МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА РОЗПОДІЛЕНОЇ СИСТЕМИ ДЛЯ ВІЗУАЛІЗАЦІЇ ПОЛІГОНАЛЬНИХ МОДЕЛЕЙ»**

Виконав: студент 2 курсу, групи 8.1218
спеціальності 121 інженерія програмного забезпечення
(шифр і назв1а спеціальності)
освітньої програми інженерія програмного забезпечення
Д.О. Кисельов
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н.Чопоров С.В
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

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

_____ Лісняк А.О.

(підпис)

« 29 » 05 2019 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кисельову Дмитру Олександровичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка розподіленої системи для візуалізації полігональних моделей

керівник роботи (проекту) Чопоров Сергій Вікторович, к.т.н., доцент

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 29 » 05 2019 року № 811-с

2. Строк подання студентом роботи _____

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.Виконання практичного завдання

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	30.05.2019	
2.	Збір вихідних даних.	20.06.2019	
3.	Обробка методичних та теоретичних	06.07.2019	
	джерел.		
4.	Розробка першого та другого розділу.	03.10.2019	
5.	Розробка третього та четвертого розділу.	05.11.2019	
6.	Оформлення та нормоконтроль	19.12.2019	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.01.2020	

Студент _____ Д.О. Кисельов
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка розподіленої системи візуалізації полігональних моделей»: 52 с., 15 рис., 41 джерел, 1 додаток.

АСИНХРОННЕ ПРОГРАМУВАННЯ, ПОЛІГОНАЛЬНІ МОДЕЛІ, РОЗПОДІЛЕНІ СИСТЕМИ, STL, PYTHON.

Об’єкт дослідження: розподілені обчислювальні системи, методи зберігання та візуалізації полігональних моделей.

Мета роботи: розробити реалізацію розподіленої системи для візуалізації полігональних моделей.

Методи дослідження: експериментально-теоретичний.

В кваліфікаційній роботі розглядаються основні методи, підходи та практики роботи розподілених систем обчислення, створення тривимірних моделей та їх використання у веб. Описано способи представлення полігональних моделей.

Результати можуть бути використані для розробки обчислювальних систем різного ступеню складності.

SUMMARY

Master’s Qualifying Thesis «Development of a Distributed System for Polygonal Models Visualization»: 52 pages, 15 figures, 41 references, 1 supplements.

ASYNCHRONOUS PROGRAMMING, POLYGONAL MODELS, DISTRIBUTED SYSTEMS, STL, PYTHON.

The object of the study is distributed computing systems, methods of storage and visualization of polygonal models.

The aim of the study is to develop the implementation of a distributed system for visualization of polygonal models.

The method of research is experimental-theoretical.

The qualification work discusses the basic methods, approaches and practices of distributed computing systems, creation of 3D models and its usage in the Web. Described methods of representation of polygonal models.

The results can be used to develop computing systems of varying degrees of complexity.

ЗМІСТ

ВСТУП

Розподілені обчислення – це галузь інформатики, яка вивчає розподілені системи. Розподілена система – це система, компоненти якої розміщені на різних мережевих комп'ютерах, які передають і координують свої дії, передаючи повідомлення один одному. Компоненти взаємодіють один з одним для досягнення спільної мети. Три суттєві характеристики розподілених систем: сумісність компонентів, відсутність глобального годинника та незалежний збій компонентів [1].

Комп'ютерна програма, яка працює в розподіленій системі, називається розподіленою програмою (а розподілене програмування – це процес написання таких програм). Існує багато різних типів реалізацій для механізму передачі повідомлень, включаючи чисті HTTP, RPC-подібні роз'єми та черги повідомлень [2].

Розподіл навантаження на таких складних системах є викликом для архітекторів програмного забезпечення. В цьому полягає актуальність даної роботи.

Таким чином, метою роботи стає розробка розподіленої системи для візуалізації полігональних моделей. Для виконання роботи поставлені задачі:

– ознайомитися з основними принципами побудови розподілених систем обчислення;

– дослідити методи візуалізації полігональних моделей;

– розглянути існуючі інструменти візуалізації тривимірних моделей;

– виконати практичне завдання.

1 РОЗПОДІЛЕНІ ІНФОРМАЦІЙНІ СИСТЕМИ

1.1 Архітектура розподілених систем

Для розподілених обчислень використовуються різні архітектури апаратури та програмного забезпечення. На нижчому рівні необхідно з'єднати декілька процесорів з якоюсь мережею, незалежно від того, розміщена ця мережа на друкованій платі або складається з нещільно пов'язаних пристроїв та кабелів. На більш високому рівні необхідно з'єднати процеси, що працюють на цих процесорах, з якоюсь системою зв'язку [1].

Програмування розподілених систем зазвичай належить до однієї з декількох базових архітектур: клієнт-сервер, трирівнева, n-ярусна або однорангова; або категорії: нещільний або щільний зв’язок [2].

1.1.1 Архітектура клієнт-сервер

Архітектура, де розумні клієнти звертаються до сервера для отримання даних, а потім відформатують і відображають їх користувачам [3].

Модель клієнт-сервер – це розподілена структура додатків, яка розбиває завдання або навантаження між постачальниками ресурсу чи послуги, що називаються серверами, і запитувачами послуг, що називаються клієнтами [4]. Часто клієнти та сервери спілкуються через комп'ютерну мережу на окремому апаратному забезпеченні, але і клієнт, і сервер можуть знаходитися в одній системі. Хост сервера запускає одну або декілька серверних програм, які діляться своїми ресурсами з клієнтами. Клієнт не ділиться жодними своїми ресурсами, але він запитує контент або послугу з сервера. Тому клієнти ініціюють сеанси зв'язку із серверами, які очікують на вхідні запити. Прикладами комп’ютерних програм, які використовують модель клієнт-сервер, є електронна пошта, мережева друк та всесвітня павутина [5].

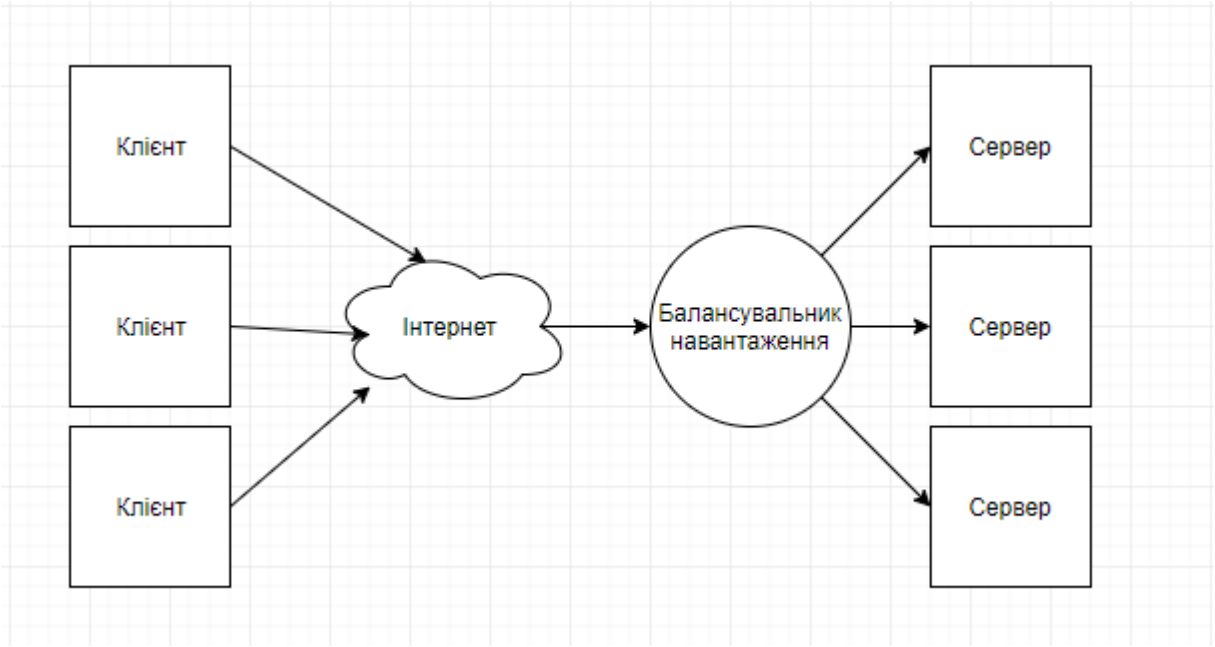

Рисунок 1.1 – Схематичне зображення архітектури клієнт-сервер

Загалом, сервіс – це абстрагування комп’ютерних ресурсів, і клієнту не потрібно перейматися тим, як працює сервер під час виконання запиту та надання відповіді. Клієнт повинен лише зрозуміти відповідь на основі відомого протоколу програми, тобто вмісту та форматування даних для запитуваного сервісу.

