МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КРОСПЛАТФОРМОВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТРИАНГУЛЯЦІЇ ПОЛІГОНАЛЬНИХ ОБЛАСТЕЙ»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.П. Піддубняк
(ініціали та прізвище)

Керівник декан математичного факультету, д.т.н., професор Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, д.т.н., професор Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Піддубняку Даніілу Петровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка кросплатформового програмного забезпечення тріангуляції полігональних областей

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка кросплатформового програмного забезпечення тріангуляції. полігональних областей

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	15.02.2023	
3.	Обробка методичних та теоретичних	01.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	05.04.2023	
5.	Розробка третього розділу.	15.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	23.06.2023	

Студент _____ Д.П. Піддубняк
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка кросплатформового програмного забезпечення триангуляції полігональних областей»: 51 с., 23 рис., 8 джерел, 1 додаток.

ГЕОМЕТРИЧНИЙ ОБ’ЄКТ, КРОСПЛАТФОРМОВЕ ПРОГРАМУВАННЯ, ПОЛІГОНАЛЬНІ ОБЛАСТІ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ПРОСТІР, СИМПЛЕКС, ТРИАНГУЛЯЦІЯ.

Об’єкт дослідження – кросплатформове програмне забезпечення та триангуляція полігональних областей.

Мета роботи: дослідження та розробка кросплатформового програмного забезпечення триангуляції полігональних областей.

Методи дослідження – порівняння, аналіз, описовий, структурний.

Актуальність кваліфікаційної роботи обумовлена розвитком сучасних операційних систем та платформ та засобів взаємодії з ними, а також еволюцією сучасних технологій у сфері комп’ютерної графіки.

У даній кваліфікаційній роботі проведено дослідження існуючих засобів розробки кросплатформового програмного забезпечення та алгоритмів триангуляції полігональних областей.

SUMMARY

Bachelor’s qualifying paper «Development of a Cross-platform Software for Triangulation of Polygonal Areas»: 51 pages, 23 figures, 8 references, 1 supplement.

GEOMETRIC OBJECT, CROSSPLATFORM PROGRAMMING, POLYGONAL AREAS, SOFTWARE, SPACE, SIMPLEX, TRIANGULATION.

Object of the study is cross-platform software and triangulation of polygonal areas.

Aim of the study is research and development of cross-platform polygon area triangulation software.

Methods of research – comparison, analysis, descriptive, structural.

The relevance of qualification work is due to the development of modern operating systems and platforms and means of interaction with them, as well as the evolution of modern technologies in the field of computer graphics.

In this qualification work, a study of existing tools for the development of cross-platform software and algorithms for triangulation of polygonal areas was carried out.

ЗМІСТ

ВСТУП

При вирішенні різних завдань математичного моделювання широко використовуються проєкційно-сіткові методи. Їх використання передбачає попередню побудову так званої «сітки», тобто певної топологічної множини точок, «вершин», «вузлів», пов’язаних між собою «ребрами» – відрізками прямих, а в деяких випадках і кривих, ліній таким чином, що вихідна область розбивається на елементи певної форми. При цьому як елементи сітки, якщо йдеться про геометрично складні області, зазвичай використовуються геометричні симплекси, тобто трикутники в двовимірному і тетраедри в тривимірному випадку. Процес побудови сітки зазвичай називається дискретизацією чи триангуляцією.

Триангуляція використовується в різних сферах діяльності людини, таких, як медицина, будівництво, фізика, хімія і т.д. Наприклад, у медицині за допомогою триангуляції можна вирішити питання діагностики обсягу ураження органу, планування та характеру оперативного втручання, правильної оцінки стану кісткових фрагментів, а в будівництві – розрахунок характеристик міцності будівлі.

Розвиток обчислювальної техніки сприяв значному прогресу в області чисельних методів взагалі та в галузі методів триангуляції зокрема. Розроблено нові класи методів, значно більш ресурсомісткі, але водночас і більш ефективні. У той же час під багато емпіричних методів триангуляції підведено теоретичну базу. В даний час для двовимірної триангуляції розроблено та теоретично обґрунтовано ефективні та надійні методи побудови та оптимізації сіток; властивості елементів-трикутників – добре вивчені. Проте більшість методів тривимірної триангуляції теоретично не обґрунтовано, а багато завдань взагалі не вирішено.

Важливим завданням є автоматична побудова триангуляційної сітки об’єкта, представленого багатокутником, так як надалі вона може

використовуватися для тріангуляції складніших областей. І її застосування значно економить машинні ресурси та час.

Мета роботи – створення кросплатформного програмного забезпечення, яке виконує автоматичну побудову тріангуляційної сітки.

1 СУЧАСНІ ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кросплатформні додатки – це програми, які працюють відразу на кількох операційних системах. Завдання розробників полягає у написанні коду, який добре розгортається на всіх операційних системах.

Універсальний підхід до розробки дозволяє виконати дві важливі умови: економія часу та коштів. Кросплатформова технологія конкурує з нативними рішеннями і в деяких аспектах навіть перевершує їх. Вона забезпечує високу продуктивність без шкоди для гнучкості та кастомізації додатку. Далі розберемо докладніше, як класифікуються додатки.

1.1 Нативні додатки

Нативні додатки (від англ. native – рідний) розробляються під конкретну апаратно-програмну платформу і пишуться мовами, створених цієї платформи. І iOs, і Android мають свої SDK (від англ. Software Development Kit – набір засобів розробки) і свій стек технологій, зав’язані певною мовою програмування. Наприклад, рідними мовами для Android є Java та Kotlin, для iOS, відповідно – Swift та Objective-C.

Нативні додатки завантажуються через магазини програм (App Store, Google Play і т.д.) встановлюються в смартфоні.

Такі додатки розробляються спеціально під конкретну платформу:

– під iOS для iPhone;
– під Android для пристроїв під керуванням ОС Android;
– під Windows для Windows Phone.

Створення нативних програм вимагають від розробника спеціальних знань та умінь для роботи в конкретному середовищі розробки (xCode для iPhone, eclipse для пристроїв на Android) [1]. У цьому випадку використовуються лише

«рідні» мови програмування написання таких додатків.

Отже, нативні програми створені під конкретну ОС і органічно виглядають на смартфоні. Такі програми легко можуть використовувати всі функції ПЗ смартфона (камера, мікрофон, акселерометр, геолокація, адресна книга, плеєр і т.д.). При цьому дбайливо витрачають ресурси телефону (акумулятор, пам’ять). Залежно від призначення програми, передбачають чи не припускають наявність інтернет-з’єднання.

Приклад нативної програми:

Instagram:

– встановлюється із магазину додатків;
– для роботи також потрібний доступ в інтернет;
– використовує програмне забезпечення смартфона: камера, геолокація, адресна книга;
– можна увімкнути отримання push-повідомлень.

Переваги:

– органічно виглядають на телефоні, тому що розробляються для певної операційної системи;
– використовують усі функції програмного забезпечення гаджета: гіроскоп, компас, датчик освітленості/наближення, мікрофон, камеру, диктофон, геолокацію, адресну книгу та інші;
– мають швидкий відгук та плавність анімацій інтерфейсу;
– надійніше (працюють стабільніше);
– економніше витрачають ресурси телефону: батарею, пам’ять.

Недоліки:

– під кожну платформу потрібно писати окрему програму окремим розробником/-ками, що призводить до додаткових фінансових та тимчасових витрат;
– ціна.

Нативні програми – недешеве задоволення. Їх використовують для масштабних проєктів (наприклад, соцмережі) та для проєктів, які

масштабуються, доповнюються, змінюються. Їм не варто боятися частих оновлень фреймворку (середовища розробки), які можуть спричинити неможливість подальшого розвитку програми, створеної на старій версії;

1.2 Гібридні додатки

Гібридні додатки поєднують деякі функції нативних і веб-додатків (сайт, оптимізований під смартфон): кросплатформенність і можливість використання ПО телефону.

Такі додатки можуть бути завантажені через магазини програм, і при цьому мають можливість незалежного оновлення інформації. Гібридні програми вимагають підключення до Інтернету, оскільки веб частина оновлюється через Інтернет.

TripCase – органайзер для планування подорожей:

– завантажується із магазину;
– може використовувати геолокацію;
– необхідне підключення до Інтернету;
– може використовувати стільникову мережу;
– можна настроїти push-сповіщення.

Переваги:

– запускається не з браузера, можливе незалежне оновлення;
– швидкість розробки, а отже, і ціна значно нижча, ніж нативні програми;
– можливість розповсюджуватися через магазини програм (на відміну від веб-застосунків).

Недоліки:

– швидкість та стабільність роботи залишають бажати кращого;
– складності у масштабуванні та розвитку проєкту;
– складнощі та нестабільність при використанні різних модулів/функцій смартфона.

1.3 Прогресивні додатки

PWA або Progressive Web Application – технологія, яка дозволяє клієнтам встановити ваш сайт на смартфон як програму. Українською мовою перекладається як «прогресивний веб-додаток». Тепер не потрібно розробляти окремо сайт, додаток під iOS та додаток під Android. Достатньо мати і підтримувати лише сайт [2].

Програми на базі PWA ми використовуємо частіше, ніж здається. Бренди Twitter, Tinder, Uber, Telegram, Starbucks, Forbes, AliExpress, Aviasales використовують програми на базі PWA як основу або на додаток до мобільного додатка [3].

Переваги:

- дозволяє відправляти Push-сповіщення;
- розробка та підтримка на 70 % дешевше, ніж у мобільних додатків;
- економія на повторному залученні клієнта, оскільки немає витрат за інструменти повернення клієнта (ретаргетинг, ремаркетинг);
- не потрібно розміщувати програму в AppStore та Google Play, його можна завантажити безпосередньо з вашого сайту;
- працює швидше за сайт;
- працює без інтернету;
- розмір PWA програми, як правило, не перевищує 1 мб (це менше, ніж мобільний додаток);
- PWA програма працює з функціями: геолокація, камера, мікрофон.

Недоліки:

- PWA додаток немає можливості відправляти повідомлення на iPhone, тому що в iOS браузер за замовчуванням Safari. Однак у iOS 14 Apple додала функцію вибору браузера за промовчанням (це означає, що якщо користувач вибере стандартним браузером не Safari, повідомлення можна буде відправляти і на iPhone);
- також PWA програми не підтримують функцію Touch ID та обмежено працюють з функцією Bluetooth.

