizontal(|ui| {
                    if ui.button("Apply").on_hover_text("Apply changes and return to selection menu").clicked() {
                        self.apply_temp_mehes(frame.gl());
                        self.switch_to_selection_menu(frame.gl());
                    }
                    if ui.button("Back").on_hover_text("Reset changes and return to selection menu").clicked() {
                        self.switch_to_selection_menu(frame.gl());
                    }
                });
            }
            PanelState::SimplificationMenu => {
                let mut error = self.settings.simplification_error;
                let mut agr = self.settings.simplification_agr;
                ui.add(egui::Slider::new(&mut error, 0.001..=1.0).text("Error"));
                ui.add(egui::Slider::new(&mut agr, 1.0..=20.0).text("Agresiveness"));

                if (self.settings.simplification_error - error).abs() > std::f32::EPSILON
                    || (self.settings.simplification_agr - agr).abs() > std::f32::EPSILON {

                    self.settings.total_num_faces_temp = 0;
                    for (mesh, new_mesh) in self.indexed_meshes.iter().zip(self.indexed_meshes_temp.iter_mut()) {
                        *new_mesh = mesh.clone();
```

```
                let mut simp = Simplify::from(new_mesh);
                simp.simplify_mesh((error * (new_mesh.indices.len() / 3) as f32) as
usize, agr);
                simp.to(new_mesh);

                self.settings.total_num_faces_temp += new_mesh.indices.len() / 3;
            }

            self.settings.simplification_error = error;
            self.settings.simplification_agr = agr;
            self.render_scene_ref.lock()
                .reset_temp_and_create_temp_meshes(frame.gl(),
&self.indexed_meshes_temp);
        }

        ui.label(&format!("faces before: {}", self.settings.total_num_faces));
        ui.label(&format!("faces after: {}", self.settings.total_num_faces_temp));

        ui.horizontal(|ui| {
            if ui.button("Apply").on_hover_text("Apply changes and return to
selection menu").clicked() {
                self.apply_temp_mehes(frame.gl());
                self.switch_to_selection_menu(frame.gl());
            }
            if ui.button("Back").on_hover_text("Reset changes and return to
selection menu").clicked() {
                self.switch_to_selection_menu(frame.gl());
            }
        });
```

```
            }
        }


ui.with_layout(egui::Layout::bottom_up(egui::Align::Min).with_cross_justify(true),  |ui|
{
            ui.checkbox(&mut self.settings.is_cull_face, "set cull faces");
            ui.checkbox(&mut self.settings.is_flat_shading, "set flat shading");
        });
    });

    egui::CentralPanel::default().show(ctx, |ui| {
        ctx.request_repaint();

        self.camera.set_size(ui.max_rect().width(), ui.max_rect().height());
        self.camera.dist -= ui.input().scroll_delta.y * self.settings.scroll_sensitivity;
        self.camera.dist = self.camera.dist.max(self.settings.min_camera_dist);
        if ui.input().pointer.middle_down() {
            let delta_from_prev_frame = ui.input().pointer.delta();
            let right = self.camera.up.cross(self.camera.dir_from_center).normalize();
            self.camera.up = self.camera.dir_from_center.cross(right).normalize();

            let r_xz = Matrix3::from_axis_angle(self.camera.up, Deg(-
delta_from_prev_frame.x));
            let r_yz = Matrix3::from_axis_angle(right, Deg(-delta_from_prev_frame.y));
            self.camera.dir_from_center = r_yz * r_xz * self.camera.dir_from_center;
        }

        let triangle = self.render_scene_ref.clone();
```

```
            let camera = self.camera.clone();
            let settings = self.settings.clone();

            let callback = egui::PaintCallback {
                rect: ui.max_rect(),
                callback: std::sync::Arc::new(move |_info, render_ctx| {
                    if let Some(painter) = render_ctx.downcast_ref::<egui_glow::Painter>() {
                        triangle.lock().render(painter.gl(), &settings, &camera);
                    } else {
                        eprintln!("Can't do custom painting because we are not using a glow context");
                    }
                }),
            };
            ui.painter().add(callback);
        });
    }

    fn on_exit(&mut self, gl: &glow::Context) {
        self.render_scene_ref.lock().destroy(gl);
    }
}

#[derive(Default)]
struct Files {}

impl Files {
    fn save_file_binary(filename: &str, content: Vec<u8>) -> bool {
        let window = web_sys::window();
```

```
if window.is_none() {
    return false;
}
let window = window.unwrap();

let document = window.document();
if document.is_none() {
    return false;
}
let document = document.unwrap();

let element = document.create_element("a");
if element.is_err() {
    return false;
}
let element = element.unwrap();

let a_element = element.dyn_into::<web_sys::HtmlAnchorElement>();
if a_element.is_err() {
    return false;
}
let a_element = a_element.unwrap();

let body = document.body();
if body.is_none() {
    return false;
}
let body = body.unwrap();
```

```
    let append_child = body.append_child(&a_element);
    if append_child.is_err() {
        return false;
    }

    let uint8arr = js_sys::Uint8Array::new(&unsafe { js_sys::Uint8Array::view(&content) }.into());
    let blob_content = js_sys::Array::new();
    blob_content.push(&uint8arr.buffer());

    let blob = web_sys::Blob::new_with_u8_array_sequence_and_options(
        &blob_content,
        web_sys::BlobPropertyBag::new().type_("application/octet-stream")
    );
    if blob.is_err() {
        return false;
    }
    let blob = blob.unwrap();

    let url = web_sys::Url::create_object_url_with_blob(&blob);
    if url.is_err() {
        return false;
    }
    let url = url.unwrap();

    a_element.set_href(&url);
    a_element.set_download(filename);
    a_element.click();
    a_element.remove();
```