Клієнти та сервери обмінюються повідомленнями за схемою запит-відповідь. Клієнт надсилає запит, а сервер повертає відповідь. Цей обмін повідомленнями є прикладом міжпроцесорної комунікації. Для спілкування комп'ютери повинні мати загальну мову, і вони повинні дотримуватися правил, щоб і клієнт, і сервер знали, чого очікувати. Мова та правила спілкування визначені у протоколі зв'язку. Всі протоколи клієнт-сервер працюють на рівні додатків. Протокол рівня додатків визначає основні шаблони діалогу. Щоб ще більше формалізувати обмін даними, сервер може впровадити інтерфейс програмування додатків (API) [6]. API – це рівень абстракції для доступу до послуги. Обмежуючи спілкування певним форматом контенту, полегшується аналіз. Абстрагуючи доступ, полегшується обмін даними між платформами [7].

Сервер може отримувати запити від багатьох різних клієнтів за короткий проміжок часу. Комп'ютер може виконувати лише обмежену кількість завдань у

будь-який момент та покладається на систему планування для визначення пріоритетності вхідних запитів клієнтів для їх розміщення. Для запобігання зловживань та максимальної доступності серверне програмне забезпечення може обмежувати доступність клієнтів.

1.1.2 Багаторівнева архітектура

Архітектура клієнт-сервер, в якій функції представлення, обробки додатків та управління даними фізично розділені. Найпоширенішим використанням багаторівневої архітектури є трирівнева архітектура.

N-ярусна архітектура додатків забезпечує модель, за допомогою якої розробники можуть створювати гнучкі та багаторазові програми. Розділяючи додаток на рівні, розробники отримують можливість зміни або додавання певного шару, замість того, щоб переробляти весь додаток. Трирівнева архітектура, як правило, складається з рівня представлення, рівня бізнес логіки та рівня зберігання даних.

Хоча поняття шару та ярусу часто використовуються взаємозамінно, одна досить поширена точка зору полягає в тому, що дійсно є різниця. Ця думка стверджує, що шар – це логічний механізм структурування елементів, що складають програмне рішення, а рівень – фізичний механізм структурування системної інфраструктури. Наприклад, тришарове рішення легко може бути розгорнуте на одному рівні, наприклад, на персональній робочій станції [8].

У логічній багатошаровій архітектурі для інформаційної системи з об'єктно-орієнтованим дизайном найбільш поширені наступні чотири рівні:

- рівень представлення (рівень інтерфейсу користувача);
- прикладний рівень (сервісний рівень або рівень контролера);
- рівень бізнес логіки;
- рівень доступу до даних.

Трирівнева архітектура – це структура архітектури програмного забезпечення клієнт-сервер, в якій інтерфейс користувача (представлення),

функціональна бізнес логіка, зберігання даних у комп'ютері та доступ до даних розробляються та підтримуються як незалежні модулі, найчастіше на окремих платформах [9]. Він був розроблений Джоном Дж. Донованом в корпорації Open Environment Corporation (OEC), інструментальній компанії, яку він заснував у Кембриджі, штат Массачусетс.

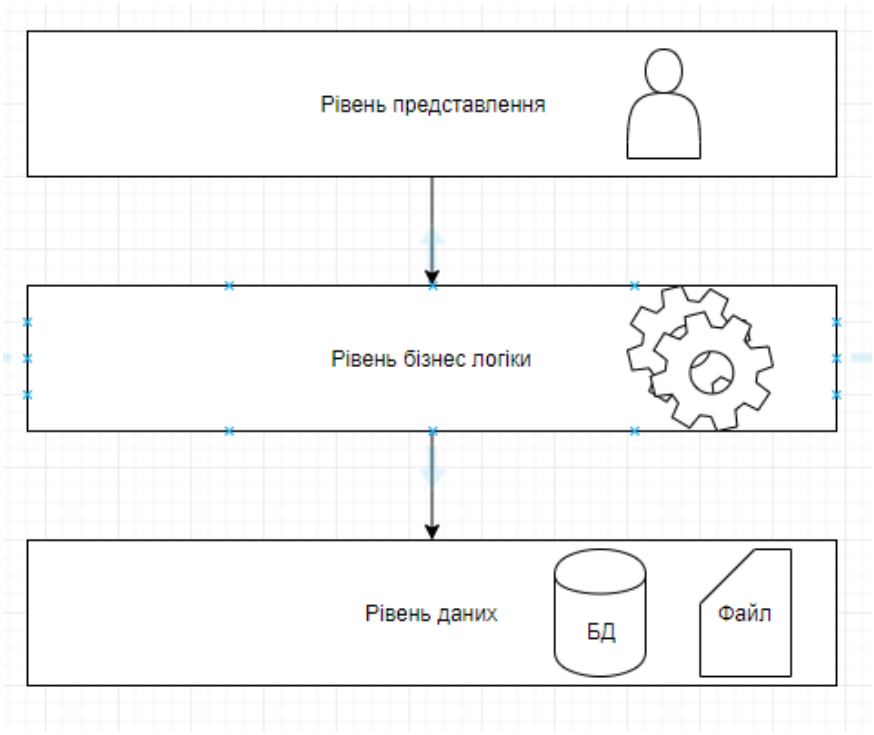

Рисунок 1.2 – Трирівнева архітектура

Крім звичайних переваг модульного програмного забезпечення з чітко визначеними інтерфейсами, трирівнева архітектура призначена для того, щоб дозволити модернізацію або заміну будь-якого з трьох ярусів у відповідь на зміни вимог чи технології. Наприклад, зміна операційної системи в ярусі презентації вплине лише на код інтерфейсу користувача.

Зазвичай інтерфейс користувача працює на настільному ПК або робочій станції і використовує стандартний графічний інтерфейс користувача, функціональну логіку процесу, яка може складатися з одного або декількох окремих модулів, що працюють на робочій станції або сервері прикладних програм, і БД на сервері баз даних або мейнфреймі, що містить логіку

зберігання даних комп'ютера. Середній рівень може бути багаторівневим (у цьому випадку загальна архітектура називається "n-ярусна архітектура").

Рівень представлення – це найвищий рівень програми. Він спілкується з іншими рівнями, за допомогою яких виводить результати на рівень браузера / клієнта та всі інші рівні в мережі. Простіше кажучи, це рівень, до якого користувачі можуть отримати доступ безпосередньо (наприклад, веб-сторінка або графічний інтерфейс операційної системи).

Рівень бізнес логіки – контролює функціональність програми, здійснюючи обробку даних.

Рівень даних – включає механізми збереження даних (сервери баз даних, спільні файли тощо) та рівень доступу до даних, який інкапсулює механізми збереження та отримання даних. Рівень доступу до даних повинен надавати API до рівня додатків, який розкриває методи управління збереженими даними, створюючи залежностей від механізмів зберігання даних. Уникнення залежностей від механізмів зберігання дозволяє здійснювати зміни, не змінюючи клієнтів рівня додатків.

1.1.2 Архітектура Peer-to-peer

Архітектури, де немає спеціальних машин, які надають послугу або керують мережевими ресурсами. Натомість усі обов'язки рівномірно розподіляються між усіма машинами, відомими як піри. Вони можуть виступати як в якості клієнтів, так і як сервери. Приклади такої архітектури включають BitTorrent та мережу біткойн [10].