1.4 Висновки до розділу 1

Враховуючи розглянуті сучасні засоби розробки програмного забезпечення та на основі аналізу переваг та недоліків, очевидні переваги мають саме прогресивні веб додатки. В даній кваліфікаційні роботі буде використано саме метод розробки прогресивного веб додатку для реалізації кросплатформного програмного забезпечення триангуляції полігональних областей.

2 ТРИАНГУЛЯЦІЯ ПОЛІГОНАЛЬНИХ ОБЛАСТЕЙ

Майбутнє машинної графіки пов’язане із формуванням реалістичних тривимірних зображень. Необхідність їх формування виникає в багатьох сферах діяльності людини, наприклад, у машинобудівному та архітектурному моделюванні, дизайні, рекламі, мультиплікації тощо.

Інтерес до синтезу тривимірних (3D) зображень пояснюється тим, що вони найбільше реально відображають навколишню дійсність. 3D зображення є найбільш інформативними, оскільки дозволяють оцінити конструктивні та естетичні особливості об’єктів.

Системи синтезу реалістичних зображень повинні забезпечувати передачу всіх властивостей об’єкта, що моделюється: об’ємність, розташування, передачу півтонів, тіні, освітлення, текстури поверхні. Чим вище рівень реалістичності зображення, тим більше потрібно обчислень для його формування.

2.1 Машинна графіка

Графічний конвеєр – це логічна сукупність обчислень, які виконуються послідовно і дають на виході синтезовану сцену. Обчислення в конвеєрі розділені на кілька етапів, у кожному з яких апаратно чи програмно виконується певна функція.

Більшість програм використовують стандартну схему побудови 3D сцен. Для скорочення тимчасових витрат за використовуються апаратні можливості із застосуванням прикладних інтерфейсів API.

Етапи конвеєра можна розділити на 2 основні стадії: етап геометричних перетворень та етап рендерингу.

На початковому етапі виконується опис тривимірної сцени, зображення

якої необхідно синтезувати. Тривимірні об’єкти зазвичай конструюються з графічних примітивів, переважно з трикутників, оскільки трикутник є найпростішим полігоном, однозначно задає найпростішу площину в просторі і забезпечує рішення задачі декомпозиції. Будь-яка поверхня може бути апроксимована сіткою трикутників і якщо така сітка досить добре складена, то з її допомогою можна представити будь-яку поверхню з необхідною точністю.

Процес розбиття об’єкта, що моделюється, на трикутники називається – триангуляцією. Зазвичай, вона виконується програмно. Найпростішим рішенням завдання триангуляції є розщеплення полігону вздовж деякої хорди на два полігони і подальше рекурсивне розбиття їх до ситуації, коли полігон, що підлягає триангуляції, є трикутником. Даний алгоритм застосовується лише для опуклих полігонів. Кожен пакет 3D моделювання здійснює триангуляцію декількома способами залежно від поставленого завдання. Наприклад, при моделюванні віддалених об’єктів немає сенсу застосовувати дуже детальну мережу для їх опису і, навпаки, для близьких об’єктів використовують більше трикутників.

Етап модельних перетворень включає афінні операції перенесення, повороту та зміну масштабу. Перетворення дозволяють переміщати об’єкти у сцені та маніпулювати сюжетом.

Висвітлення. На цьому етапі вибираються моделі освітлення та обчислюється освітленість об’єктів. Модель освітлення визначає тип використовуваних джерел світла. Освітленість та тонування поверхонь об’єктів визначається розташуванням джерел світла та їх типом, а також оптичними властивостями матеріалу, з якого виконані поверхні. Загальноприйняті моделі освітлення включають розсіяне світло, спрямоване та точкове джерело світла. Об’єкт буде видимим, якщо його поверхня відбиває чи пропускає світло. Якщо об’єкт при цьому поглинає деякі довжини хвиль, то він набуває певного кольору. Методи, що використовуються для моделювання освітленості та тонування поверхні, оперують з відбитим світлом. Його властивості залежать від будови,

напряму та форми джерела світла, а також орієнтації та властивостей поверхні. Відбите світло має дві складові – дифузну та дзеркальну. Дзеркальне відображення поверхні випромінює світло лише в одному напрямку. Більшість поверхонь реальних об’єктів має властивості дифузного і дзеркального відображення.

Видові перетворення. Тут визначаються нові координати всім вершин примітивів, з положення спостерігача і напрями його погляду. Сцена проєктується на екранну систему координат. Для відображення тривимірного об’єкта на двовимірний екран або інший зовнішній пристрій використовується математичне перетворення, яке називається проєктуванням. Точки, що визначають відрізки прямих, криві та інші елементи проєктуються на двовимірну площину, при цьому віртуальна проєкційна площина, звана картинною площиною, поміщається між об’єктом та спостерігачем, перпендикулярно до напрямку погляду. Проводяться проєкційні лінії від точок об’єкта до спостерігача. Точки перетину картинної площини з променями проєктування є відповідними точками проєкції.

Проєкції поділяють на два різні класи: паралельні та перспективні. При паралельному перетворенні проєкційні лінії йдуть паралельно до погляду спостерігача. При перспективному проєктуванні ці лінії перетинаються у вічі спостерігача. Перспективні проєкції виглядають більш реалістично, ніж паралельні, проте паралельне проєктування потребує менших обчислювальних витрат.

Видалення невидимих поверхонь – на цьому етапі зі списку примітивів виключаються повністю невидимі поверхні, що залишилися позаду або збоку. У програмах машинної графіки видалення невидимих частин зображення найбільшого поширення отримали метод трасування променів, метод рядкового сканування і метод Z-буфера. Стандартом OpenGL рекомендовано використовувати метод Z-буфера, реалізований практично у всіх графічних акселераторах. Метод призначений для зберігання інформації, необхідної для

правильного відображення по глибині видимих об’єктів, залежно від положення спостерігача. У процесі сканування поверхня моделі перетворюється на значення пікселів у відеобуфері. Одночасно з цим обчислюється відстань від кожної точки поверхні до деякої довільної, але фіксованої площини, розташованої за сценою. Ці відстані зберігаються в окремій частині пам’яті, званої Z-буфером, яка містить дані для кожного пікселя зображення. Комп’ютер сканує цю модель, і якщо відстань, обчислена для даного пікселя, більша, ніж раніше для нього ж записане, то Z-буфер оновлюється – в нього заноситься більше з двох значень, і пікселя приписується колір поверхні, що розглядається. Якщо ж ця відстань менша, ніж раніше записана (це вказує на те, що точка лежить на поверхні, яка розташована за фрагментом, розглянутим раніше), то в осередку Z-буфера залишається старе значення і колір піксела не змінюється. Після перегляду всіх фрагментів сцени відеобуфер міститиме зображення з віддаленими невидимими поверхнями.

Рендеринг – це процес перетворення об’єкта або сцени, створених у додатку тривимірної графіки для виведення на екран. На стадії рендерингу визначаються пікселі зображення та їх адреси. На відміну від стадії геометричних перетворень, у процесі рендерингу обсяг операцій із плаваючою точкою значно меншийі в основному складається з простих операцій над пікселями. Стадія рендерингу включає такі процедури [4].

Накладення текстури чи текстурування це етап, з якого на поверхню об’єкта накладається деяке зображення, зване зображенням текстури. У загальному контексті конвеєра візуалізації цей метод відкриває нетривіальні можливості. Перерахуємо деякі області, де застосування текстур виявляється дуже корисним:

– текстури можна використовувати у тому, щоб показати матеріал, з якого зроблено об’єкт;
– за допомогою текстурування можна наочно уявити фізичні властивості об’єктів у додатках наукової візуалізації (наприклад, дані про температуру кодуються кольором і наносяться на об’єкт, дозволяючи

бачити, як геометрія впливає на перебіг процесів теплоперенесення);

– текстури дають можливість моделювати світлові ефекти, наприклад, відображення, при створенні фотореалістичних зображень.

Текстури складаються з текселів, еквівалентних пікселям дисплея. Для відображення на дисплеї текселі проєктуються на пікселі.

Процес накладання текстури суперечить точності уявлення об’єкта залежно від віддаленості від спостерігача. При видаленні текстура має ставати дрібнішими. Коректне уявлення текстур у просторовій перспективі забезпечується кількома прийомами. Для представлення текстур на різній відстані від спостерігача та для трикутників різного розміру використовується серія текстурних карток різного розміру, звана міп-меппінгом. У процесі рендерингу об’єкта використовується та текстурна карта з міп-мепінгу, яка найбільше відповідає розмірам об’єкта та відстані його до спостерігача. Білінійна фільтрація передбачає вибір текстурної картки з міп-мепінгу та обчислення виваженої суми найближчих сусідніх текселів для отримання пікселя, який буде виведений на дисплей. Трилінійна фільтрація має велику обчислювальну складність, оскільки передбачає обробку двох найближчих текстурних карток з міп-мепінгу. Над кожною з пари текстурних карток виконується білінійна фільтрація, а з отриманої пари значень береться зважена сума, яка і є результатом. Ще однією важливою особливістю є можливість корекції перспективи під час накладання текстури. Корекція просторової перспективи необхідна, коли об’єкт має значну довжину і текстура на об’єкті має цю довжину відобразити.

Забарвлення примітивів. Для того, щоб створювати правдоподібні зображення, застосовують спеціальні алгоритми забарвлення для імітації нерівномірного освітлення. Для виконання цього завдання використовують найчастіше три моделі: плоске зафарбовування, зафарбовування по Гуро та Фонгу. Плоске зафарбовування не потребує великих обчислювальних витрат. Отримавши опис джерела світла, що висвітлює об’єкти сцени, визначають

відтінки кольорів, ґрунтуючись на тому, під яким кутом падає світло. Чим більше повернена до джерела поверхня об’єкта, тим яскравіше вона зафарбована. Забарвлення по Гуро – згладжує тоновий перехід між ребрами полігону, обчислюючи засобами лінійної інтерполяції інтенсивність кольору кожного пікселя полігону. Отримана поверхня більш природно моделює віддзеркалення світла та робить візуалізацію реалістичнішою. Модель забарвлення Гуро застосовується як стандартний метод OpenGL. На відміну від лінійної інтерполяції інтенсивності кольору вершин Гуро, забарвлення Фонга інтерполює вектори нормалей і робить розрахунок освітленості для кожного пікселя. Зафарбування по Фонгу дозволяє отримати правильне сприйняття кривизни та світлові відблиски, проте через складність розрахунків для її створення буде потрібно набагато більше часу.