```
    true
}

fn check_dropped_files_then_preview_load(
    ctx: &egui::Context,
    gl: &glow::Context,
    web_editor: &mut WebEditor
) {
    if !ctx.input().raw.dropped_files.is_empty() {
        let dropped_files = ctx.input().raw.dropped_files.clone();

        web_editor.reset_all(gl);

        for dropped_file in dropped_files.iter() {
            if let Some(bytes_ref) = &dropped_file.bytes {
                let file = std::io::Cursor::new(bytes_ref);

                let ext = std::path::Path::new(&dropped_file.name)
                    .extension()
                    .and_then(std::ffi::OsStr::to_str);

                if let Some(ext) = ext {
                    let mesh = Files::read_indexed_mesh(file, ext);

                    if let Ok(mesh) = mesh {
                        if !mesh.is_empty() {
                            web_editor.push_indexed_mesh(gl, mesh);
                        }
```

```
                }
            }
        }
    }

    web_editor.recalculate_camera_view();
  }
  Files::preview_files_being_dropped(ctx);
}

fn read_indexed_mesh<T>(mut file: std::io::Cursor<T>, ext: &str) -> Result<IndexedMesh, std::io::Error>
  where
    T: std::convert::AsRef<[u8]>,
  {
    match ext {
      "stl" | "STL" => {
        let mut stl = stl_io::create_stl_reader(&mut file)?;
        let stl_indexed_mesh = stl.as_indexed_triangles()?;

        let mut mesh = IndexedMesh {
          positions: stl_indexed_mesh.vertices
            .into_iter()
            .map(|vertex| Vector3::new(vertex[0], vertex[1], vertex[2]))
            .collect(),

          normals: vec![],

          indices: stl_indexed_mesh.faces
```

```
                .into_iter()
                .flat_map(|face|
                    [face.vertices[0] as u32, face.vertices[1] as u32, face.vertices[2] as
u32]
                )
                .collect(),
            };
            mesh.recalculate_normals();
            Ok(mesh)
        }
        "ply" | "PLY" => {
            use ply_rs::*;

            struct Vertex {
                v: [f32; 3],
            }
            struct Face {
                vertices: Vec<i32>,
            }

            impl ply::PropertyAccess for Vertex {
                fn new() -> Self {
                    Vertex { v: [0.0, 0.0, 0.0] }
                }
                fn set_property(&mut self, key: String, property: ply::Property) {
                    match (key.as_ref(), property) {
                        ("x", ply::Property::Float(v)) => self.v[0] = v,
                        ("y", ply::Property::Float(v)) => self.v[1] = v,
                        ("z", ply::Property::Float(v)) => self.v[2] = v,
```

```
            (_, _) => {},
        }
    }
}
impl ply::PropertyAccess for Face {
    fn new() -> Self {
        Face { vertices: Vec::new() }
    }
    fn set_property(&mut self, key: String, property: ply::Property) {
        match (key.as_ref(), property) {
            ("vertex_index", ply::Property::ListInt(vec)) => self.vertices = vec,
            ("vertex_indices", ply::Property::ListInt(vec)) => self.vertices = vec,
            (_, _) => {},
        }
    }
}
let vertex_parser = parser::Parser::<Vertex>::new();
let face_parser = parser::Parser::<Face>::new();

let header = vertex_parser.read_header(&mut file)?;

let mut mesh = IndexedMesh::default();
for (_ignore_key, element) in &header.elements {
    match element.name.as_ref() {
        "vertex" => {
            mesh.positions = vertex_parser
                .read_payload_for_element(&mut file, &element, &header)
                .unwrap()
                .into_iter()
```

```
                .map(|vertex| Vector3::new(vertex.v[0], vertex.v[1], vertex.v[2]))
                .collect();
            },
        "face" => {
            let ply_faces = face_parser
                .read_payload_for_element(&mut file, &element, &header)
                .unwrap();

            for face in ply_faces {
                for face_idx in (0..face.vertices.len()).into_iter().step_by(2) {
                    mesh.indices.extend_from_slice(&[
                        face.vertices[face_idx + 0] as u32,
                        face.vertices[(face_idx + 1) % face.vertices.len()] as u32,
                        face.vertices[(face_idx + 2) % face.vertices.len()] as u32
                    ]);
                }
            }
        },
        _ => {},
      }
    }

    mesh.recalculate_normals();
    Ok(mesh)
}
_ => {
    Err(std::io::Error::new(
        std::io::ErrorKind::Other, format!("Not supported format `{}`", ext)
    ))
```

```
        }
    }
}

fn preview_files_being_dropped(ctx: &egui::Context) {
    use egui::*;
    if ctx.input().raw.hovered_files.is_empty() {
        return;
    }

    let mut text = "Dropping files:\n".to_string();
    for dropped_file in &ctx.input().raw.hovered_files {
        if let Some(path) = &dropped_file.path {
            text += &format!("\n{}", path.display());
        } else if !dropped_file.mime.is_empty() {
            text += &format!("\n{}", dropped_file.mime);
        } else {
            text += "\nUnknown file";
        }
    }

    let render =
        ctx.layer_painter(LayerId::new(Order::Foreground, Id::new("dropping_files")));

    let screen_rect = ctx.input().screen_rect();
    render.rect_filled(screen_rect, 0.0, Color32::from_black_alpha(180));
    render.text(
        screen_rect.center(),
        Align2::CENTER_CENTER,
```

```
            text,
            TextStyle::Heading.resolve(&ctx.style()),
            Color32::WHITE,
        );
    }
}
```

Лістинг коду файлу camera.rs:

```
use cgmath::*;

#[derive(Clone)]
pub struct OrbitalCamera {
    render_width: f32,
    render_height: f32,

    pub fov: f32,
    pub near: f32,
    pub far: f32,

    pub up: Vector3<f32>,
    pub center: Vector3<f32>,
    pub dir_from_center: Vector3<f32>,
    pub dist: f32,
}

impl OrbitalCamera {
    pub fn calculate_perspective_matrix(&self) -> Matrix4<f32> {
        perspective(
            Deg(self.fov),
```