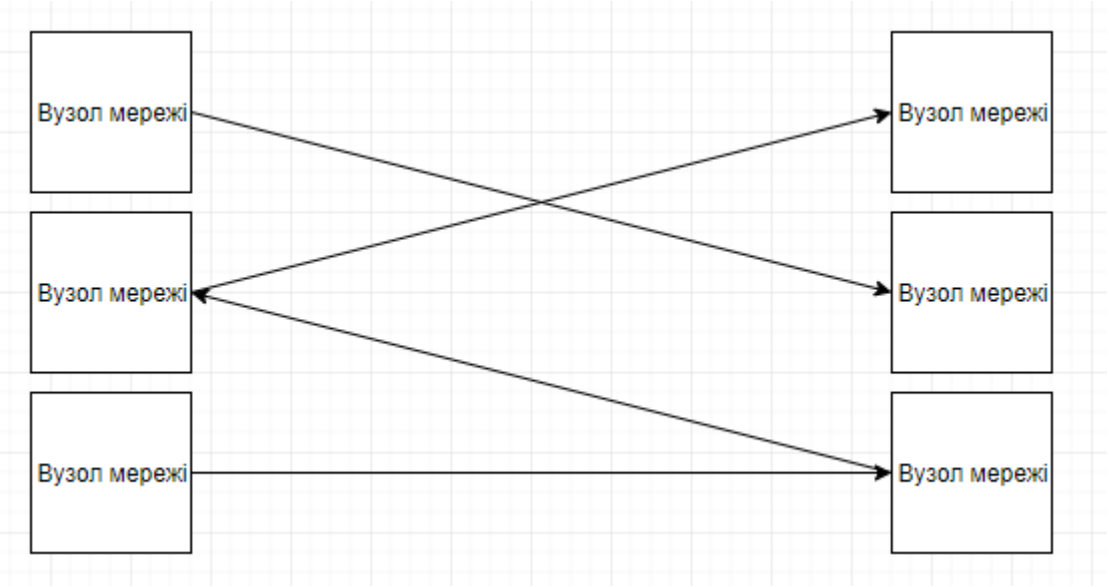

Рисунок 1.3 – Однорангова мережа

Вузли мережі (піри) роблять частину своїх ресурсів, таких як ресурси процесора, диск або пропускна здатність мережі, безпосередньо доступними для інших учасників мережі, без необхідності центральної координації з боку серверів [11]. Вони є одночасно постачальниками та споживачами ресурсів, на відміну від традиційної моделі клієнт-сервер, в якій споживання та пропозиція ресурсів розділені.

Однорангова мережа створена навколо поняття рівних однорангових вузлів, які одночасно функціонують як "клієнти", так і "сервери" для інших вузлів мережі. Ця модель розташування мережі відрізняється від моделі клієнт-сервер, де спілкування зазвичай відбувається з центральним сервером і з ним. Типовим прикладом передачі файлів, що використовує модель клієнт-сервер, є послуга протоколу передачі файлів (FTP), в якій клієнт і серверні програми відрізняються: клієнти ініціюють передачу, а сервери задовольняють ці запити.

Однорангові мережі, як правило, реалізують певну форму віртуальної мережі, що накладається поверх топології фізичної мережі, де вузли утворюють підмножину вузлів фізичної мережі. Даними все ще обмінюються безпосередньо через базову мережу TCP/IP, але на рівні додатків піри можуть спілкуватися один з одним безпосередньо. Виходячи з того, як вузли пов’язані між собою в межах накладеної мережі та як ресурси індексуються та розміщені,

ми можемо класифікувати мережі як неструктуровані або структуровані (або як гібрид між цими двома) [12].

1.2 Порівняння архітектури клієнт-сервер та peer-to-peer

У моделі клієнт-сервер сервер часто призначений для роботи як централізована система, яка обслуговує багатьох клієнтів. Вимоги до обчислювальної потужності, пам’яті та пам’яті для сервера повинні бути відповідними до очікуваного робочого навантаження (тобто кількості клієнтів, що підключаються одночасно). Системи балансування навантаження та відмовлення часто використовуються для масштабування реалізації сервера.

У мережі однорангових два або більше комп'ютерів об'єднують свої ресурси та спілкуються в децентралізованій системі. Піри – це рівносильні, або рівнопотужні вузли в неієрархічній мережі. На відміну від клієнтів у мережі клієнт-сервер або клієнт-черга-клієнт, піри спілкуються один з одним безпосередньо. У одноранговій мережі, алгоритм протоколу зв'язку одноранговий зв’язок врівноважує навантаження і навіть вузли мережі зі скромними ресурсами можуть допомогти поділити навантаження. Якщо вузол стає недоступним, його спільні ресурси залишаються доступними до тих пір, поки пропонують його інші вузли мережі. В ідеалі вузлу мережі не потрібно досягати високої доступності, оскільки інші вузли компенсують будь-який час простою ресурсів. В міру зміни доступності та завантаженості вузлів протокол перенаправляє запити.

1.3 Розподіленні бази даних

Розподілена база даних, яка називається іноді паралельна база даних або в англійській абревіатурі, DDB – це єдина база даних, а не довільний набір файлів, індивідуально збережених на різних вузлах мережі і є розподіленою файловою системою. Дані являють собою DDB, тільки якщо вони пов'язані відповідно до деякого структурного формалізмому, реляційною моделлю, а доступ до них забезпечується єдиним високорівневим інтерфейсом [13].

До переваг розподілених баз даних відносяться:

– відповідність структури розподілених баз даних структурі організацій;

– гнучка взаємодія локальних БД;

– широкі можливості централізації вузлів;

– безпосередній доступ до інформації, зниження вартості передач (за рахунок ущільнення і концентрації даних);

– високі системні характеристики (малий час відгуку за рахунок розпаралелювання процесів, висока надійність);

– модульна реалізація взаємодії, розширення апаратних засобів, можливість використання об'єктно-орієнтованого підходу в програмуванні;

– можливість розподілу файлів відповідно до їх активністю;

– незалежні розробки локальних БД через стандартний інтерфейс.

Разом з тим розподілені бази даних мають більш складну структуру, що викликає появу додаткових проблем (надмірність, неузгодженість даних за часом, узгодження процесів оновлення і запитів, використання телекомунікаційних ресурсів, облік роботи додатково під'єднаних локальних БД, стандартизація загального інтерфейсу) узгодження роботи елементів.

Серйозні проблеми виникають при інтеграції в рамках розподілених баз даних однорідних локальних БД з однаковими, найчастіше реляційними, моделями даних.

Проблеми значно ускладнюються, якщо локальні БД побудовані з використанням різних моделей даних (неоднорідні бази даних).

Система управління розподіленою базою даних складається з (можливо, порожнього) набору вузлів прийому запитів і набору вузлів збереження даних. Вузли прийому запитів реалізують прозорий інтерфейс доступу до даних, що зберігаються в вузлах збереження даних, та приховують фрагментацію даних між вузлами. Кожен з вузлів може бути представлений незалежним комп'ютером в комп'ютерній мережі з власною (можливо відмінною від інших вузлів) операційною системою.

Система управління розподіленою базою даних є однорідною (гомогенною), якщо на кожному з вузлів збереження даних застосовуються однакові СКБД, в іншому випадку система управління розподіленою базою даних є неоднорідною (гетерогенною). Внаслідок застосування стандартизованих механізмів доступу до баз даних відмінності між однорідними та неоднорідними системами нівелюються і не є критичними [14].