Згладжування, фінальна обробка. На цьому заключному етапі рендерингу відбувається обробка всієї результуючої сцени. Оскільки графічні пристрої є дискретними, то при поданні об’єкта, що має похилі лінії, виникає ефект ступінчастості. Алгоритм згладжування (antialiasing) знижує прояв цього ефекту. Метод повноекранного згладжування (full screen antialiasing) дозволяє значно покращити якість сцени, що виводиться за рахунок розрахунку зображення з великим розширенням і подальшим усередненням значень кольору сусідніх пікселів. Подібно усувають різкі межі між полігональними областями. На цьому етапі застосовуються інші ефекти, такі як змащування (smooth), туман (fogging), які надають сцені більшої реалістичності.

2.2 Загальні положення триангуляції

Генерація об’ємних зображень представляє складне обчислювальне завдання, у зв’язку з чим практично виконують її декомпозицію. Складні зображення формують фрагментарно, навіщо їх розбивають на складові. Процес

розбиття поверхні об’єктів на полігони отримав назву тесселяції. Ця стадія на даному етапі розвитку машинної графіки проводиться повністю програмно, незалежно від технічного рівня 3D-апаратури.

В сучасності з’явилася велика різноманітність графічних акселераторів, які мають апаратні засоби для забарвлення тривимірних об’єктів, видалення невидимих частин, накладання текстур та інші графічні функції. Для використання переваг 3D-прискорювачів необхідно програмно провести тестування вихідних об’єктів, передаючи отримані полігональні області для подальшої обробки акселератором.

На практиці найчастіше використовується розбиття зображень на трикутники. Це пояснюється такими причинами:

– трикутник є найпростішим полігоном, вершини якого однозначно задають грань;
– будь-яку область можна гарантовано розбити на трикутники;
– обчислювальна складність алгоритмів розбиття на трикутники істотно менше, ніж при використанні інших полігонів;
– реалізація процедур рендерингу найбільш проста для області, обмеженої трикутником;
– для трикутника легко визначити три його найближчих сусіди, які мають з ним загальні грані.

Процес розбиття полігональної області зі складною конфігурацією набір трикутників називається триангуляція.

При аналізі чи синтезі складних поверхонь їх апроксимують сіткою трикутників, згодом надалі оперують з найпростішими полігональними областями. Особливий інтерес до алгоритмів триангуляції визначається тим, що вони використовуються в багатьох процедурах машинної графіки, таких як фарбування, видалення невидимих частин, відсікання, формування поверхонь.

Наведемо приклад створення напівсфери, проілюструвавши етапи рендерингу. Полігональна область (див. рис. 2.1) є сукупністю трикутників (див.

рис. 2.2). При цьому важливо досить точно апроксимувати зовнішній контур.

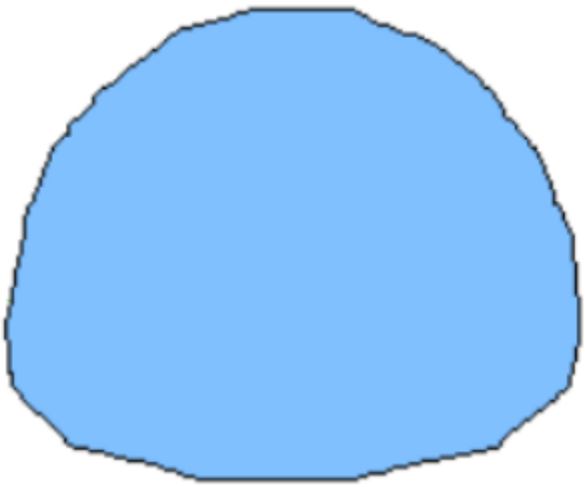

Рисунок 2.1 – Полігональна область

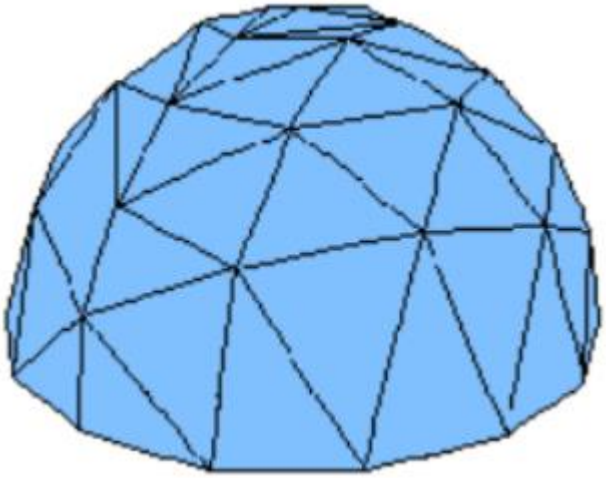

Рисунок 2.2 – Сукупність трикутників на полігональній області

Будь-яка поверхня може бути апроксимована з необхідною точністю сіткою трикутників. Точність апроксимації визначається кількістю трикутників та способом їх вибору. Для якісної візуалізації об’єкта, що знаходиться поблизу точки спостереження, потрібно врахувати у багато разів більше трикутників, ніж у ситуації, коли той самий об’єкт розташований на відстані. Навіть досить груба сітка корисна практично, оскільки методи згладжування, що застосовуються у процесі відображення, можуть значно поліпшити уявлення про кривизну

поверхні.

Наступний етап – зафарбовування граней (рис. 2.3), обмежених трикутниками.

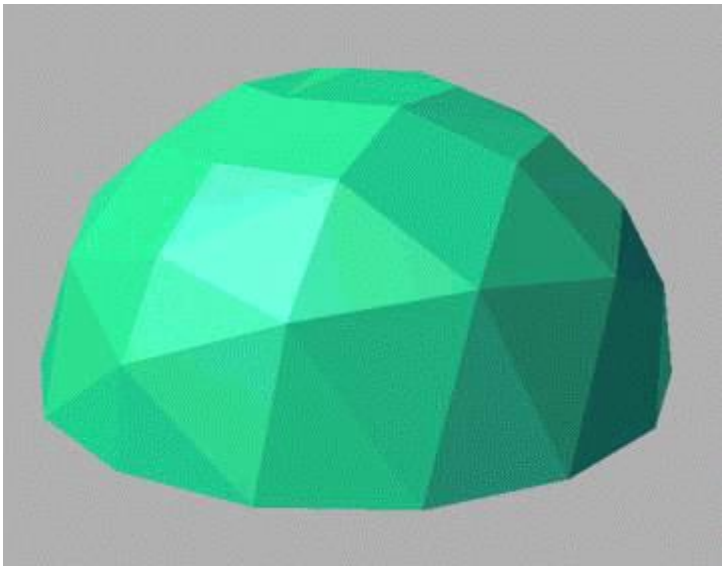

Рисунок 2.3 – Зафарбовування граней

Останнім етапом є застосування алгоритмів згладжування, які усувають ребристість поверхні, ефект Маха. На малюнку 2.4 представлена вже гладка напівсфера з плавним переходом напівтонів та областями освітлення.

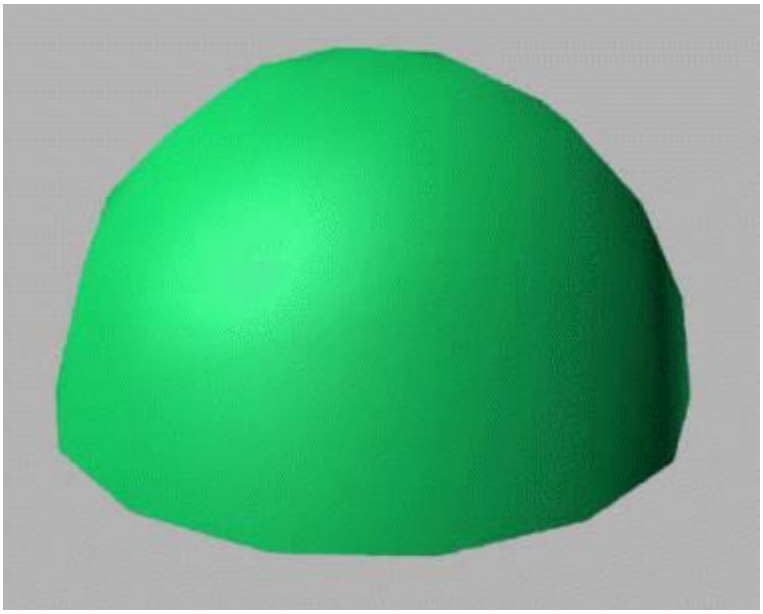

Рисунок 2.4 – Гладка напівсфера

Під час моделювання об’єкта виникає завдання, яку кількість трикутників потрібно використовувати. З одного боку потрібно досягти максимальної якості при остаточній візуалізації, а з іншого – знизити навантаження на процесор і 3D-акселератор. Для зниження навантаження можна використовувати менш деталізовану триангуляційну мережу, але при цьому якість буде далеко не найкращою.

При формуванні динамічних зображень точність представлення об’єктів у кадрі через його переміщення не сприймається людиною, у зв’язку з чим недоцільно використовувати детальну триангуляцію.

При виборі підходів та алгоритмів триангуляції слід керуватися метою, що ставиться конкретним завданням. Наприклад, при моделюванні об’єкта, що знаходиться вдалині, немає сенсу застосовувати дуже детальну мережу для його опису і, навпаки, для близьких об’єктів це дуже важливо, наприклад, для побудови моделі голови людини потрібно близько 60 тис. трикутників.

2.3 Алгоритми триангуляції

Розглянемо найпоширеніші алгоритми триангуляції полігонів. Найпростіше рішення задачі триангуляції полягає в розщепленні полігону вздовж деякої хорди на два полігони і в подальшому рекурсивному розбитті їх до ситуації, коли полігон, що підлягає триангуляції, є трикутником.

Даний спосіб застосовується лише для триангуляції опуклих полігонів. Випуклим вважається полігон, якщо відрізок прямий, що з’єднує будь-які дві точки, повністю лежить у внутрішній області (див. рис. 2.5). Полігон рисунку 2.6 перестав бути опуклим, оскільки відрізок ab виходить за межі багатокутника.