```
            self.render_width / self.render_height,
            self.near, self.far
        )
    }

    pub fn calculate_view_matrix(&self) -> Matrix4<f32> {
        let eye_pos = self.calculate_pos();
        Matrix4::look_to_rh(Point3::from_vec(eye_pos), -self.dir_from_center, self.up)
    }

    pub fn set_size(&mut self, width: f32, height: f32) {
        self.render_width = width;
        self.render_height = height;
    }

    pub fn calculate_pos(&self) -> Vector3<f32> {
        self.center + self.dir_from_center * self.dist
    }
}

impl Default for OrbitalCamera {
    fn default() -> Self {
        Self {
            render_width: 0.0f32,
            render_height: 0.0f32,

            fov: 60.0f32,
            near: 1.0f32,
            far: 1_000.0f32,
```

```
            up: Vector3::new(0.0f32, 1.0, 0.0),
            center: Vector3::new(0.0f32, 0.0, 0.0),
            dir_from_center: Vector3::new(0.0f32, 0.0, 1.0),
            dist: 5.0f32,
        }
    }
}
```

Лістинг коду файлу lib.rs:

```
mod simplification;
mod remesh;
mod camera;
mod render;
mod mesh;
mod app;
pub use app::WebEditor;

#[cfg(target_arch = "wasm32")]
use eframe::wasm_bindgen::{self, prelude::*};

// wasm entry
#[cfg(target_arch = "wasm32")]
#[wasm_bindgen]
pub fn start(canvas_id: &str) -> Result<(), eframe::wasm_bindgen::JsValue> {

    console_error_panic_hook::set_once();
    tracing_wasm::set_as_global_default();
```

```
    // ui stuff
    eframe::start_web(canvas_id, Box::new(|cc| Box::new(WebEditor::new(cc))))?;

    Ok(())
}
```

Лістинг коду файлу mesh.rs:

```
use cgmath::*;

#[derive(Default, Clone)]
pub struct IndexedMesh {
    pub positions: Vec<Vector3<f32>>,
    pub normals: Vec<Vector3<f32>>,
    pub indices: Vec<u32>,
}

impl IndexedMesh {
    pub fn is_empty(&self) -> bool {
        self.positions.is_empty() || self.normals.is_empty() || self.indices.is_empty()
    }
    pub fn clear(&mut self) {
        self.positions.clear();
        self.normals.clear();
        self.indices.clear();
    }

    pub fn recalculate_normals(&mut self) {
        self.normals.resize(self.positions.len(), Vector3::new(0.0, 0.0, 0.0));
```

```
    for face_idxs in self.indices.windows(3).step_by(3) {
        let v0 = self.positions[face_idxs[0] as usize];
        let v1 = self.positions[face_idxs[1] as usize];
        let v2 = self.positions[face_idxs[2] as usize];

        let face_normal = (v1 - v0).cross(v2 - v0);
        self.normals[face_idxs[0] as usize] += face_normal;
        self.normals[face_idxs[1] as usize] += face_normal;
        self.normals[face_idxs[2] as usize] += face_normal;
    }
    for normal in self.normals.iter_mut() {
        *normal = normal.normalize();
    }
}

pub fn calculate_center_point(&self) -> Vector3<f32> {
    let mut center_point = Vector3::new(0.0f32, 0.0, 0.0);
    for v in self.positions.iter() {
        center_point += *v / self.positions.len() as f32;
    }

    center_point
}

pub fn calculate_aabb(&self) -> (Vector3<f32>, Vector3<f32>) {
    let (mut min, mut max) = (
        Vector3::new(std::f32::MAX, std::f32::MAX, std::f32::MAX),
        Vector3::new(std::f32::MIN, std::f32::MIN, std::f32::MIN)
    );
```

```
    for v in self.positions.iter() {
        min.x = min.x.min(v.x);
        min.y = min.y.min(v.y);
        min.z = min.z.min(v.z);

        max.x = max.x.max(v.x);
        max.y = max.y.max(v.y);
        max.z = max.z.max(v.z);
    }

    (min, max)
}

pub fn box3d(len: Vector3<f32>) -> IndexedMesh {

    let mut box3d = IndexedMesh::default();
    let half_len = len / 2.0;
    let center = Vector3::new(0.0f32, 0.0, 0.0);

    box3d.positions.push(center + Vector3::new(-half_len.x, -half_len.y, -half_len.z));
    box3d.positions.push(center + Vector3::new(half_len.x, -half_len.y, -half_len.z));
    box3d.positions.push(center + Vector3::new(-half_len.x, half_len.y, -half_len.z));
    box3d.positions.push(center + Vector3::new(half_len.x, half_len.y, -half_len.z));

    box3d.positions.push(center + Vector3::new(-half_len.x, -half_len.y, half_len.z));
    box3d.positions.push(center + Vector3::new(half_len.x, -half_len.y, half_len.z));
    box3d.positions.push(center + Vector3::new(-half_len.x, half_len.y, half_len.z));
    box3d.positions.push(center + Vector3::new(half_len.x, half_len.y, half_len.z));
```

```
        box3d.indices.extend_from_slice(&[1, 0, 2]);
        box3d.indices.extend_from_slice(&[2, 3, 1]);

        box3d.indices.extend_from_slice(&[5, 1, 7]);
        box3d.indices.extend_from_slice(&[3, 7, 1]);

        box3d.indices.extend_from_slice(&[4, 5, 6]);
        box3d.indices.extend_from_slice(&[7, 6, 5]);

        box3d.indices.extend_from_slice(&[0, 4, 2]);
        box3d.indices.extend_from_slice(&[6, 2, 4]);

        box3d.indices.extend_from_slice(&[3, 2, 7]);
        box3d.indices.extend_from_slice(&[6, 7, 2]);

        box3d.indices.extend_from_slice(&[1, 4, 0]);
        box3d.indices.extend_from_slice(&[4, 1, 5]);

        box3d.recalculate_normals();

        box3d
    }
}
```

Лістинг коду файлу remesh.rs:

```
use crate::mesh::IndexedMesh;

// just split triangles
pub struct Remesher {}
```

```
impl Remesher {
    pub fn split_faces(mesh: &mut IndexedMesh, iteration: usize) {
        let mut new_indices = Vec::with_capacity(mesh.indices.len());
        for _ in 0..iteration {
            for face_idxs in mesh.indices.windows(3).step_by(3) {
                let v0 = mesh.positions[face_idxs[0] as usize];
                let v1 = mesh.positions[face_idxs[1] as usize];
                let v2 = mesh.positions[face_idxs[2] as usize];

                let centroid = (v0 + v1 + v2) / 3.0;
                let new_idx = mesh.positions.len() as u32;
                mesh.positions.push(centroid);

                new_indices.extend([face_idxs[0], face_idxs[1], new_idx]);
                new_indices.extend([face_idxs[1], face_idxs[2], new_idx]);
                new_indices.extend([face_idxs[2], face_idxs[0], new_idx]);
            }

            std::mem::swap(&mut mesh.indices, &mut new_indices);
        }
    }
}
```

Лістинг коду файлу render.rs:

```
use cgmath::*;
use egui_glow::glow;

use crate::app::Settings;
use crate::camera::OrbitalCamera;
```

```
use crate::mesh::IndexedMesh;

enum RenderBuffersUsage {
    Static,
    Dynamic,
}

struct IndexedMeshRenderBuffers {
    vertices_cnt: u32,
    triangles_cnt: u32,

    positions_vbo: glow::Buffer,
    normals_vbo: glow::Buffer,
    indices_ebo: glow::Buffer,

    vao: glow::VertexArray,
}

impl IndexedMeshRenderBuffers {
    fn from_mesh(
        gl: &glow::Context,
        mesh: &IndexedMesh,
        usage: RenderBuffersUsage
    ) -> Result<IndexedMeshRenderBuffers, String> {
        use glow::HasContext as _;

        let usage_gl = match usage {
            RenderBuffersUsage::Static => glow::STATIC_DRAW,
            RenderBuffersUsage::Dynamic => glow::DYNAMIC_DRAW,
```

```
    };

    unsafe {
        let vao = gl.create_vertex_array()?;
        gl.bind_vertex_array(Some(vao));

        let positions_vbo = gl.create_buffer()?;

        gl.bind_buffer(glow::ARRAY_BUFFER, Some(positions_vbo));
        let positions_u8: &[u8] = core::slice::from_raw_parts(
            mesh.positions.as_ptr() as *const u8,
            mesh.positions.len() * 3 * core::mem::size_of::<f32>(),
        );
        gl.buffer_data_u8_slice(glow::ARRAY_BUFFER, positions_u8, usage_gl);
        gl.enable_vertex_attrib_array(0);
        gl.vertex_attrib_pointer_f32(0, 3, glow::FLOAT, false, 3 *
core::mem::size_of::<f32>() as i32, 0);

        let normals_vbo = gl.create_buffer()?;

        gl.bind_buffer(glow::ARRAY_BUFFER, Some(normals_vbo));
        let normals_u8: &[u8] = core::slice::from_raw_parts(
            mesh.normals.as_ptr() as *const u8,
            mesh.normals.len() * 3 * core::mem::size_of::<f32>(),
        );
        gl.buffer_data_u8_slice(glow::ARRAY_BUFFER, normals_u8, usage_gl);
        gl.enable_vertex_attrib_array(1);
        gl.vertex_attrib_pointer_f32(1, 3, glow::FLOAT, false, 3 *
core::mem::size_of::<f32>() as i32, 0);
```

```
        let indices_ebo = gl.create_buffer()?;
        gl.bind_buffer(glow::ELEMENT_ARRAY_BUFFER, Some(indices_ebo));
        let indices_u8: &[u8] = core::slice::from_raw_parts(
            mesh.indices.as_ptr() as *const u8,
            mesh.indices.len() * core::mem::size_of::<u32>(),
        );
        gl.buffer_data_u8_slice(glow::ELEMENT_ARRAY_BUFFER, indices_u8,
usage_gl);

        gl.bind_vertex_array(None);

        Ok(IndexedMeshRenderBuffers {
            vertices_cnt: mesh.positions.len() as u32,
            triangles_cnt: (mesh.indices.len() / 3) as u32,

            positions_vbo,
            normals_vbo,
            indices_ebo,
            vao,
        })
    }
}

pub fn destroy(&self, gl: &glow::Context) {
    use glow::HasContext as _;
    unsafe {
        gl.delete_vertex_array(self.vao);
        gl.delete_buffer(self.positions_vbo);
```

```
            gl.delete_buffer(self.normals_vbo);
            gl.delete_buffer(self.indices_ebo);
        }
    }
}

pub struct RenderScene {
    program_default_indexed_mesh: glow::Program,
    indexed_render_buffers: Vec<IndexedMeshRenderBuffers>,
    indexed_render_buffers_temp: Vec<IndexedMeshRenderBuffers>,
}

// for glow
#[allow(unsafe_code)]
impl RenderScene {
    pub fn new(gl: &glow::Context) -> Self {
        use glow::HasContext as _;

        let shader_version = if cfg!(target_arch = "wasm32") {
            "#version 300 es"
        } else {
            "#version 410"
        };

        unsafe {
            let program = gl.create_program().expect("Cannot create program");

            let (vertex_shader_source, fragment_shader_source) = (
                r#"
```

```
            layout (location = 0) in vec3 in_position;
            layout (location = 1) in vec3 in_normal;

            out vec3 vs_out_pos;
            out vec3 vs_out_unproject_pos;
            out vec3 vs_out_normal;

            uniform mat4 u_model;
            uniform mat4 u_view;
            uniform mat4 u_proj;

            void main() {
                vs_out_pos = vec3(u_view * u_model * vec4(in_position.xyz, 1.0));
                vs_out_normal = mat3(transpose(inverse(u_view * u_model))) * in_normal;
                gl_Position = u_proj * u_view * u_model * vec4(in_position.xyz, 1.0);
            }
        "#,
        r#"
            precision mediump float;

            in vec3 vs_out_pos;
            in vec3 vs_out_normal;

            out vec4 out_color;

            uniform vec3 u_light_pos;
            uniform vec3 u_camera_pos;
            uniform vec4 u_color;
```