Різновиди архітектури розподілених баз даних:

– однорангова мережа – всі комп'ютери мережі є серверами та на кожному комп'ютері розміщено розподілену СКБД і базу даних та з кожного комп'ютера можна надіслати до іншого запит на отримання необхідних даних;

– з багатьма незалежними серверами – існують багато серверів, що мають доступ до своїх локальних баз даних. У такому випадку на комп'ютерах клієнтів має зберігатись інформація стосовно того, які дані на яких серверах розташовані, а також має бути розміщене програмне забезпечення, що дає змогу розкласти запити для їхнього виконання на різних серверах і потім об'єднати результати;

– взаємодіючі сервери – кожен сервер містить повну інформацію стосовно того, які дані на яких серверах зберігаються, а також здатний обробляти розподілені запити. У разі потреби один сервер може звернутися до іншого для одержання необхідних даних;

– клієнт-сервер – передбачено наявність єдиного комп'ютера-сервера і багатьох комп'ютерів-клієнтів, що взаємодіють між собою через канали зв'язку. На сервері розташована СКБД та інтегрована (централізована) база даних. Ніякого розподілу баз даних за вузлами мережі немає.

Розподіл даних між вузлами збереження даних забезпечується на основі механізмів фрагментації та реплікації, і досягається шляхом вертикального (на окремі поля записів бази даних) або горизонтального (на окремі записи бази даних) поділу даних.

Реплікація – це механізм розподілу даних за вузлами, що дозволяє зберігати копії тих самих даних на різних вузлах мережі з метою прискорення пошуку і підвищення стійкості до відмов. Відношення чи фрагмент є реплікованим, якщо його копії зберігаються на двох або більше вузлах (копії ще називають репліками) [15].

Для реалізації реплікації використовуються три сервери: видавець, дистриб’ютор і передплатник.

Видавець-сервер, що надає розміщені на ньому дані для копіювання на інші сервери. Окрім створення копії даних, видавець відстежує внесені до його бази даних зміни і готує нову копію.

Дистриб’ютор (посередник) – сервер, що підтримує розподілену базу даних. Він виконує роль посередника, копіює всі публікації, підготовлені видавцем, і пересилає їх передплатникам. Дистриб’ютором може бути виділений сервер або сервер, сконфігурований як видавець чи передплатник. Конкретні функції, що їх виконує дистриб’ютор, залежать від методів реплікації.

Передплатник – сервер, що отримує копії даних, надані видавцем. Механізми зміни даних передплатником відрізняються від механізмів зміни даних видавцем [15].

Існують два методи відновлення даних передплатників:

– реплікація за запитом полягає в тому, що передплатник періодично звертається до дистриб'ютора із запитом про зміни, що відбулися з моменту останнього з'єднання.

– примусова реплікація, в свою чергу, коли дистриб'ютор сам встановлює з'єднання з передплатником і пересилає йому необхідні дані.

Залежно від методу реплікації, передплатники можуть, або не можуть вносити зміни в репліковані дані. У найпростішому випадку змінювати дані може тільки видавець, у складніших моделях реплікації – передплатники і видавці. Змінені дані, отримані від усіх передплатників, синхронізуються і поєднуються з даними видавця, а потім розсилаються передплатникам.

Моделі реплікації

– реплікація моментальних знімків;

– реплікація транзакцій [14].

Властивості розподілених баз даних:

– локальна автономія – управління даними на кожному з вузлів розподіленої системи виконується локально;

– незалежність вузлів – всі вузли рівноправні і незалежні, а розташовані на них БД є рівноправними постачальниками даних в загальний простір даних;

– безперервні операції – можливість безперервного доступу до даних в рамках розподіленої БД незалежно від їх розташування і незалежно від операцій, що виконуються на локальних вузлах;

– прозорість розташування – користувач, що звертається до БД, нічого не повинен знати про реальне, фізичне розміщення даних у вузлах інформаційної системи;

– прозора фрагментація – можливість розподіленого (тобто на різних вузлах) розміщення даних, логічно поєднаних в єдине ціле. Існує фрагментація двох типів: горизонтальна і вертикальна;

– прозоре тиражування – тиражування даних – це асинхронний процес перенесення змін об'єктів вихідної бази даних в бази, розташовані на інших вузлах розподіленої системи;

– обробка розподілених запитів – можливість виконання операцій вибірки даних з розподіленої БД, за допомогою запитів, сформульованих на мові SQL;

– обробка розподілених транзакцій – можливість виконання операцій оновлення розподіленої бази даних, які не порушують цілісність і узгодженість даних [16];

– незалежність від устаткування – як вузли розподіленої системи можуть виступати ПК будь-яких моделей і виробникі;

– незалежність від операційних систем – різноманіття операційних систем, керуючих вузлами розподіленої системи;

– прозорість мережі – спектр підтримуваних конкретною СУБД мережевих протоколів не має бути обмеженням системи, заснованої на розподіленій БД;

– незалежність від баз даних – в розподіленій системі можуть працювати СУБД різних виробників, і можливі операції пошуку і оновлення в базах даних різних моделей і форматів [13].

2 ТРИВИМІРНА ГРАФІКА

Тривимірна графіка – розділ комп'ютерної графіки, присвячений методам створення зображень або відео шляхом моделювання об'ємних об'єктів в тривимірному просторі [17].

3D-моделювання – це процес створення тривимірної моделі об'єкта. Завдання 3D-моделювання – розробити зоровий об'ємний образ бажаного об'єкта. При цьому модель може як відповідати об'єктам з реального світу (колесо, шестерня, літак), так і бути повністю абстрактною (проекція трикутника).

Історія комп’ютерної анімації почалася ще в 40-х та 1950-х роках, коли люди почали експериментувати з комп’ютерною графікою [18].

Вільяму Феттеру було приписано в 1961 році термін комп'ютерна графіка для опису його роботи в Boeing [18].

2.1 Створення тривимірної моделі

Об'єкти в 3D-комп'ютерній графіці часто називають 3D-моделями. На відміну від відтвореного зображення, дані моделі містяться у графічному файлі даних. 3D-модель – це математичний опис будь-якого тривимірного об'єкта; модель технічно не є графічною, поки вона не відображається. Модель може відображатися візуально як двовимірне зображення через процес, який називається 3D-відтворенням, або вона може бути використана у не графічних комп'ютерних моделюваннях та обчисленнях [19].

Для отримання тривимірного зображення на поверхні необхідні наступні кроки:

– моделювання;

– текстурування;

– освітлення;

– анімація;

– візуалізація [20].

Моделювання сцени включає в себе декілька категорій об'єктів: геометрія, матеріали, джерела світла, камери.

Задача тривимірного моделювання – описати ці об'єкти і розмістити їх в сцені з допомогою геометричних перетворень відповідно до вимог до зображення [21].

Текстурування передбачає проектування растрових або процедурних текстур на поверхні тривимірного об'єкту відповідно до карти UV-координат, де кожній вершині об'єкта ставиться у відповідність певна координата на двомірному просторі текстури.

Освітлення полягає в створенні, напрямку і налаштування віртуальних джерел світла. При цьому в віртуальному світі джерела світла можуть мати негативну інтенсивність, відбираючи світло із зони свого «негативного висвітлення» [19].

Одне з головних покликань тривимірної графіки – надання руху (анімація) тривимірної моделі, або імітація руху серед тривимірних об'єктів. Універсальні пакети тривимірної графіки мають досить багаті можливості по створенню анімації. Існують також вузькоспеціалізовані програми, створені суто для анімації і володіють дуже обмеженим набором інструментів моделювання [20].

На етапі відтворення, математична (векторна) просторова модель перетворюється в плоску (растрову) картинку. Якщо потрібно створити фільм, то відображається послідовність таких картинок-кадрів. Як структура даних, зображення на екрані представлено матрицею точок, де кожна точка визначена, що як найменше, трьома числами: інтенсивністю червоного, синього і зеленого кольору. Таким чином, відтворення перетворює тривимірну векторну структуру даних в плоску матрицю пікселів. Цей крок часто вимагає дуже складних обчислень, особливо якщо потрібно створити ілюзію реальності.