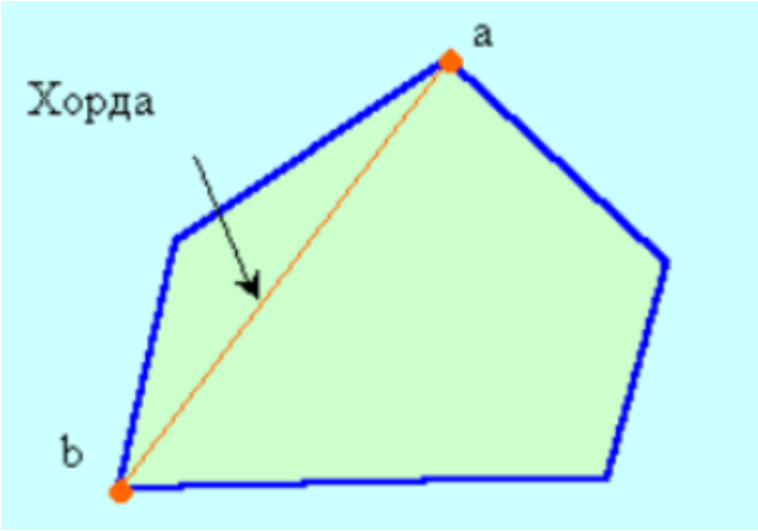

Рисунок 2.5 – Опуклий полігон

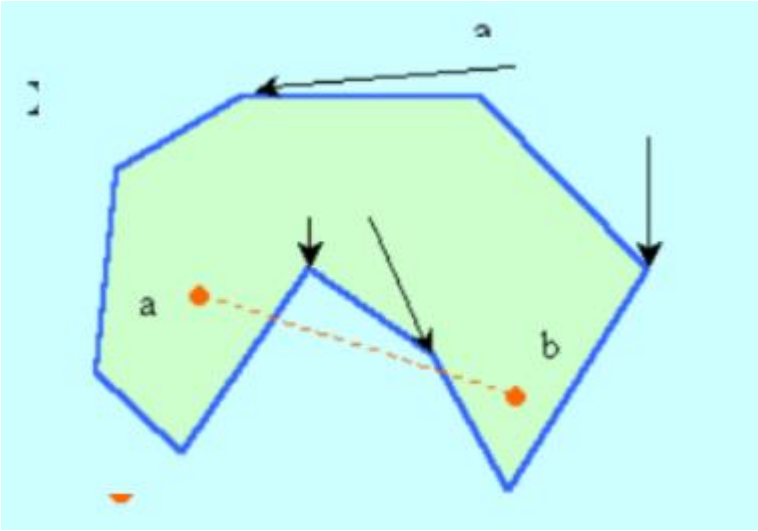

Рисунок 2.6 – Неопуклий полігон

Триангуляція неопуклих полігонів не така проста, тому попередня розбивка неопуклих багатокутників на опуклі істотно спрощує алгоритми їх подальшої обробки. Найпростіше це можна зробити шляхом послідовного перенесення і повороту багатокутника так, щоб одна з його вершин збігалася з початком координат, а сторона, що виходить з неї, – з віссю Ох. При розташуванні будь-яких сторін нижче осі відбувається їх відсікання, і алгоритм рекурсивно повторюють для отриманих полігонів, поки вони не стануть опуклими.

Наведемо універсальний алгоритм. Триангуляцію будь-якого полігону

можна здійснити за наступним універсальним алгоритмом.

Вибирається крайня ліва вершина і між двома її суміжними сторонами проводиться діагональ. При цьому можуть мати місце такі два випадки: рис. 2.7 – діагональ ac є хордою; рис. 2.8 – діагональ ac не хорда, так як всередину трикутника abc потрапляє d полігону (загалом їх може бути кілька). З усіх вершин усередині трикутника abc вершина d найбільш віддалена від боку ac. Цю вершину називатимемо вторинною. У разі, коли вторинної вершини немає, то отриманий трикутник заносять у сітку трикутників, і алгоритм рекурсивно обробляє полігон, що залишився до тих пір, поки він не виродиться в трикутник.

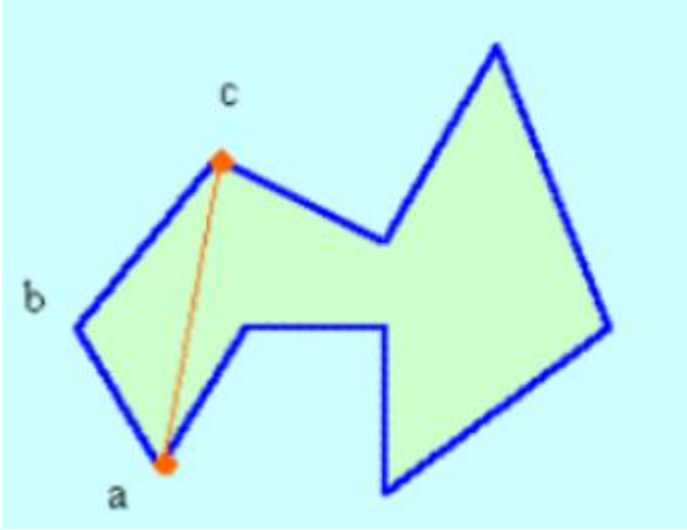

Рисунок 2.7 – Хорда

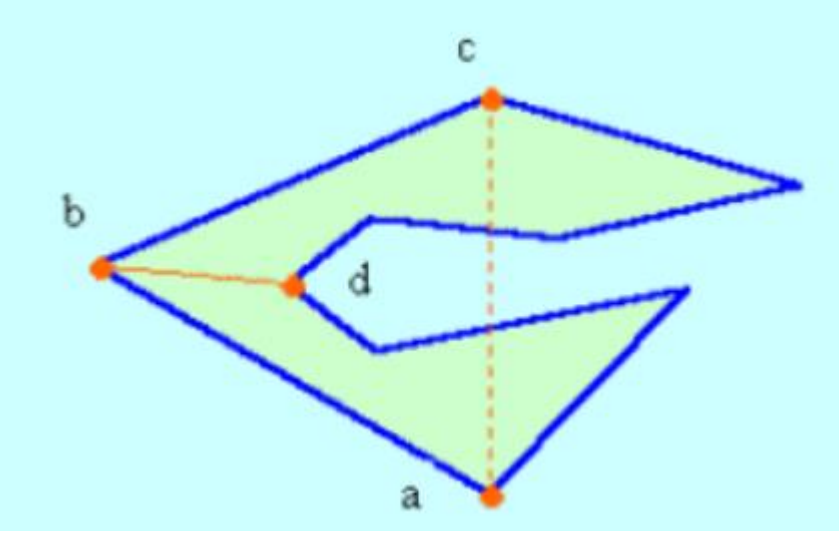

Рисунок 2.8 – Виявлена вершина, що вторгається

При виявленні вершини, що вторгається, проводиться діагональ з поточної до вторинної вершини. Отримані полігони рекурсивно обробляють до отримання трикутників.

Один із підходів до триангуляції полягає у знаходженні внутрішньої точки області, обмеженої полігоном, та поєднанні з нею всіх вершин.

З огляду на те, що триангуляція є підготовчим етапом рендерингу, то до вибору внутрішньої точки можуть бути визначені певні вимоги. Так, наприклад, при багатопроцесорній обробці доцільно внутрішню точку вибрати таким чином, щоб площа трикутників була приблизно однаковою. У цьому випадку буде забезпечено однаковий ступінь обчислювального завантаження апаратури.

В одному з найпоширенішого програмного інтерфейсу (API) для розробки додатків у галузі двовимірної та тривимірної графіки стандарті OpenGL для формування триангуляційної сітки використовуються такі команди:

GL_TRIANGLES. Трикутники. Команда малює серію трикутників, використовуючи трійки вершин v0, v1 та v2, потім v3, v4 та v5 тощо (рис. 2.9).

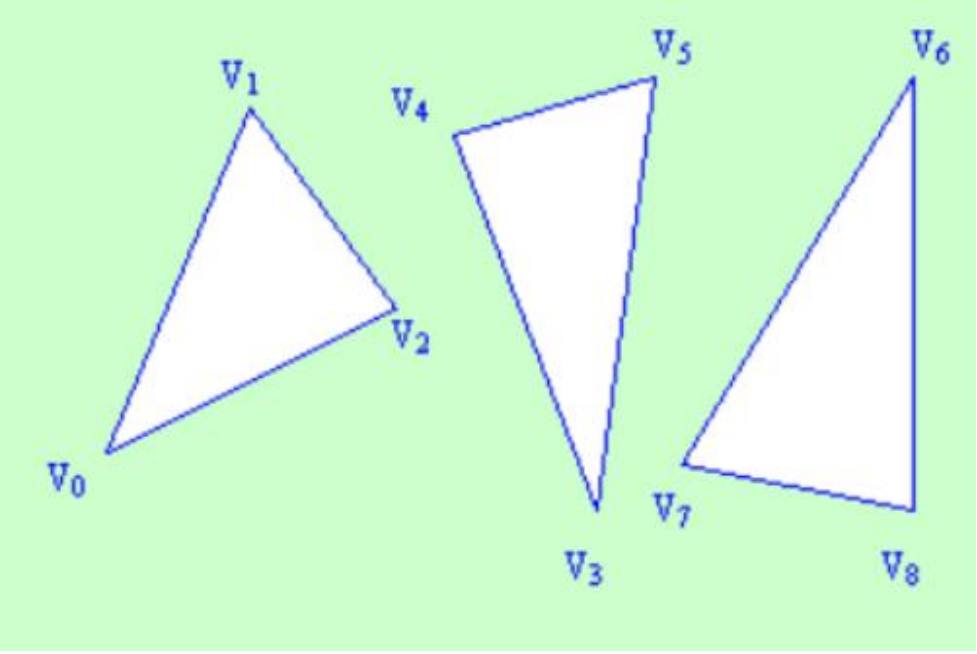

Рисунок 2.9 – Серія трикутників

Якщо кількість вершин не кратна 3, точки, що залишилися, ігноруються. GL_TRIANGLE_STRIP. Смуга трикутників. Команда малює серію трикутників,

використовуючи вершини в наступному порядку: v0, v1, v2, потім v2, v1, v3, потім v2, v3, v4 і т.д. Порядок вершин має гарантувати, що трикутники мають правильну орієнтацію. Смуга може бути використана для формування частини поверхні.

GL_TRIANGLE_FAN. Команда схожа на GL_TRIANGLE_STRIP, але при формуванні трикутників використовують вершини в такому порядку: v0, v1, v2, потім v0, v2, v3, потім v0, v3, v4 і т.д. (рис. 2.10). Усі трикутники мають загальну вершину v0.