```
uniform int u_is_flat_shading;

void main() {
    vec3 normal;
    if (u_is_flat_shading == 0) {
        normal = normalize(vs_out_normal);
    } else {
        normal = normalize(cross(dFdx(vs_out_pos), dFdy(vs_out_pos)));
    }

    vec3 light_dir = normalize(u_light_pos - vs_out_pos);
    vec3 light_color = vec3(1.0, 1.0, 1.0);

    vec3 view_dir = normalize(u_camera_pos - vs_out_pos);
    vec3 reflect_dir = reflect(-light_dir, normal);

    float ambient_strength = 0.1;
    vec3 ambient = ambient_strength * light_color;

    float diff = max(dot(normal, light_dir), 0.0);
    vec3 diffuse = diff * light_color;

    float specular_strength = 0.5;
    float spec = pow(max(dot(view_dir, reflect_dir), 0.0), 32.0);
    vec3 specular = specular_strength * spec * light_color;

    vec3 color = (ambient + diffuse + specular) * u_color.rgb;
```

```
                out_color = vec4(color, u_color.a);
            }
        "#,

    );

    let shader_sources = [
        (glow::VERTEX_SHADER, vertex_shader_source),
        (glow::FRAGMENT_SHADER, fragment_shader_source),
    ];

    let shaders: Vec<_> = shader_sources
        .iter()
        .map(|(shader_type, shader_source)| {
            let shader = gl
                .create_shader(*shader_type)
                .expect("Cannot create shader");
            gl.shader_source(shader, &format!("{}\n{}", shader_version, shader_source));
            gl.compile_shader(shader);
            if !gl.get_shader_compile_status(shader) {
                panic!("{}", gl.get_shader_info_log(shader));
            }
            gl.attach_shader(program, shader);
            shader
        })
        .collect();

    gl.link_program(program);
```

```
        if !gl.get_program_link_status(program) {
            panic!("{}", gl.get_program_info_log(program));
        }

        for shader in shaders {
            gl.detach_shader(program, shader);
            gl.delete_shader(shader);
        }

        Self {
            program_default_indexed_mesh: program,
            indexed_render_buffers: vec![],
            indexed_render_buffers_temp: vec![],
        }
    }
}

pub fn destroy(&self, gl: &glow::Context) {
    use glow::HasContext as _;
    unsafe {
        gl.delete_program(self.program_default_indexed_mesh);
        for buffer in self.indexed_render_buffers.iter() {
            buffer.destroy(gl);
        }
        for buffer in self.indexed_render_buffers_temp.iter() {
            buffer.destroy(gl);
        }
    }
}
```

```
    pub fn reset_buffers(&mut self, gl: &glow::Context) {
        for buffer in self.indexed_render_buffers.iter() {
            buffer.destroy(gl);
        }
        self.indexed_render_buffers.clear();

        self.reset_temp_buffers(gl);
    }

    pub fn push_static_mesh(&mut self, gl: &glow::Context, mesh: &IndexedMesh) {
        self.indexed_render_buffers
            .push(IndexedMeshRenderBuffers::from_mesh(gl, &mesh,
RenderBuffersUsage::Static).unwrap());
    }

    pub fn reset_static_and_create_static_meshes(&mut self, gl: &glow::Context,
meshes: &[IndexedMesh]) {
        self.reset_buffers(gl);

        for mesh in meshes.iter() {
            self.push_static_mesh(gl, mesh);
        }
    }

    pub fn reset_temp_and_create_temp_meshes(&mut self, gl: &glow::Context,
meshes: &[IndexedMesh]) {
        self.reset_temp_buffers(gl);
```

```
        for mesh in meshes.iter() {
            self.indexed_render_buffers_temp
                .push(IndexedMeshRenderBuffers::from_mesh(gl, &mesh,
RenderBuffersUsage::Dynamic).unwrap());
        }
    }

    pub fn reset_temp_buffers(&mut self, gl: &glow::Context) {
        if self.indexed_render_buffers_temp.is_empty() { return; }

        for buffer in self.indexed_render_buffers_temp.iter() {
            buffer.destroy(gl);
        }
        self.indexed_render_buffers_temp.clear();
    }

    pub fn render(&self, gl: &glow::Context, settings: &Settings, camera:
&OrbitalCamera) {
        use glow::HasContext as _;

        let proj = camera.calculate_perspective_matrix();
        let view = camera.calculate_view_matrix();
        let model = Matrix4::identity();

        unsafe {
            gl.use_program(Some(self.program_default_indexed_mesh));

            gl.uniform_matrix_4_f32_slice(
```

```
            gl.get_uniform_location(self.program_default_indexed_mesh,
"u_model").as_ref(),
            false,
            std::slice::from_raw_parts(model.as_ptr(), 16)
        );
        gl.uniform_matrix_4_f32_slice(
            gl.get_uniform_location(self.program_default_indexed_mesh,
"u_view").as_ref(),
            false,
            std::slice::from_raw_parts(view.as_ptr(), 16)
        );
        gl.uniform_matrix_4_f32_slice(
            gl.get_uniform_location(self.program_default_indexed_mesh,
"u_proj").as_ref(),
            false,
            std::slice::from_raw_parts(proj.as_ptr(), 16)
        );
        gl.uniform_3_f32_slice(
            gl.get_uniform_location(self.program_default_indexed_mesh,
"u_light_pos").as_ref(),
            &settings.light_pos
        );

        let camera_pos = camera.calculate_pos();
        gl.uniform_3_f32(
            gl.get_uniform_location(self.program_default_indexed_mesh,
"u_camera_pos").as_ref(),
            camera_pos.x, camera_pos.y, camera_pos.z
        );
```

```
        let is_flat_shading_i32 = if settings.is_flat_shading { 1 } else { 0 };
        gl.uniform_1_i32(
            gl.get_uniform_location(self.program_default_indexed_mesh,
"u_is_flat_shading").as_ref(),
            is_flat_shading_i32
        );

        gl.enable(glow::DEPTH_TEST);
        gl.clear(glow::DEPTH_BUFFER_BIT);

        if settings.is_cull_face {
            gl.enable(glow::CULL_FACE);
            gl.cull_face(glow::BACK);
        }

        if settings.is_render_static {
            const MESH_COLOR: [f32; 4] = [0.8, 0.8, 0.8, 1.0];

            for buffer in self.indexed_render_buffers.iter() {
                gl.uniform_4_f32_slice(
                    gl.get_uniform_location(self.program_default_indexed_mesh,
"u_color").as_ref(),
                    &MESH_COLOR
                );

                gl.bind_vertex_array(Some(buffer.vao));
                gl.draw_elements(glow::TRIANGLES, buffer.triangles_cnt as i32 * 3,
glow::UNSIGNED_INT, 0);
```