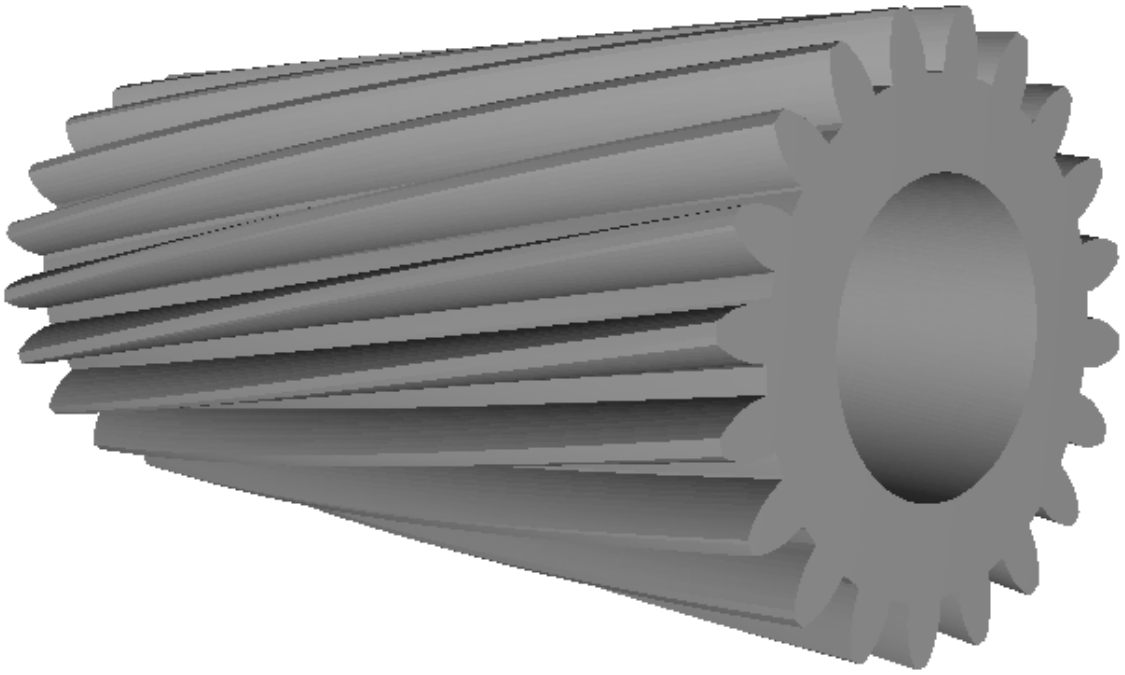

Рисунок 2.1 – Відтворення тривимірної моделі

Найпростіший вид відтворення – це побудувати контури моделей на екрані комп'ютера за допомогою проекції, як показано вище. Зазвичай цього недостатньо, і потрібно створити ілюзію матеріалів, з яких виготовлені об'єкти, а також розрахувати спотворення цих об'єктів за рахунок прозорих середовищ [19].

2.2 Тривимірна графіка у веб

Традиційно підтримка тривимірної комп’ютерної графіки обмежувалася високопродуктивними комп'ютерами або спеціалізованими ігровими консолями, а програмування вимагало застосування складних алгоритмів. Однак завдяки зростанню продуктивності персональних комп'ютерів і

розширенню можливостей браузерів стало можливим створення і відображення тривимірної графіки із застосуванням веб-технологій [22].

WebGL (Web Graphics Library) – це JavaScript API для візуалізації інтерактивної 2D та 3D графіки в будь-якому сумісному веб-браузері без використання плагінів. WebGL повністю інтегрований з іншими веб-стандартами, що дозволяє прискореним графічним процесором використання фізики та обробки зображень та ефектів як частини веб-сторінки. Елементи WebGL можна використовувати з іншими елементами HTML і комбінувати з іншими частинами сторінки або фоном сторінки. Програми WebGL складаються з коду управління, написаного на JavaScript, і шейдерного коду, який записаний на OpenGL ES Shading Language (ESSL), мовою, схожою на C або C ++, і виконується на блоці обробки графіки комп'ютера (GPU) [23].

На відміну від інших технологій для роботи з тривимірною графікою (таких як OpenGL і Direct3D), WebGL призначена для використання в веб-сторінках і не вимагає установки спеціалізованих розширень або бібліотек. Одна з переваг WebGL – додатки конструюються як веб-сторінки, тобто одна і та ж програма буде успішно виконуватися на самих різних пристроях (наприклад, на смартфонах, планшетних комп'ютерах та ігрових консолях) [22].

З розвитком HTML розробники отримали можливість створювати все більш складні веб-додатки. На зорі свого розвитку мова HTML пропонував тільки можливість відображення статичного контенту, але з додаванням підтримки JavaScript стало можливим реалізовувати більш складні взаємодії елементів і відображення динамічного контенту. Впровадження стандарту HTML5 дозволило використовувати нові можливості, включаючи підтримку двомірної графіки у вигляді тега canvas. Створення технології WebGL дозволило відображати і маніпулювати тривимірною графікою на веб-сторінках за допомогою JavaScript. За допомогою WebGL розробники можуть створювати абсолютно нові інтерфейси, тривимірні ігри і використовувати тривимірну графіку для візуалізації різної інформації. Незважаючи на значні можливості,

WebGL відрізняється від інших технологій доступністю і простотою використання, що сприяє її швидкому поширенню [23].

Технологія WebGL використовує низькорівневе API, цей аспект полегшує впровадження технології розробниками браузерів в свої продукти, але створює чималі труднощі при створенні інтерфейсів. Велику кількість часу і сил було вкладено в розробку бібліотек, фреймворків і сторонніх програмних засобів, які спростили роботу розробникам сайтів.

WebGL 1.0 базується на OpenGL ES 2.0 та надає API для роботи з тривимірною графікою Він використовує елемент canvas HTML5 і отримує доступ до нього за допомогою DOM [24].

WebGL 2.0 базується на OpenGL ES 3.0 та гарантує доступність багатьох необов'язкових розширень WebGL 1.0 та відкриває нові API [25].

До найпопулярніших інструментів для роботи з тривимірною графікою у веб належать бібліотеки:

– A-Frame;

– PlayCanvas;

– OSG.JS;

– Three JS;

– Babylon JS.

Порівняння названих вище бібліотек наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльна таблиця веб каркасів для побудови тривимірних моделей

Назва	Моделювання	Анімація	Інтегрована фізика	Версія WebGL	Формати імпорту
A-Frame	ні	так	ні	1.0	FBX, OBJ

Продовження таблиці 2.1

PlayCanvas	ні	так	так	1.0	FBX, OBJ
OSG.JS	ні	так	ні	1.0	відсутні
Three JS	ні	так	ні	1.0	FBX, OBJ, STL
Babylon JS	ні	так	так	1.0, 2.0	OBJ, FBX, STL, Babylon, glTF
Sketchfab	ні	так	ні	1.0	.kmz, .las, .lwo, .q3d, .mc2obj, .flt, ., .osg, .ply, .bsp, .md2, .mdl, .shp, .stl, .txp, .vpk, .wrl, .vrm

Отже, проаналізувавши дані таблиці, можемо зробити висновок, що лише три з наведених у ній бібліотеки для побудови тривимірних моделей підтримують формат зберігання полігональних моделей STL.

Також, жодна з наведених бібліотек не підтримує можливість створення тривимірних моделей, а лише відображають раніше експортовані моделі, створені за допомогою спеціалізованого програмного забезпечення.

Щодо версії WebGL, що використовується даними бібліотеками, можемо спостерігати, що представлена у 2017 році версія 2.0 досі не набула широкої популярності, лише одна, з наведених у таблиці бібліотек її підтримує.

3 ПОЛІГОНАЛЬНІ МОДЕЛІ

Багатокутна сітка – це сукупність вершин, ребер і граней, яка визначає форму багатогранного об’єкта в тривимірній комп'ютерній графіці та суцільному моделюванні. Грані зазвичай складаються з трикутників, чотирикутників (квадратиків) або інших простих опуклих багатокутників, оскільки це спрощує візуалізацію, але також може складатися з увігнутих багатокутників або навіть багатокутників з отворами [26].

Вивчення полігональних моделей – це великий підрозділ комп’ютерної графіки (зокрема тривимірної комп'ютерної графіки) та геометричного моделювання. Для різних застосувань і цілей використовуються різні зображення багатокутних сіток. Різноманітність операцій, що виконуються на сітках, може включати: операції двійкової логіки, вирівнювання, спрощення та багато інших.