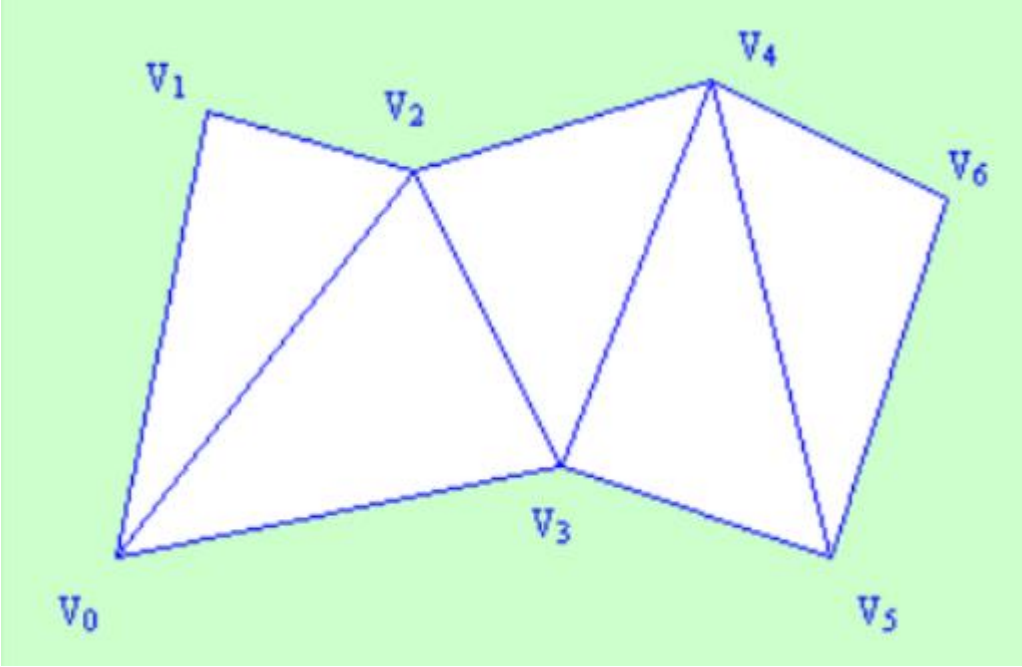

Рисунок 2.10 – Формування трикутників за вершиною v0

У машинній графіці завдання тріангуляції можна розглядати у двох напрямках – це тріангуляція полігональних областей та тріангуляція набору точок. Остання має місце при описі поверхні набором крапок та інтенсивністю їх кольорів. Поточковий опис поверхонь застосовують у тих випадках, коли поверхня дуже складна і не має гладкості, а детальне уявлення численних геометричних особливостей важливе для практики. До таких поверхонь можна віднести ділянки ґрунту, форми малих небесних тіл, мікрооб’єкти, зняті за допомогою електронного мікроскопа та інші утворення зі складною формою.

Розглянемо триангуляцію точок на площині методом Делоне. Ця триангуляція добре збалансована в тому сенсі, що трикутники, що формуються, прагнуть рівнокутності (рис. 2.11). Триангульоване зображення рисунку 2.11 може бути віднесено до триангуляції Делоне.

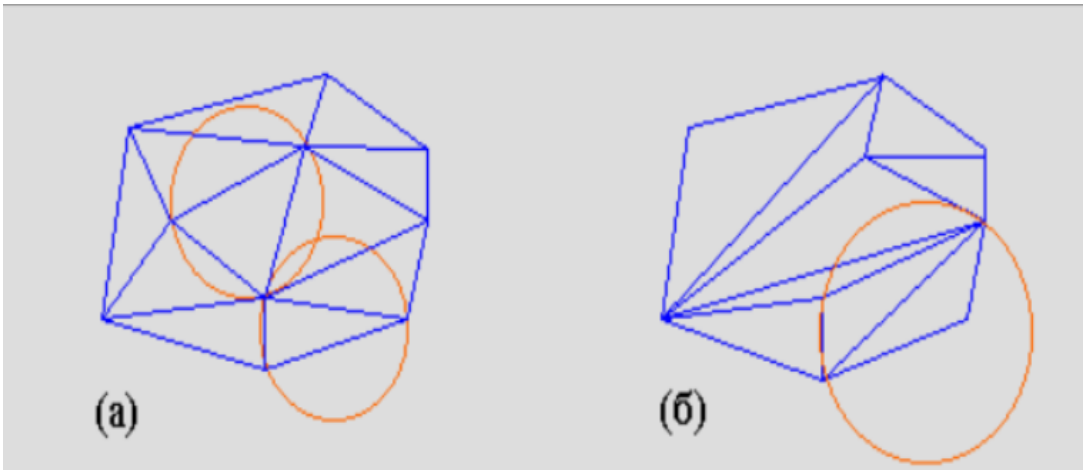

Рисунок 2.11 – Два варіанти триангуляції для одного набору точок

Триангуляція набору точок буде триангуляцією Делоне, якщо описане коло для кожного трикутника буде вільним від точок, тобто всередині її не буде більше жодної точки з набору. На малюнку 2.11(а) показано два кола, які не містять у собі інших точок (можна провести кола і для інших трикутників, щоб переконатися, що вони також вільні від точок набору). На малюнку 2.11(б) цього правила не дотримується – всередині області, обмеженою колом, потрапила одна точка іншого трикутника. Отже, ця триангуляція не стосується типу Делоне.

Алгоритм працює шляхом постійного нарощування до поточної триангуляції по одному трикутнику за крок. Спочатку триангуляція складається з одного ребра, після закінчення роботи триангуляція є триангуляцією Делоне. На кожній ітерації алгоритм шукає новий трикутник, який підключається до межі поточної триангуляції. Пошук і підключення трикутника образно можна уявити, як надування двовимірного міхура, прив’язаного до одного з ребер кордону. Якщо такий міхур досягає деякої, ще включеної в триангуляцію точки з набору, то утворюється трикутник. Якщо точка не зустрінута, обробляється наступне

ребро кордону.

Алгоритми триангуляції реалізовані в багатьох професійних графічних пакетах 3D-моделювання, таких як 3D Studio MAX, OpenGL Optimizer, LightWave та ін.

Враховуючи те, що алгоритми триангуляції є невід'ємною частиною практично всіх програмних продуктів 3D-графіки, інтенсивно ведуться роботи з їх удосконалення. Можна припустити, що в майбутньому вони будуть реалізовані апаратно в графічних акселераторах.

2.4 Висновки до розділу 2

У даному розділі розглянуті та обґрунтовані сучасні засоби візуалізації за допомогою комп'ютерної графіки, а також доведена актуальність методу триангуляції полігональних областей з детальнім аналізом існуючих сучасних алгоритмів реалізації наведеного методу візуалізації віртуальних тривімирних об'єктів. Результати аналізів показали, що триангуляція є сучасним та ефективним способом віртуалізації тривимірних об'єктів. У наступному розділі розглянемо використання отриманих результатів на практичному прикладі, розроблюючи кросплатформне програмне забезпечення.

3 РОЗРОБКА КРОСПЛАТФОРМОВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТРИАНГУЛЯЦІЇ ПОЛІГОНАЛЬНИХ ОБЛАСТЕЙ

3.1 Проєктування середовища розробки

Прогрес комп’ютерних технологій визначив процес появи нових різноманітних знакових систем запису алгоритмів – мов програмування [5]. Сенс появи такої мови – оснащений набір обчислювальних формул додаткової інформації, що перетворює цей набір на алгоритм. Мова програмування – формальна знакова система, призначена запису комп'ютерних програм. Мова програмування визначає набір лексичних, синтаксичних та семантичних правил, що задають зовнішній вигляд програми та дії, які виконає виконавець (комп’ютер) під її керуванням.

З часу створення перших програмованих машин людство придумало вже понад вісім з половиною тисяч мов програмування. Щороку їхня кількість поповнюється новими. Деякими мовами вміє користуватися лише невелика кількість їхніх власних розробників, інші стають відомими мільйонам людей. Професійні програмісти іноді застосовують у своїй роботі понад десяток різноманітних мов програмування.

Автори мов по-різному тлумачать поняття мову програмування. До найпоширеніших тверджень належать такі:

– функція: мова програмування призначена для написання комп’ютерних програм, що застосовуються для передачі комп’ютеру інструкцій щодо виконання того чи іншого обчислювального процесу та організації управління окремими процесами;
– завдання: мова програмування відрізняється від природних мов тим, що призначено передачі команд і даних від людини комп’ютеру, тоді як природні мови використовуються спілкування людей між собою. У принципі можна узагальнити визначення «мов програмування» – це

спосіб передачі команд, наказів, чіткого керівництва до дії, тоді як людські мови служать тільки для обміну інформацією;

– виконання: мова програмування може використовувати спеціальні конструкції для визначення та маніпуляції структурами даних та управління процесом обчислень.

Процес роботи комп’ютера полягає у виконанні програми, тобто набору певних команд у певному порядку. Машинний вид команди, що складається з нулів та одиниць, вказує, яку саме дію має виконати центральний процесор. Отже, щоб задати комп’ютеру послідовність дій, які він має виконати, потрібно задати послідовність двійкових кодів відповідних команд. Програми в машинних кодах складаються із тисячі команд. Писати такі програми – заняття складне та стомлююче. Програміст повинен пам’ятати комбінацію нулів та одиниць двійкового коду кожної програми, а також двійкові коди адрес даних, що використовуються при її виконанні. Набагато простіше написати програму якоюсь мовою, ближчою до природної людської мови, а роботу з перекладу цієї програми в машинні коди доручити комп’ютеру. Так виникли мови, призначені спеціально для написання програм – мови програмування. Серед однозначних лідерів мов програмування можна виділити JavaScript.

JavaScript – мова програмування, яка в 21 столітті стала виступати як одна з найпопулярніших. Вона використовується для створення веб-сторінок та інтернет-додатків. JavaScript – мова, яка у своїй основі містить скрипт. Це документ, що описує дії, необхідні для обробки та виконання. Працювати зі скриптами зможе кожен браузер, що робить мову кросплатформною. Важливий факт – найпопулярнішою мовою серед українських розробників за статистикою 2022 року залишається JavaScript – 18,8 % комерційного використання [6].