```
            }

            if !self.indexed_render_buffers.is_empty() {
                gl.bind_vertex_array(None);
            }
        }

        if settings.is_render_temp {
            const MESH_COLOR: [f32; 4] = [0.4, 0.4, 0.4, 1.0];

            for buffer in self.indexed_render_buffers_temp.iter() {
                gl.uniform_4_f32_slice(
                    gl.get_uniform_location(self.program_default_indexed_mesh,
"u_color").as_ref(),
                    &MESH_COLOR
                );

                gl.bind_vertex_array(Some(buffer.vao));
                gl.draw_elements(glow::TRIANGLES, buffer.triangles_cnt as i32 * 3,
glow::UNSIGNED_INT, 0);
            }

            if !self.indexed_render_buffers_temp.is_empty() {
                gl.bind_vertex_array(None);
            }
        }
    }
}
```

Лістинг коду файлу render.rs:

```
// Quadric Mesh Simplification

use std::ops::{Index, IndexMut, Add, AddAssign};

use cgmath::*;

use crate::mesh::IndexedMesh;

#[derive(Default, Clone, Copy)]
struct SymetricMatrix {
    m: [f32; 10],
}

impl SymetricMatrix {
    fn new(c: f32) -> Self {
        let mut m = Self::default();
        for e in &mut m.m {
            *e = c;
        }

        m
    }

    fn from_symetric(
        m11: f32, m12: f32, m13: f32, m14: f32,
        m22: f32, m23: f32, m24: f32,
        m33: f32, m34: f32,
```

```
    m44: f32
) -> Self {
    let mut m = Self::default();

    m.m[0] = m11;  m.m[1] = m12;  m.m[2] = m13;  m.m[3] = m14;
    m.m[4] = m22;  m.m[5] = m23;  m.m[6] = m24;
    m.m[7] = m33;  m.m[8] = m34;
    m.m[9] = m44;

    m
}

fn from_plane(a: f32, b: f32, c: f32, d: f32) -> Self {
    let mut m = Self::default();

    m.m[0] = a * a; m.m[1] = a * b; m.m[2] = a * c;  m.m[3] = a * d;
    m.m[4] = b * b; m.m[5] = b * c; m.m[6] = b * d;
    m.m[7] = c * c; m.m[8] = c * d;
    m.m[9] = d * d;

    m
}

fn det(
    &mut self,
    a11: usize, a12: usize, a13: usize,
    a21: usize, a22: usize, a23: usize,
    a31: usize, a32: usize, a33: usize
) -> f32 {
```

```
        self.m[a11] * self.m[a22] * self.m[a33] + self.m[a13] * self.m[a21] * self.m[a32]
+ self.m[a12] * self.m[a23] * self.m[a31]
            - self.m[a13] * self.m[a22] * self.m[a31] - self.m[a11] * self.m[a23] *
self.m[a32]- self.m[a12] * self.m[a21] * self.m[a33]
    }
}

impl Index<usize> for SymetricMatrix {
    type Output = f32;
    fn index<'a>(&'a self, i: usize) -> &'a f32 {
        &self.m[i]
    }
}

impl IndexMut<usize> for SymetricMatrix {
    fn index_mut<'a>(&'a mut self, i: usize) -> &'a mut f32 {
        &mut self.m[i]
    }
}

impl Add<SymetricMatrix> for SymetricMatrix {
    type Output = SymetricMatrix;

    fn add(self, rhs: SymetricMatrix) -> SymetricMatrix {
        SymetricMatrix::from_symetric(
            self.m[0] + rhs.m[0], self.m[1] + rhs.m[1], self.m[2] + rhs.m[2], self.m[3] +
rhs.m[3],
            self.m[4] + rhs.m[4], self.m[5] + rhs.m[5], self.m[6] + rhs.m[6],
            self.m[7] + rhs.m[7], self.m[8] + rhs.m[8],
```

```
            self.m[9] + rhs.m[9]
        )
    }
}

impl AddAssign<SymetricMatrix> for SymetricMatrix {

    fn add_assign(&mut self, rhs: SymetricMatrix) {
        self.m[0] += rhs.m[0]; self.m[1] += rhs.m[1]; self.m[2] += rhs.m[2]; self.m[3] += rhs.m[3];
        self.m[4] += rhs.m[4]; self.m[5] += rhs.m[5]; self.m[6] += rhs.m[6];
        self.m[7] += rhs.m[7]; self.m[8] += rhs.m[8];
        self.m[9] += rhs.m[9];
    }
}

#[derive(Clone)]
struct Triangle {
    v: [u32; 3],
    err: [f32; 4],
    deleted: i32,
    dirty: i32,
    n: Vector3<f32>,
}
#[derive(Clone)]
struct Vertex {
    p: Vector3<f32>,
    tstart: i32,
    tcount: i32,
```

```
    q: SymetricMatrix,
    border: i32,
}
#[derive(Clone)]
struct Ref {
    tid: i32,
    tvertex: i32,
}

pub struct Simplify {
    triangles: Vec<Triangle>,
    vertices: Vec<Vertex>,
    refs: Vec<Ref>,
}

impl Simplify {
    pub fn from(mesh: &IndexedMesh) -> Self {
        let mut simp = Simplify {
            triangles: vec![],
            vertices: vec![],
            refs: vec![],
        };

        for p in mesh.positions.iter() {
            let v = Vertex {
                p: *p,
                tstart: 0,
                tcount: 0,
                q: SymetricMatrix::new(0.0),
```

```
            border: 0
        };
        simp.vertices.push(v);
    }
    for face_idxs in mesh.indices.windows(3).step_by(3) {
        let t = Triangle {
            v: [face_idxs[0], face_idxs[1], face_idxs[2]],
            err: [0.0; 4],
            deleted: 0,
            dirty: 0,
            n: Vector3::new(0.0, 0.0, 0.0),
        };
        simp.triangles.push(t);
    }

    simp
}

pub fn to(&self, mesh: &mut IndexedMesh) {
    mesh.clear();

    for v in &self.vertices {
        mesh.positions.push(v.p);
    }

    for t in &self.triangles {
        if t.deleted != 0 { continue; }
        mesh.indices.extend(t.v);
    }
```