Об'ємні моделі відрізняються від полігональних тим, що вони явно представляють як поверхню, так і об'єм структури, тоді як полігональних сітки лише явно представляють поверхню (об’єм неявний). Оскільки полігональні сітки широко використовуються в комп'ютерній графіці, існують також алгоритми для відстеження променів, виявлення зіткнень та динаміки твердого тіла з полігоновими моделями. Якщо для кожного краю намальовані лінії, замість того, щоб виводити грані, модель стає моделлю каркаса [27].

Об'єкти, створені за допомогою багатокутних сіток, повинні зберігати різні типи елементів. До них відносяться вершини, краї, грані, багатокутники та поверхні. У багатьох програмах зберігаються лише вершини, ребра та грані або багатокутники. Візуалізатор може підтримувати лише тристоронні грані, тому багатокутники повинні бути побудовані з багатьох із них. Однак багато інструментів візуалізації або підтримують квадрати та різнобічні багатокутники, або можуть перетворити багатокутники на трикутники на льоту, що робить зайвим зберігати сітку у трикутній формі.

Переваги полігональних моделей:

– легке масштабування об’єктів моделі;

– підтримка на апаратному рівні більшості поширених операцій;

– невеликий обсяг даних, що необхідний для опису поверхонь.

Елементи з яких складаються полігональні моделі:

– вершини – положення (як правило, у тривимірному просторі) разом з іншою інформацією, такою як колір, вектор нормалі і координати текстури;

– ребра – зв'язок між двома вершинами;

– грані – набір найближчих ребер, у якого трикутна грань має три ребра, а квадратна грань – чотири ребра. Багатокутник – це копланарний набір граней. У системах, які підтримують багатосторонні грані, багатокутники та грані рівнозначні. Однак більшість апаратних засобів візуалізації підтримують лише 3- або 4-сторонні грані, тому багатокутники представлені у вигляді декількох граней. Математично полігональну сітку можна вважати неструктурованою сіткою або непрямим графіком з додатковими властивостями геометрії, форми та топології;

– многокутники – частіше називаються згладжуючими групами, вони корисні, але не є необхідними для групування гладких областей. Розглянемо циліндр з кришками, наприклад, банку содової. Для плавного затінення сторін усі поверхневі нормалі повинні бути спрямовані горизонтально від центру, тоді як нормалі кришки повинні бути спрямовані прямо вгору та вниз. Винесені у вигляді єдиної, затіненої Фонгом поверхні, вершини складок мали б неправильні нормалі. Таким чином, певний спосіб визначення місця припинення згладжування необхідний для групування гладких частин моделей, подібно до того, як багатокутники групують трибічні грані. Як альтернатива забезпеченню поверхонь та згладжувальних груп, сітка може містити інші дані для обчислення тих самих даних, як кут розщеплення (багатокутники з нормалями вище цього порогу або автоматично розглядаються як окремі згладжувальні групи, або якась техніка, така як розщеплення або скошування фаски, автоматично наноситься на край між ними). Крім того, у сіток з дуже

високою роздільною здатністю виникає менше проблем, які потребують згладжування груп, оскільки їх багатокутники настільки малі, щоб зробити необхідність невідповідною. Крім того, існує ще одна альтернатива у можливості простого від'єднання самих поверхонь від решти сітки. Рендери не намагаються згладити краї поперек суміжних багатокутників;

– поверхні – деякі формати полігональних моделей містять групи, які визначають окремі елементи сітки, і є корисними для визначення окремих суб’єктів для скелетної анімації або окремих акторів для не скелетної анімації;

– тіла – як правило, будуть визначені матеріали, що дозволяють різним ділянкам сітки використовувати різні шейдери під час відтворення [26].

Полігональні моделі можуть бути представлені різними способами, використовуючи різні способи зберігання даних вершини, граней та поверхонь. До них належать:

– список граней – простий список вершин і набір полігонів, які вказують на вершини, які вони використовують;

– «крилате» представлення – кожен край вказує на дві вершини, дві грані та чотири (за годинниковою та проти годинникової стрілки) ребра, які торкаються їх. Крилаті сітчасті сітки дозволяють постійно обходити поверхню, але потребують більше місця на диску;

– півреберні моделі – подібні до крилатих моделей, за винятком того, що використовується лише половина інформації про проходження грані;

– чотириреберні моделі – які зберігають ребра, півграні та вершини без будь-якого посилання на багатокутники. Полігони неявні і їх можна знайти, обходячи структуру. Вимоги до пам'яті схожі на півреберних моделей;

– табліця кутів – які зберігають вершини у заздалегідь заданій таблиці, таким чином, що обхід таблиці неявно визначає багатокутники. Представлення є більш компактним та більш ефективним для отримання полігонів, але операції зі зміни полігонів повільні. Крім того, таблиці кутів не представляють модель повністю. Для представлення більшості сіток потрібно кілька таблиць кутів;

– вершинне представлення – представлені лише вершини, що вказують на інші вершини. Інформація про грані та ребра виражена неявно в цьому поданні. Однак, простота представлення дозволяє проводити над сіткою безліч ефективних операцій [27].

Вибір структури даних визначається застосуванням, необхідною продуктивністю, розміром даних, операціями, які будуть виконуватися. Наприклад, легше мати справу з трикутниками, ніж з багатокутниками загального вигляду, особливо в обчислювальній геометрії. Для певних операцій необхідно мати швидкий доступ до топологічної інформації, такої як ребра або сусідні грані; для цього потрібні більш складні структури, такі як «крилате» представлення.

3.1 Вершинне представлення

Вершинне представлення описує об'єкт як множину вершин, з'єднаних з іншими вершинами. Це найпростіше представлення, але воно не широко використовується, тому що інформація про грані та ребра не виражена явно. Тому потрібно обійти всі дані, щоб згенерувати список граней для візуалізації. Крім того, нелегко виконуються операції на ребрах і гранях. Однак, сітки ВП отримують вигоду з малого використання пам'яті і ефективної трансформації [27].

3.2 Список граней

Полігональна модель, що побудована за допомогою списка граней, представляє об'єкт у вигляді сукупності граней та набору вершин. Це найпоширеніше представлення сітки, оскільки, як правило, приймається сучасним графічним обладнанням [28].

Список граней кращий для моделювання, ніж вершинне представлення тим, що він дозволяє явний пошук вершин грані, і граней оточуючих вершину.

Для візуалізації список граней зазвичай передається до GPU у вигляді набору індексів вершин, а вершини надсилаються у вигляді структур. Цей метод має ту перевагу, що зміни форми, але не геометрії, можна динамічно оновлювати шляхом простого повторного надсилання даних вершин без оновлення зв'язності граней [28].

Моделювання вимагає легкого обходу всіх конструкцій. З моделлю, що використовує список граней дуже легко знайти вершини грані. Також список вершин містить список граней, що виходять з кожної вершини. На відміну від вершинного представлення, і грані, і вершини явні, тому розміщення сусідніх граней і вершин постійно за часом. Однак ребра є неявними, тому все ще потрібен пошук, щоб знайти всі грані, що оточують задану грань. Інші динамічні операції, такі як розділення або злиття граней, також є складними операціями для подібних полігональних моделей [29].

3.3 «Крилате» представлення

«Крилате» представлення – це спосіб представлення полігональних моделей в пам'яті комп'ютера. Це тип граничного зображення та описує як геометрію, так і топологію моделі [30]. Використовуються три типи записів: вершинні записи, записи ребер та записи граней. Даючи посилання на записи ребер, можна постійно відповідати на кілька типів запитів суміжності (запити про сусідні ребра, вершини та грані). Така інформація про суміжність корисна для таких алгоритмів, як розбиття поверхні [26].

«Крилате» представлення чітко описує геометрію та топологію граней, ребер і вершин, коли три або більше поверхонь збираються разом і зустрічаються на спільному ребрі. Упорядкованість така, що поверхні розташовують проти годинникової стрілки щодо орієнтації краю перетину[30].

Структура даних «крилате» реберне представлення дозволяє швидко обходити грані, ребра і вершини через явно пов'язану структуру мережі. Це потужний засіб опису неструктурованої моделі [27].