JavaScript – це мультипарадигмова мова. Вона підтримує кілька стилів одночасно:

– функціональний;

– об’єктно-орієнтований;

– імперативний.

Найбільша потреба у цієї мови спостерігається у веб-розробці. Програміст на JS може створити не лише програму «для браузера», але й невелику гру, а також повноцінний веб-сайт чи односторінок. Клієнти та сервери для JS – ключові сфери застосування на практиці.

До переваг JavaScript можна віднести такі моменти:

– незамінність у веб-розробці. JS – це основна «технологія» для клієнт-серверних моделей та програмування «для інтернету»;
– наявність повної інтеграції з верстkою сторінок та серверною частиною;
– раціональність застосування та простоту (вирішити елементарне завдання за допомогою JS вдається за кілька хвилин, сама мова має зрозумілий синтаксис, який не вимагає поглибленого вивчення);
– швидкість роботи. JS може зробити підсумковий додаток більш швидким та зручним;
– продуктивність;
– комфортність використання інтерфейсів користувача;
– наявність власної потужної екосистеми (інфраструктури) (особливо помітно останніми роками, приклад – поява величезної кількості корисних фреймворків, які підійдуть будь-якому випадку).

Чому потрібно використовувати JavaScript, зрозуміло – це зручна та функціональна мова. Але вона має свої недоліки. До них відносять:

– відсутність можливості читання та завантаження документів (основна причина наявності цього мінусу – міркування безпеки);
– відсутність віддаленого доступу (повноцінно для мережного програмного забезпечення відповідна мова не використовується);
– нестрогу типізацію;
– вільне трактування типів (нестиковки в коді ігноруватимуться);
– доступність конкурентам (пов’язано це з високою читальністю коду JavaScript).

В рамках даної кваліфікаційної роботи прийнято рішення використовувати

мову програмування JavaScript через актуальність, високий рівень підтримки та розвитку мови, велику кількість бібліотек та фреймворків, простоту використання та багатоплатформність.

Серед найпопулярніших багатоплатформних фреймворків JavaScript безпосередніми лідерами вважаються Angular, React та Vue. Проведемо порівняльний аналіз наведених фреймворків задля визначення найбільш доречного інструментарію для реалізації поставленої задачі.

React вважається бібліотекою інтерфейсу користувача, Angular – повномасштабним front-end фреймворком, що надає власні інструменти для всіх пов’язаних з розробкою веб-додатків функцій, а Vue – прогресивним фреймворком, реалізованим як додаткова розмітка для HTML.

Всі три фреймворки можуть використовуватися практично взаємозамінно для створення компонентних frontend-додатків з розширеними можливостями інтерфейсу користувача. Однак остаточний вибір залежить від вимог проєкту та переваг розробника.

Angular – це повноцінний фреймворк веб-додатків на основі TypeScript, безкоштовний і відкритий. Він широко використовується для створення односторінкових додатків. Він був спочатку запущений Google у 2016 році як продовження AngularJS, який був випущений у 2010 році.

На початкових етапах, коли Angular вийшов на ринок, він привернув увагу багатьох розробників завдяки своїм чудовим характеристикам, а також його підтримує технічний гігант Google. Багато людей плутають Angular з AngularJS, який пізніше став першою версією Angular, запущеною Google.

Angular використовує TypeScript для розробки, тоді як AngularJS використовує JavaScript. Різкий перехід від JavaScript до TypeScript розчарував багатьох людей, оскільки JavaScript є більш відомою мовою, ніж TypeScript.

Переваги використання Angular:

– надійність для майбутнього: оскільки Angular підтримується Google, розробники мають величезну довіру для створення великомасштабних програм, знаючи, що це буде підтримуватися в довгостроковій

перспективі;

– документація: документація Angular є детальною та дуже добре пояснює функціональність і роботу Angular, крім того, кожне поняття пояснюється на прикладі та легкою мовою, яка є корисною навіть для початківців;
– масштабність: робота над проєктом Angular у команді дуже масштабована, оскільки будь-які незначні зміни, внесені будь-яким членом команди, не потребують оновлення всієї структури проєкту, крім того, кодова база є високоузгодженою та читабельною, що підвищує ефективність проєкту.

Обмеження використання Angular:

– крива навчання: Angular вимагає від вас навичок роботи з TypeScript, якому, згідно з опитуванням StackOverflow, віддають перевагу 30% розробників, тому вивчення Angular вимагає більше зусиль, ніж інші фреймворки;
– розмір проєкту: розмір проєкту визначає багато параметрів для програми, однак це помітно лише в невеликих програмах, тоді як у великих програмах усі проєкти важать приблизно однаково.

React – це інтерфейсна бібліотека JavaScript з відкритим кодом, яка використовується для розробки інтерфейсів користувача на основі компонентів. Він був розроблений Facebook у 2013 році і зараз підтримується спільнотою з відкритим кодом і Facebook.

Це найпоширеніша бібліотека на основі JavaScript через легкість створення веб-додатків. Крім того, окрім створення інтерфейсу користувача, він пропонує кілька функцій, таких як Flux і React Native.

Однією з найбільших переваг використання React є те, що ви можете створювати як веб-, так і мобільні додатки, використовуючи ту саму кодову базу з фреймворком під назвою React Native. Джордан Волке спочатку заснував бібліотеку в 2011 році. Кожен компонент у додатку React служить будівельним блоком, що робить додаток придатним для повторного використання. Повторно

використовувані компоненти в JavaScript скорочують час і складність розробки.

Переваги використання React:

– багаторазові компоненти: під час розробки додатків React кількість складності та кодування менша через багаторазові компоненти в React (це принесло більше функціональності та чітку кодову базу, яку також легко підтримувати);
– продуктивність: React не залежить від традиційної DOM і використовує структуру JavaScript, віртуальну DOM (використання віртуального DOM покращує продуктивність і збільшує швидкість програм);
– крива навчання: React значно випереджає своїх конкурентів завдяки легкій кривій навчання, крім того, веб-розробникам, які щойно закінчили свої концепції в HTML і JavaScript, легко вивчити React.

Обмеження використання React:

– швидкість розробки: оскільки React є найпоширенішою бібліотекою JavaScript, React пропонує постійні оновлення, що спонукало розробників заново вивчати нові концепції та адаптуватися до темпу React;
– вступ до JSX: навчання React не вимагає жодних попередніх умов. Однак JSX не стане в нагоді багатьом розробникам під час вивчення розробки React. Окрім JSX, розробники скаржаться на функцію вбудованого сценарію в React, яка також є виснажливим завданням для розробників.

Vue – це прогресивний зовнішній фреймворк із відкритим вихідним кодом, який набув широкої популярності для створення односторінкових програм (SPA). SPA – це веб-програми, які мають лише один файл HTML, у якому відображається весь код. Він був випущений у 2014 році та привернув увагу розробників завдяки своєму простому інтерфейсу та легкому навчанню.

Вивчити Vue здавалося легшим, ніж більшість існуючих на той час технологій. Angular надихнув творців Vue відкрити Vue, заснований на об’єднанні всіх найкращих частин Angular під одним дахом і нехтуванні всіма

обмеженнями Angular для створення високоефективного інструменту. Vue зосереджується на розробниках-початківцях, допомагаючи їм створювати динамічні веб-програми, не потребуючи попереднього виснажливого навчання.

Переваги Vue.Js:

– коротка крива навчання: у React і Angular ви повинні володіти TypeScript і JavaScript відповідно, однак Vue зручний для початківців і не вимагає жодних попередніх навичок;
– розмір проєкту: розмір вашого проєкту Vue має кілька переваг, наприклад, безпосередньо впливає на SEO вашої веб-сторінки, оскільки пошукова консоль Google не показує важкі веб-сайти спереду, що потребує більше часу для завантаження;
– форуми та спільнота: форуми та інша підтримка спільноти дуже важливі, коли ви збираєтеся вибирати технологію (це допомагає підтримувати та вивчати будь-яку технологію в подальшій хорошій кривій).

Обмеження використання Vue.Js:

– екосистема: екосистема відіграє життєво важливу роль у адаптації програм до кількох браузерів і операційних систем, Vue має дуже вузьку екосистему, тому не відображається в старіших версіях операційних систем і веб-переглядачах;
– розробники: інші фреймворки, такі як Angular і React, підтримуються Google і Facebook, що автоматично створює довіру серед людей, однак Vue, як правило, не заслуговує довіри серед аудиторії.

У даній кваліфікаційній роботи прийнято рішення використовувати саме React framework для розробки програмного забезпечення тріангуляції полігональних областей через низьку криву навчання, простоту використання та величезну кількість інтеграцій з різноманітними нативними JavaScript бібліотеками.

3.2 Ініціалізація багатоплатформного React додатку

Існує 3 основних методи створення PWA React додатку:

– за допомогою NPX утиліти;
– створення власного service worker для забезпечення багатоплатформного використання;
– React PWA framework – фреймворк на базі React з глобальними налаштуваннями та додатками для PWA.

Створення власного service worker надає багато можливостей для тонкого налаштування кожного аспекту додатку як для мобільної версії так і для веб-версії додатку [7], що потребує великої кількості часу для ознайомлення, а React PWA framework останньої версії підтримує лише React 16.8.0, коли остання версія React framework на момент написання кваліфікаційної роботи – 18.2.0 Задля простоти експерименту та актуальності засобів розробки програмного забезпечення прийнято рішення використання NPX утиліту, що підтримує останні версії React фреймворку та надає готові service worker приклади.

NPX – інструмент, призначений для того, щоб допомогти стандартизувати досвід використання npm-пакетів: так само, як і npm упрощає установку й управління залежностями, розміщеними в реєстрі, npx упрощає використання CLI-утиліт та інших виконуваних файлів. Це значно упрощає ряд речей, які ми раніше робили за допомогою звичайного npm.

Ініціалізуємо проєкт за допомогою NPX у bash терміналі, використовуючи один із користувальницьких шаблонів PWA, отримаючи файл src/service-worker.js, який стане хорошою відправною точкою для роботи в автономному режимі: npx create-react-app my-app --template cra-template-pwa.

Отримаємо наступну структуру згенерованого додатку (див. рис. 3.1).

Встановимо необхідні залежності для додатку за допомогою пакетного менеджеру Yarn: yarn add delanuator color-convert [8].

Delanuator – неймовірно швидка та надійна бібліотека JavaScript для триангуляції Делоне двовимірних точок.