```
        mesh.recalculate_normals();
    }

    fn vertex_error(q: &SymetricMatrix, v: &Vector3<f32>) -> f32 {
        q[0] * v.x * v.x + 2.0 * q[1] * v.x * v.y + 2.0 * q[2] * v.x * v.z + 2.0 * q[3] * v.x +
q[4] * v.y * v.y
            + 2.0 * q[5] * v.y * v.z + 2.0 * q[6] * v.y + q[7] * v.z * v.z + 2.0 * q[8] * v.z +
q[9]
    }

    fn calculate_error(&self, id_v1: u32, id_v2: u32, p_result: &mut Vector3<f32>) ->
f32 {

        let mut q = self.vertices[id_v1 as usize].q + self.vertices[id_v2 as usize].q;
        let border = self.vertices[id_v1 as usize].border & self.vertices[id_v2 as
usize].border;
        let det = q.det(0, 1, 2, 1, 4, 5, 2, 5, 7);
        let error;

        if det != 0.0 && border == 0 {
            p_result.x = -1.0 / det * q.det(1, 2, 3, 4, 5, 6, 5, 7, 8);
            p_result.y =  1.0 / det * q.det(0, 2, 3, 1, 5, 6, 2, 7, 8);
            p_result.z = -1.0 / det * q.det(0, 1, 3, 1, 4, 6, 2, 5, 8);
            error = Simplify::vertex_error(&q, &p_result);
        }
        else {
            let p1 = self.vertices[id_v1 as usize].p;
            let p2 = self.vertices[id_v2 as usize].p;
            let p3 = (p1 + p2) / 2.0;
```

```
        let error1 = Simplify::vertex_error(&q, &p1);
        let error2 = Simplify::vertex_error(&q, &p2);
        let error3 = Simplify::vertex_error(&q, &p3);

        error = error1.min(error2.min(error3));
        if error1 == error { *p_result = p1; }
        if error2 == error { *p_result = p2; }
        if error3 == error { *p_result = p3; }
    }

    error
      }

fn clean_mesh(&mut self) {
    let mut dst = 0usize;
    for v in &mut self.vertices {
        v.tcount = 0;
    }
    for i in 0..self.triangles.len() {
        if self.triangles[i].deleted == 0 {
            self.triangles[dst] = self.triangles[i].clone();
            dst += 1;

            let t = &self.triangles[i];
            for j in 0..3 {
                self.vertices[t.v[j] as usize].tcount = 1;
            }
        }
```

```
}
self.triangles.resize(dst,
    Triangle {
        v: [0; 3],
        err: [0.0; 4],
        deleted: 0,
        dirty: 0,
        n: Vector3::new(0.0, 0.0, 0.0),
    }
);

dst = 0;
for i in 0..self.vertices.len() {
    if self.vertices[i].tcount != 0 {
        self.vertices[i].tstart = dst as i32;
        self.vertices[dst].p = self.vertices[i].p;
        dst += 1;
    }
}
for t in &mut self.triangles {
    for j in 0..3 {
        t.v[j] = self.vertices[t.v[j] as usize].tstart as u32;
    }
}
self.vertices.resize(dst,
    Vertex {
        p: Vector3::new(0.0, 0.0, 0.0),
        tstart: 0,
        tcount: 0,
```

```
            q: SymetricMatrix::new(0.0),
            border: 0,
        }
    );
}

fn update_mesh(&mut self, iteration: usize) {
    if iteration > 0 {
        let mut dst = 0usize;

        for i in 0..self.triangles.len() {
            if self.triangles[i].deleted == 0 {
                self.triangles[dst] = self.triangles[i].clone();
                dst += 1;
            }

        }
        self.triangles.resize(dst,
            Triangle {
                v: [0; 3],
                err: [0.0; 4],
                deleted: 0,
                dirty: 0,
                n: Vector3::new(0.0, 0.0, 0.0),
            }
        );
    }

    for v in &mut self.vertices {
```

```
    v.tstart = 0;
    v.tcount = 0;
}
for t in &mut self.triangles {
    for t_v in t.v {
        self.vertices[t_v as usize].tcount += 1;
    }
}

let mut tstart = 0;
for v in &mut self.vertices {
    v.tstart = tstart;
    tstart += v.tcount;
    v.tcount = 0;
}

self.refs.resize(self.triangles.len() * 3, Ref { tid: 0, tvertex: 0 });
for (i, t) in self.triangles.iter().enumerate() {
    for j in 0..3 {
        let v = &mut self.vertices[t.v[j] as usize];
        self.refs[(v.tstart + v.tcount) as usize].tid = i as i32;
        self.refs[(v.tstart + v.tcount) as usize].tvertex = j as i32;
        v.tcount += 1;
    }
}

if iteration == 0 {

    let mut vcount = vec![];
```

```
let mut vids = vec![];

for v in &mut self.vertices {
    v.border = 0;
}

for i in 0..self.vertices.len() {
    vcount.clear();
    vids.clear();
    for j in 0..self.vertices[i].tcount {

        let k = self.refs[(self.vertices[i].tstart + j) as usize].tid;
        let t = &self.triangles[k as usize];
        for k in 0..3 {
            let mut ofs = 0;
            let id = t.v[k];
            while ofs < vcount.len() {
                if vids[ofs] == id { break; }
                ofs += 1;
            }
            if ofs == vcount.len() {
                vcount.push(1);
                vids.push(id);
            } else {
                vcount[ofs] += 1;
            }
        }
    }
    for j in 0..vcount.len() {
```