4 РОЗРОБКА РОЗПОДІЛЕНОЇ СИСТЕМИ ДЛЯ ВІЗУАЛІЗАЦІЇ ПОЛІГОНАЛЬНИХ МОДЕЛЕЙ

В якості практичного завдання на кваліфікаційну роботу магістра була розробка та реалізація розподіленої системи для візуалізації полігональних моделей.

Перелік технологій, що використовувався для реалізації практичного завдання, включає наступне: мова програмування Python версії 3.7, асинхронний веб фреймворк AioHttp, реляційну базу даних MySQL, javascript фреймворк VueJS та ThreeJS.

Використання асинхронного програмування на стороні бекенду дозволило значно підвищити швидкодію та утилізацію ресурсів при надходженні великої кількості конкурентних запитів від клієнтів на отримання змісту полігональних моделей.

Перед початком виконання практичної частини роботи, була розроблена діаграма класів (рис 4.1).

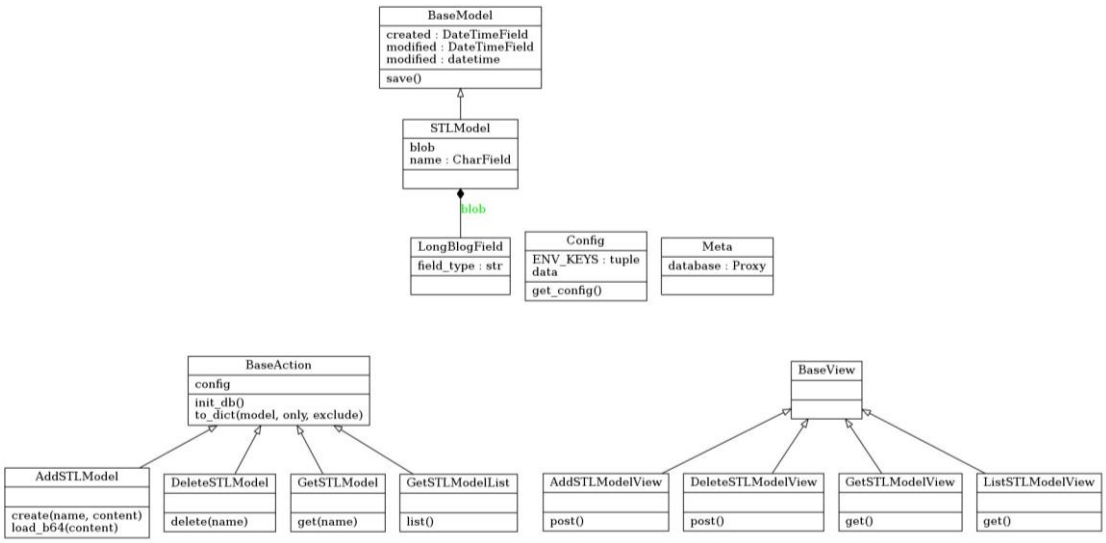

Рисунок 4.1 – Діаграма класів системи

4.1 Мова програмування Python

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [31].

Python підтримує динамічну типізацію, тобто, тип змінної визначається лише під час виконання. З базових типів слід зазначити підтримку цілих чисел довільної довжини і комплексних чисел. Python має багату бібліотеку для роботи з рядками, зокрема, кодованими в юнікоді.

З колекцій Python підтримує кортежі (tuples), списки (масиви), словники (асоціативні масиви) і від версії 2.4, множини [31].

Система класів підтримує множинне успадкування і метапрограмування. Будь-який тип, включаючи базові, входить до системи класів, й за необхідності можливе успадкування навіть від базових типів [32].

Дизайн мови Python побудований навколо об'єктно-орієнтованої моделі програмування. Реалізація ООП в Python є елегантною, потужною та добре продуманою, але разом з тим, достатньо специфічною в порівнянні з іншими об'єктно-орієнтованими мовами [31].

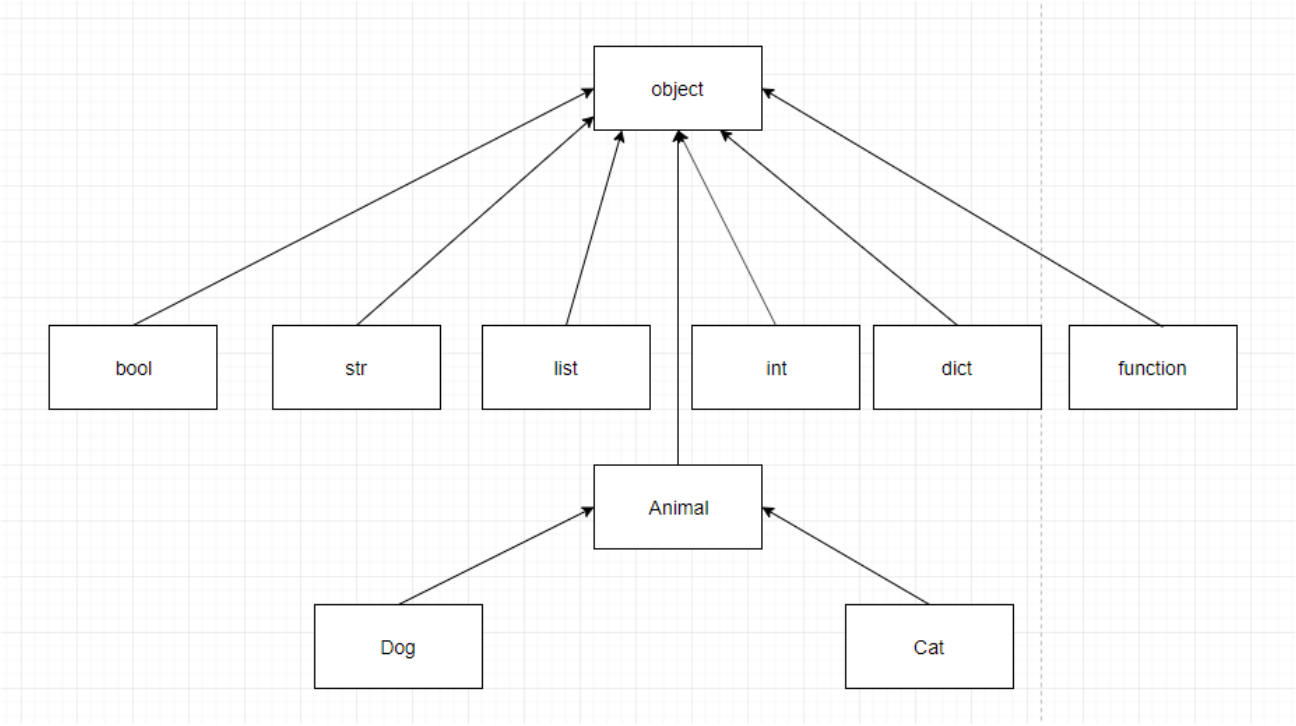

Рисунок 4.2 – Об’єктна модель Python

Можливості та особливості:

– класи є одночасно об'єктами з усіма нижче наведеними можливостями;

– успадкування, в тому числі множинне;

– поліморфізм (всі функції віртуальні);

– інкапсуляція (два рівні – загальнодоступні та приховані методи і поля).

– особливість – приховані члени доступні для використання та помічені як приховані лише особливими іменами;

– спеціальні методи, що керують життєвим циклом об'єкта: конструктори, деструктори, розподільники пам'яті;

– перевантаження операторів (усіх, крім is, '.', '=' і символьних логічних);

– властивості (імітація поля за допомогою функцій);

– управління доступу до полів (емуляція полів і методів, частковий доступ тощо);

– методи для управління найпоширенішими операціями (істинносте значення, len(), глибоке копіювання, серіалізація, ітерація по об'єкту);

– метапрограмування (управління створенням класів, тригери на створення класів, та ін);

– повна інтроспекція;

– класові та статичні методи, класові поля;

– класи, вкладені у функції та інші класи [33].