Color-convert – це бібліотека перетворення кольорів для JavaScript і node. Він перетворює всі способи між rgb, hsl, hsv, hwb, cmyk, ansi, ansi16, шістнадцятковими рядками та ключовими словами.

Рисунок 3.1 – Ініціалізований React додаток

React додаток вважається ініціалізованим та налаштованим для подальшої розробки.

3.3 Реалізація триангуляції полігональних областей

Створимо окремий файл для реалізації алгоритму триангуляції – Triangulation.js. Імпортуємо необхідні залежності (рис. 3.2).

```
import { React, useState, useEffect, useRef } from 'react';
import Delaunator from 'delaunator';
import colorConvertor from 'color-convert';
```

Рисунок 3.2 – Фрагмент коду ініціалізації залежностей

Визначимо ширину та висоту поля для візуалізації (див. рис. 3.3).

```
//global config
let canvasWidth = window.innerWidth;
let canvasHeight = window.innerHeight;
```

Рисунок 3.3 – Фрагмент коду налаштування поля візуалізації

Визначимо допоміжні функції задля калькуляції точок триангуляції, вершин триангуляції та кольорової схеми (рис.3.4).

```
const generateColorVariant = (hexcolor, tint, shade) => {
    //https://github.com/edelstone/tints-and-shades
    const rgb = colorConvertor.hex.rgb(hexcolor);
    let r = rgb[0]; r = r + ((255 - r) * tint); r = r * shade;
    let g = rgb[1]; g = g + ((255 - g) * tint); g = g * shade;
    let b = rgb[2]; b = b + ((255 - b) * tint); b = b * shade;
    return "#" + colorConvertor.rgb.hex(r, g, b);
}
const edgesOfTriangle = (i) => {
    return [3 * i, 3 * i + 1, 3 * i + 2];
}
```

Рисунок 3.4 – Фрагмента коду допоміжних функцій

Задля ініціалізації та первинного налаштування компоненту використаємо React-hook useEffect (рис. 3.5).

```
//executed only once - initialisation
useEffect(() => {
    init(true); //first time init
    var resizeTimer;

    //event - resize
    window.addEventListener('resize', () => {
        if(stretching) {
            renderFrame = false;
            if(resizeTimer) { clearTimeout(resizeTimer); }
            resizeTimer = setTimeout(handleResize, 100);
        } else {
            canvasHeight = window.innerHeight;
            canvasWidth = window.innerWidth;
            init(false);
        };
    });

    //cleanup
    return () => {
        cancelAnimationFrame(loop);
    }
}, []);
```

Рисунок 3.5 – Фрагмент коду реалізації React hook

Визначимо функцію генерації трикутників (рис.3.6).

```
let drawTriangle = (i, triangle) => {
    ctx.beginPath();
    ctx.moveTo(triangle[0][0], triangle[0][1]);
    ctx.lineTo(triangle[1][0], triangle[1][1]);
    ctx.lineTo(triangle[2][0], triangle[2][1]);
    ctx.closePath();

    ctx.fillStyle = generateColor(
        triangle[0][0],
        triangle[0][1],
        triangle[1][0],
        triangle[1][1],
        triangle[2][0],
        triangle[2][1]
    );

    ctx.shadowColor = triangleShadowColor;
    ctx.shadowBlur = triangleShadowBlur;
    ctx.fill();

    ctx.strokeStyle = lineColor;
    ctx.lineWidth = lineWidth;
    ctx.stroke();
};
```

Рисунок 3.6. – Фрагмент коду функції генерації трикутників

Визначимо функцію генерації трикутників(точок у просторі та їх вершин) за допомогою бібліотеки Delanuator (рис. 3.7).

```
let particlesForDelaunator = particles.map(particle => {
    return Object.values(particle);
});

//delaunay triangles generated
const delaunator = Delaunator.from(particlesForDelaunator);
```

Рисунок 3.7 – Фрагмент коду функції генерації трикутників

Створимо базовий HTML canvas для відображення триангуляції та визначимо HTML код як результат виконання Triangulation.js компоненту (див. рис. 3.8).

Застосуємо реалізований компонент Triangulation у root React компоненті index.js (див. рис. 3.9).

```
return(
    <div style = {{
        backgroundColor: `${fadeColor}`,
        position: 'absolute',
        zIndex: -1,
        width: "100%",
        height: "100%"
    }}>
        <canvas
            ref = { canvasRef }
            style = {{
                position: 'absolute',
                zIndex: -1,
                width: "100%",
                height: "100%"
            }}

            width = { canvasWidth }
            height = { canvasHeight }
        />
    </div>
);
```

Рисунок 3.8 – Фрагмент коду компоненту відображення канвасу

```
ReactDOM.render(
  <Triangulation
    //main colors
    topcolor = { "#221A33" }
    botcolor = { "#8A3D99" }

    //points settings
    pointscolor = { "#000000" }
    mincirclesize = { 3 }
    maxcirclesize = { 8 }
    count = { 100 }
    minspeed = { 0.1 }
    maxspeed = { 0.5 }
    pointshadowblur = { 0 }
    pointshadowcolor = { "#00000000" }

    //point color variation
    colorvariance = { false }
    tint = { 0.5 }
    shade = { 0.7 }


    //triangle settings
    triangleshadowblur = { 0 }
    triangleshadowcolor = { "#00000000" }
    linewidth = { 0.5 }
    linecolor = { "#00000000" }

    //config
    fps = { 60 }
    backgroundcolor = { "#00000000" }
    fadecolor = { "#221A33" }
    stretching = { false }
  />,
document.getElementById('root'));
```

Рисунок 3.9 – Фрагмент коду використання компоненту триангуляції

У результаті запуску React додатку за допомогою пакетного менеджеру Yarn, отримуємо динамічну веб-сторінку з візуалізацією автоматичної триангуляції полігональних областей(див. рис. 3.10 – 3.12).

Рисунок 3.10 – Візуалізація триангуляції полігональних областей

Рисунок 3.11 – Динаміка зображення триангуляції

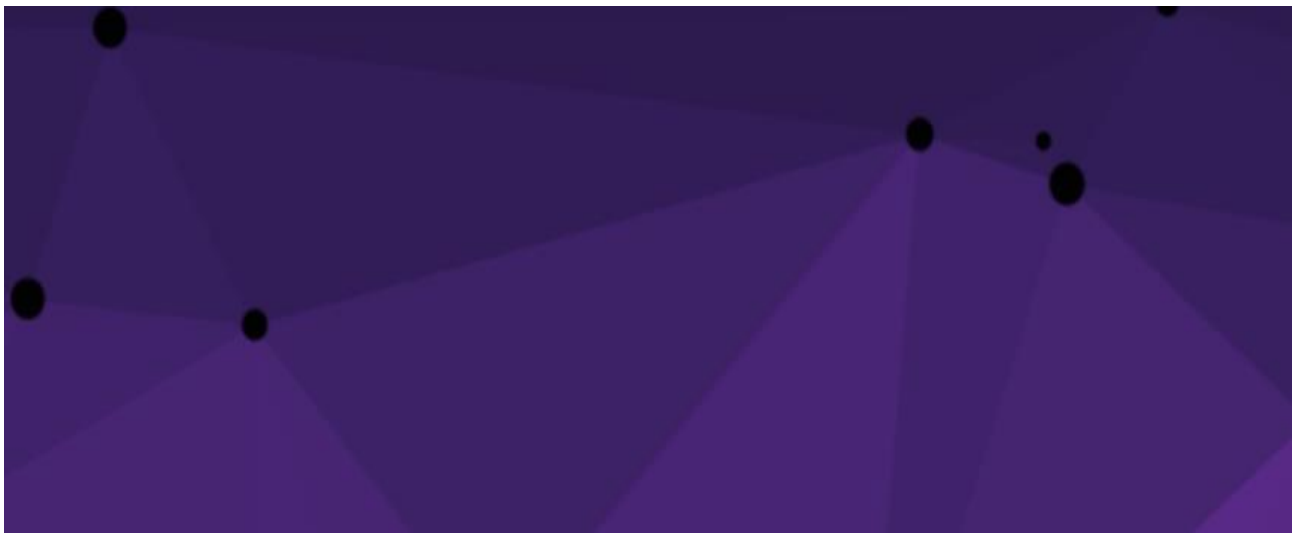

Рисунок 3.12 – Результат виконання додатку з проміжком в 1 хвилину

ВИСНОВКИ

В роботі було доведено актуальність проблеми триангуляції полігональних областей та розробки кросплатформового програмного забезпечення. Було розглянуто і проаналізовано основні сучасні методи розробки кросплатформних додатків та алгоритми реалізації віртуальної триангуляції.

В роботі було виконано порівняння різних типів додатків, виділено переваги та недоліки кожного з розглянутих типів. Виходячи з результатів порівняльного аналізу було прийнято рішення використовувати прогресивний веб додаток у якості прикладу кросплатформового програмного забезпечення.

Проведено порівняльний аналіз сучасних веб-фреймворків та мов програмування, прийнято рішення реалізувати практичне завдання мовою програмування JavaScript та фреймворком React. Визначено перелік допоміжних JavaScript бібліотек для вирішення задачі триангуляції. На основі обраних методологій і інструментів, враховуючи вимоги задачі триангуляції, було розроблено кросплатформне програмне забезпечення триангуляції полігональних областей.

Розроблена система виконує поставлені задачі і задовольняє поставленим вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1. Darcey L. Android Wireless Application Development : Android Essentials. Addison-Wesley Professional, 2012. 115 p.
2. Соловйов В. М., Сердюк О. А., Данильчук Г. Б. Моделювання складних систем. Київ : Видавництво О. Ю. Вовчок, 2016. 201 с.
3. Куленко М. Я. Основи графічного дизайну : підручник для студентів вищих навч. закладів. Київ : Кондор, 2007. 492 с.
4. Шмідт Я. Нова мережа: ознаки, практики і наслідки веб 2.0 : посібник для вузів. Київ : Академія Української Преси, 2013. 283 с.
5. Бернерс-Лі Заснування павутини : З чого починалася і до чого прийде Всесвітня мережа. Київ : Києво-Могилянська академія, 2007. 208 с.
6. Матвієнко, О.В. Internet-технології: проектування Web- сторінки : навчальний посібник для студентів вузів. Київ : ЦНЛ, 2004. 154 с.
7. Цеслів О.В. WEB-програмування : навч. посібнник. Київ : НТУУ “КПІ”, 2011. 296 с.
8. Козловський А.В., Паночишин Ю.М., Погріщук Б.В. Комп’ютерна техніка та інформаційні технології : навч. посіб. 2-ге вид. Київ : Знання, 2012. 205 с.