```
            if vcount[j] == 1 {
                self.vertices[vids[j] as usize].border = 1;
            }
        }
    }

    for v in &mut self.vertices {
        v.q = SymetricMatrix::new(0.0);
    }

    for t in &mut self.triangles {
        let n: Vector3<f32>;
        let mut p = [Vector3::new(0.0f32, 0.0, 0.0); 3];

        for j in 0..3 {
            p[j] = self.vertices[t.v[j] as usize].p;
        }

        n = (p[1] - p[0]).cross(p[2] - p[0]).normalize();

        t.n = n;
        for j in 0..3 {
            self.vertices[t.v[j] as usize].q =
                self.vertices[t.v[j] as usize].q + SymetricMatrix::from_plane(n.x, n.y,
n.z, -n.dot(p[0]));
        }
    }
    for i in 0..self.triangles.len() {
        let mut p = Vector3::new(0.0f32, 0.0, 0.0);
```

```
            for j in 0..3 {
                self.triangles[j].err[j] =
                    self.calculate_error(self.triangles[i].v[j], self.triangles[i].v[(j + 1) % 3],
&mut p);
            }
            self.triangles[i].err[3] = self.triangles[i].err[0]
                .min(self.triangles[i].err[1].min(self.triangles[i].err[2]));
        }
    }
}

fn update_triangles(&mut self, i0: u32, v_idx: usize, deleted: &Vec<i32>,
deleted_triangles: &mut usize) {
    let mut p = Vector3::new(0.0f32, 0.0, 0.0);
    for k in 0..self.vertices[v_idx].tcount {
        let r = &self.refs[(self.vertices[v_idx].tstart + k) as usize];

        if self.triangles[r.tid as usize].deleted != 0 { continue; }
        if deleted[k as usize] != 0 {
            self.triangles[r.tid as usize].deleted = 1;
            *deleted_triangles += 1;
            continue;
        }

        self.triangles[r.tid as usize].v[r.tvertex as usize] = i0;
        self.triangles[r.tid as usize].dirty = 1;

        self.triangles[r.tid as usize].err[0] =
```

```
            self.calculate_error(self.triangles[r.tid as usize].v[0], self.triangles[r.tid as usize].v[1], &mut p);
        self.triangles[r.tid as usize].err[1] =
            self.calculate_error(self.triangles[r.tid as usize].v[1], self.triangles[r.tid as usize].v[2], &mut p);
        self.triangles[r.tid as usize].err[2] =
            self.calculate_error(self.triangles[r.tid as usize].v[2], self.triangles[r.tid as usize].v[0], &mut p);

        self.triangles[r.tid as usize].err[3] = self.triangles[r.tid as usize].err[0]
            .min(self.triangles[r.tid as usize].err[1].min(self.triangles[r.tid as usize].err[2]));

        self.refs.push(self.refs[(self.vertices[v_idx].tstart + k) as usize].clone());
    }
  }

  fn flipped(&mut self, p: &Vector3<f32>, i1: u32, v_idx: usize, deleted: &mut Vec<i32>) -> bool {
    for k in 0..self.vertices[v_idx].tcount {
        let t = &self.triangles[self.refs[(self.vertices[v_idx].tstart + k) as usize].tid as usize];
        if t.deleted != 0 { continue; }

        let s = self.refs[(self.vertices[v_idx].tstart + k) as usize].tvertex;
        let id1 = t.v[((s + 1) % 3) as usize];
        let id2 = t.v[((s + 2) % 3) as usize];

        if id1 == i1 || id2 == i1 {
```

```
            deleted[k as usize] = 1;
            continue;
        }

        let d1 = (self.vertices[id1 as usize].p - p).normalize();
        let d2 = (self.vertices[id2 as usize].p - p).normalize();

        if d1.dot(d2).abs() > 0.999 {
            return true;
        }

        let n = d1.cross(d2).normalize();
        deleted[k as usize] = 0;
        if n.dot(t.n) < 0.2 {
            return true;
        }
    }

    false
}

pub fn simplify_mesh(&mut self, target_count: usize, agr: f32) {
    for t in &mut self.triangles {
        t.deleted = 0;
    }

    let mut deleted_triangles = 0;
    let mut deleted0 = vec![];
    let mut deleted1 = vec![];
```

```
let triangle_count = self.triangles.len();

for iteration in 0..100 {
    if triangle_count - deleted_triangles <= target_count { break; }

    if iteration % 5 == 0 {
        self.update_mesh(iteration);
    }

    for t in &mut self.triangles {
        t.dirty = 0;
    }

    // error between new and old mesh
    let threshold = 0.000000001 * ((iteration + 3) as f32).powf(agr);

    for i in 0..self.triangles.len() {
        if self.triangles[i].err[3] > threshold { continue; }
        if self.triangles[i].deleted != 0 { continue; }
        if self.triangles[i].dirty != 0 { continue; }

        for j in 0..3 {
            if self.triangles[i].err[j] < threshold {
                let i0 = self.triangles[i].v[j] as usize;
                let i1 = self.triangles[i].v[(j + 1) % 3] as usize;

                if self.vertices[i0].border != self.vertices[i1].border { continue; }

                let mut p = Vector3::new(0.0f32, 0.0, 0.0);
```

```
                self.calculate_error(i0 as u32, i1 as u32, &mut p);

                deleted0.resize(self.vertices[i0].tcount as usize, 0);
                deleted1.resize(self.vertices[i1].tcount as usize, 0);

                if self.flipped(&p, i1 as u32, i0, &mut deleted0) { continue; }
                if self.flipped(&p, i0 as u32, i1, &mut deleted1) { continue; }

                self.vertices[i0].p = p;
                self.vertices[i0].q = self.vertices[i1].q + self.vertices[i0].q;
                let tstart = self.refs.len();

                self.update_triangles(i0 as u32, i0, &deleted0, &mut deleted_triangles);
                self.update_triangles(i0 as u32, i1, &deleted1, &mut deleted_triangles);

                let tcount = self.refs.len() - tstart;

                if tcount <= self.vertices[i0].tcount as usize {
                    for i in 0..tcount {
                        self.refs[self.vertices[i0].tstart as usize + i] = self.refs[tstart + i].clone();
                    }
                }
                else {
                    self.vertices[i0].tstart = tstart as i32;
                }

                self.vertices[i0].tcount = tcount as i32;
                break;
```

```
                }
            }

            if triangle_count - deleted_triangles <= target_count { break; }
        }
    }

    self.clean_mesh();
  }
}
```