ДОДАТОК А

Лістинг коду компоненту «Triangulation.js»

```
import { React, useState, useEffect, useRef } from 'react';
import Delaunator from 'delaunator';
import colorConvertor from 'color-convert';

//global config
let canvasWidth = window.innerWidth;
let canvasHeight = window.innerHeight;

//helper functions
const valueToPercent = (percent, min, max) => {
    return min + percent * (max - min);
}
const generateColorVariant = (hexcolor, tint, shade) => {
    //https://github.com/edelstone/tints-and-shades
    const rgb = colorConvertor.hex.rgb(hexcolor);
    let r = rgb[0]; r = r + ((255 - r) * tint); r = r * shade;
    let g = rgb[1]; g = g + ((255 - g) * tint); g = g * shade;
    let b = rgb[2]; b = b + ((255 - b) * tint); b = b * shade;
    return "#" + colorConvertor.rgb.hex(r, g, b);
}
const edgesOfTriangle = (i) => {
    return [3 * i, 3 * i + 1, 3 * i + 2];
}
const pointsOfTriangle = (delauney, i) => {
    return edgesOfTriangle(i).map(e => delauney.triangles[e]);
}
```

```
const Triangulation = (props) => {
    //upper color
    const topColor = colorConvertor.hex.hsv((props.topcolor) ? props.topcolor : "#221A33");
    const topHue = topColor[0];
    const topSat = topColor[1];
    const topLig = topColor[2];

    //bottom color
    const botColor = colorConvertor.hex.hsv((props.botcolor) ? props.botcolor : "#8A3D99");
    const botHue = botColor[0];
    const botSat = botColor[1];
    const botLig = botColor[2];

    //points
    const pointsColor = (props.pointscolor) ? props.pointscolor : "#000000";
    const minCircleSize = (props.mincirclesize) ? props.mincirclesize : 3;
    const maxCircleSize = (props.maxcirclesize) ? props.maxcirclesize : 8;
    const colorVariance = (props.colorvariance) ? props.colorvariance : false;
    const tintFactor = (props.tint) ? props.tint : 0;
    const shadeFactor = (props.shade) ? props.shade : 1;

    //particles config
    const particleAmount = (props.count) ? props.count : 100;
    const particleMinSpeed = (props.minspeed) ? props.minspeed : 0.1;
    const particleMaxSpeed = (props.maxspeed) ? props.maxspeed : 0.5;
    const pointShadowBlur = (props.pointshadowblur) ? props.pointshadowblur : 3;
    const pointShadowColor = (props.pointshadowcolor) ? props.pointshadowcolor : "#000000";
```

```
    //global config
    const fps = (props.fps) ? props.fps : 60;
    const backgroundColor = (props.backgroundcolor) ? props.backgroundcolor : "#00000000";
    const lineWidth = (props.linewidth) ? props.linewidth : 0.0;
    const lineColor = (props.linecolor) ? props.linecolor : "#00000000";

    //triangle config
    const triangleShadowBlur = (props.triangleshadowblur) ? props.triangleshadowblur : 0;
    const triangleShadowColor = (props.triangleshadowcolor) ? props.triangleshadowcolor : "#00000000";

    //list of particles
    let [particles, setParticles] = useState([]);

    //canvas
    let canvasRef = useRef();   //canvas reference
    let canvas = null;          //canvas element
    let ctx = null;             //canvas context

    //fading and stretching
    const stretching = (props.stretching) ? props.stretching : false;
    const fadeColor = (props.fadecolor) ? props.fadecolor : "#221A33";
    var renderFrame = true;
    let opacity = 0;
    let delta = 0.05; //fadespeed

    const fadeIn = () => {
        if(opacity < 1) {
            ctx.globalAlpha = opacity;
            opacity += delta;
        }
    }
```

```
const handleResize = () => {
    canvasHeight = window.innerHeight;
    canvasWidth = window.innerWidth;

    renderFrame = true;
    init(true);
}

//executed only once - initialisation
useEffect(() => {
    init(true); //first time init
    var resizeTimer;

    //event - resize
    window.addEventListener('resize', () => {
        if(stretching) {
            renderFrame = false;
            if(resizeTimer) { clearTimeout(resizeTimer); }
            resizeTimer = setTimeout(handleResize, 100);
        } else {
            canvasHeight = window.innerHeight;
            canvasWidth = window.innerWidth;
            init(false);
        };
    });

    //cleanup
    return () => {
        cancelAnimationFrame(loop);
    }
}, []);
```

```
const init = (requestAnimationForFirstTime) => {
    canvas = canvasRef.current;
    ctx = canvas.getContext('2d');

    opacity = 0;
    ctx.globalAlpha = opacity;
    const particlesLocal = [];

    //particles on the edges
    particlesLocal.push({ x: 0, y: 0, velocityX: 0, velocityY: 0, circleSize: 0});
    particlesLocal.push({ x: 0, y: canvasHeight, velocityX: 0, velocityY: 0, circleSize: 0 });
    particlesLocal.push({ x: canvasWidth, y: 0, velocityX: 0, velocityY: 0, circleSize: 0 });
    particlesLocal.push({ x: canvasWidth, y: canvasHeight, velocityX: 0, velocityY: 0, circleSize: 0 });

    for(let i = 0; i < particleAmount; i++) {
        addParticle(
            Math.random() * canvasWidth,
            Math.random() * canvasHeight,
            particlesLocal
        );
    }

    particles = particlesLocal;
    setParticles(particlesLocal);

    if(requestAnimationForFirstTime) {
        setTimeout(() => {
            requestAnimationFrame(loop);
        }, 1000 / fps);
    };
}
```

```
 const loop = () => {
     fadeIn();

     //TODO save and restore optimisation?
     ctx.save();
     ctx.fillStyle = backgroundColor;
     ctx.fillRect(0, 0, canvasWidth, canvasHeight);
     ctx.restore();

     //updating all particles
     let updatedParticles = particles.map((particle, i) => {
         let x = particle.x;
         let y = particle.y;
         let velocityX = particle.velocityX;
         let velocityY = particle.velocityY;

         x += velocityX;
         y += velocityY;

         //X axis bounce
         if(x < 0) {
             x = 0;
             if(velocityX < 0) { velocityX *= -1; }
         } else if(x > canvasWidth) {
             x = canvasWidth;
             if(velocityX > 0) { velocityX *= -1; }
         }
```

```
        //Y axis bounce
        if(y < 0) {
            y = 0;
            if(velocityY < 0) { velocityY *= -1; }
        } else if(y > canvasHeight) {
            y = canvasHeight;
            if(velocityY > 0) { velocityY *= -1; }
        }

        //update the particle
        particle.x = x;
        particle.y = y;
        particle.velocityX = velocityX;
        particle.velocityY = velocityY;
    });

    //transforming particles to array that delaunator can accept - Object array wrapper
    let particlesForDelaunator = particles.map(particle => {
        return Object.values(particle);
    });

    //delaunay triangles generated
```

```
    //delaunay triangles generated
    const delaunator = Delaunator.from(particlesForDelaunator);

    //draw triangle on canvas
    let drawTriangle = (i, triangle) => {
        ctx.beginPath();
        ctx.moveTo(triangle[0][0], triangle[0][1]);
        ctx.lineTo(triangle[1][0], triangle[1][1]);
        ctx.lineTo(triangle[2][0], triangle[2][1]);
        ctx.closePath();

        ctx.fillStyle = generateColor(
            triangle[0][0],
            triangle[0][1],
            triangle[1][0],
            triangle[1][1],
            triangle[2][0],
            triangle[2][1]
        );

        ctx.shadowColor = triangleShadowColor;
        ctx.shadowBlur = triangleShadowBlur;
        ctx.fill();

        ctx.strokeStyle = lineColor;
        ctx.lineWidth = lineWidth;
        ctx.stroke();
    };

    for(let i = 0; i < delaunator.triangles.length / 3; i++) {
        let triangle = pointsOfTriangle(delaunator, i).map(e => particlesForDelaunator[e]);
        drawTriangle(i, triangle);
    }
```

```
    ctx.save();
    //skipping first 4 corner particles
    for(let i = 4; i < particles.length; i++) {
        ctx.fillStyle = particles[i].circleColor;
        ctx.shadowColor = pointShadowColor;
        ctx.shadowBlur = pointShadowBlur;
        ctx.beginPath();
        ctx.arc(particles[i].x, particles[i].y, particles[i].circleSize, 0, Math.PI * 2);
        ctx.closePath();
        ctx.fill();
    };
    ctx.restore();

    if(renderFrame) {
        setTimeout(() => {
            requestAnimationFrame(loop);
        }, 1000 / fps);
    };
```

```
const generateColor = (ax, ay, bx, by, cx, cy) => {
    //triangle center coords
    //const triangleCenterX = (ax + bx + cx) / 3;
    const triangleCenterY = (ay + by + cy) / 3;
    const percentDistanceFromFloor = triangleCenterY / canvasHeight;

    //hue saturation lightness
    const hue = valueToPercent(percentDistanceFromFloor, topHue, botHue);
    const sat = valueToPercent(percentDistanceFromFloor, topSat, botSat);
    const lig = valueToPercent(percentDistanceFromFloor, topLig, botLig);

    return "#" + colorConvertor.hsl.hex(hue, sat, lig);

    // //!!! LEGACY CODE !!!//
    // variations of color calculation:
    // 1) based on triangle area
    // const area = Math.abs(0.5 * (ax*(by - cy) + bx*(cy - ay) + cx*(ay - by)));
    //
    // 2) based on distance from a point (ex. distance from center)
    // const distanceFromCenter = Math.sqrt(
    //     Math.pow(triangleCenterX - canvasWidth / 2, 2) +
    //     Math.pow(triangleCenterY - canvasHeight / 2, 2)
    // );
};
```

```
  return(
      <div style = {{
          backgroundColor: `${fadeColor}`,
          position: 'absolute',
          zIndex: -1,
          width: "100%",
          height: "100%"
      }}>
```

```
            <canvas
                ref = { canvasRef }
                style = {{
                    position: 'absolute',
                    zIndex: -1,
                    width: "100%",
                    height: "100%"
                }}

                width = { canvasWidth }
                height = { canvasHeight }
            />
        </div>
    );
}

export default Triangulation;
```