Серед основних її переваг можна назвати такі:

– чистий синтаксис (для виділення блоків слід використовувати відступи);

– переносність програм (що властиве більшості інтерпретованих мов);

– стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);

– можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);

– стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане на мові Python;

– зручний для розв'язання математичних проблем.

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ [33].

Недоліки:

- низька швидкодія;
- відсутність статичної типізації;
- неможливість модифікації вбудованих класів;
- глобальне блокування інтерпретатора (GIL).

4.2 Asyncio, aiohttp та асинхронне програмування на Python

Асинхронний ввод/вивід є формою неблокованої обробки вводу/виводу, яка дозволяє процесу продовжити виконання не чекаючи закінчення передачі даних.

Операції вводу-виводу (I/O) на комп'ютері можуть бути вельми повільними, в порівнянні з обробкою даних. Пристрій введення-виводу може бути на кілька порядків повільніше, ніж оперативна пам'ять. Наприклад, під час дискової операції, якій потрібно десять мілісекунд для виконання, процесор, який працює на частоті один гігагерц, може виконати десять мільйонів циклів команд обробки [34].

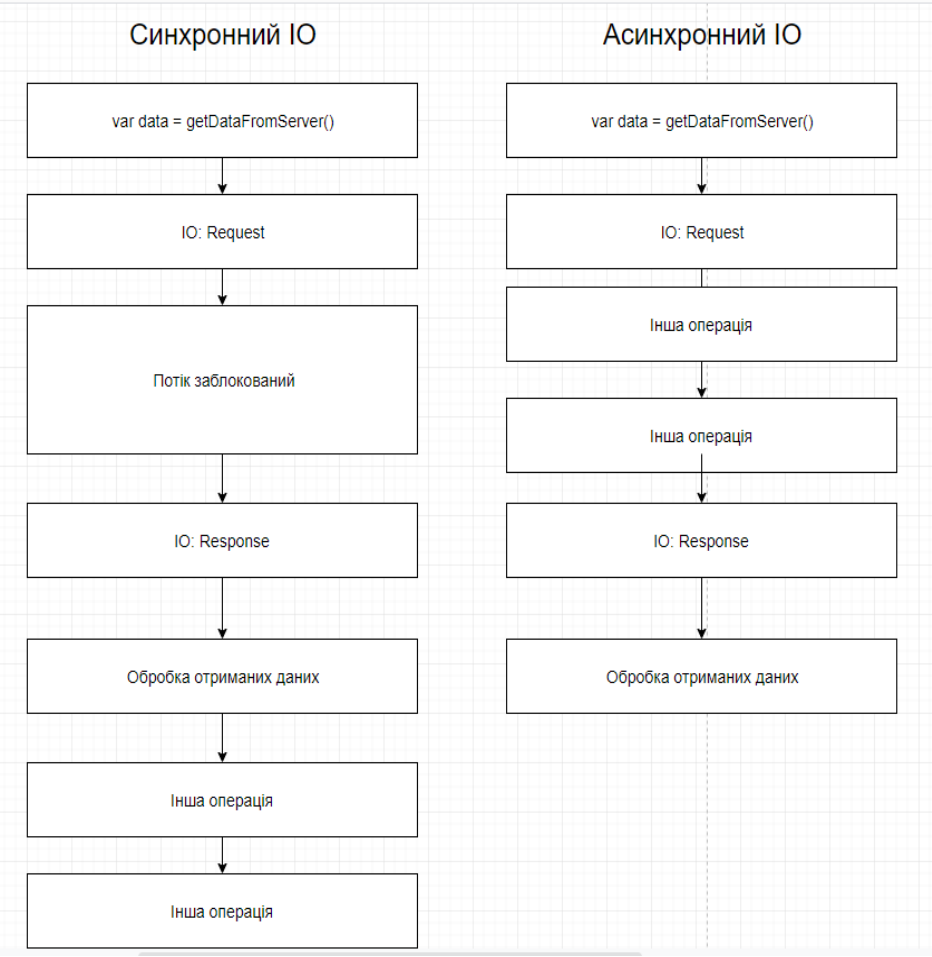

Рисунок 4.3 – Порівняння синхронного та асинхронного виконання програми

Відмінністю неблокуючого вводу-виводу є ефективне використання ресурсів процесора. Наприклад, в додатках з графічним інтерфейсом класичний блокуючий ввод-вивід може заблокувати цикл подій при довгій операції і зробити додаток повільним при взаємодії з користувачем, так як блокується весь потік виконання, в якому і виконується цикл подій. Також неблокуючий IO застосовується в мережевих додатках, де необхідно одночасно обслуговувати декілька клієнтів в одному потоці (процесі) виконання. При блокуючому підході всього лише один «повільний» клієнт уповільнював би весь потік (рис. 4.3).

Назва асинхронний означає, що ми «втрачаємо» контроль над порядком операцій введення-виведення. Порядок визначає операційна система, яка вибудовує операції виходячи з готовності пристроїв введення-виведення [35].

Методи реалізації асинхронного вводу-виводу:

– функції зворотного виклику. Потенційна проблема в тому, що глибина стеку може рости неконтрольовано, тому надзвичайно важливо планувати інше IO тільки після того, коли завершений попередній;

– корутіни (сопрограми). Дозволяють писати асинхронні програми в синхронному стилі. Прикладом є використання async/await в Python або генераторів;

– порти (черги) завершення. Запити введення-виведення видаються асинхронно, але повідомлення про виконання надаються через механізм синхронізаційних черзі в порядку їх завершення [34].

Asyncio – це бібліотека для запису паралельного коду за допомогою синтаксису async/await.

Asyncio використовується як основа для безлічі асинхронних фреймворків Python, які надають високопродуктивні мережеві та веб-сервери, бібліотеки підключення до бази даних, розподілені черги завдань тощо. Asyncio часто ідеально підходить для мережевого коду, що виконує велику кількість операцій вводу/виводу [36].

main.py saved

```python
import asyncio

async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")

async def main():
    await asyncio.gather(count(), count(), count())

if __name__ == "__main__":
    import time
    s = time.perf_counter()
    asyncio.run(main())
    elapsed = time.perf_counter() - s
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

```
One
One
One
Two
Two
Two
main.py executed in 1.00 seconds.
```

Рисунок 4.4 – Виконання асинхронної програми

В якості веб фреймворку для виконання практичної частини дипломного проекту було використано aiohttp. Оскільки це повністю асинхронний веб фреймворк, що має усі вищенаведені переваги асинхронного програмного забезпечення.

Основні можливості:

– Може виступати як в якості HTTP клієнта, так і сервера;

– Підтримує серверні та клієнтські веб сокети;

– Підтримка middleware, сигналів та роутингу.

```python
from aiohttp import web

async def handle(request):
    name = request.match_info.get('name', "Anonymous")
    text = "Hello, " + name
    return web.Response(text=text)

app = web.Application()
app.add_routes([web.get('/', handle),
                web.get('/{name}', handle)])

if __name__ == '__main__':
    web.run_app(app)
```

Рисунок 4.5 – Приклад серверного коду aiohttp

В aiohttp обробник запиту повинен бути підпрограмою, яка приймає екземпляр Request як єдиний аргумент і повертає екземпляр класу похідного від StreamResponse (наприклад, Response).

Обробники налаштовуються на обробку запитів, за допомогою реєстрації у Application.add_routes () за певним маршрутом (за допомогою пари HTTP метод та URL), використовуючи такі обробники, як get () та post () [37].

В середині aiohttp, маршрути обслуговуються за допомогою Application.router (екземпляр UrlDispatcher). Роутер – це список ресурсів. Ресурс – це запис у таблиці маршрутів, який відповідає запитуваній URL-адресі. Ресурс у свою чергу має щонайменше один маршрут. Маршрут відповідає за обробку HTTP-методу за допомогою виклику обробника. Таким чином, коли ви додаєте маршрут, створюється об'єкт ресурсу (рис. 4.6).

```
1   from aiohttp import web
2
3   from backend import views
4
5
6   def setup_routes(app: web.Application):
7       app.router.add_view("/getSTLModel", views.GetSTLModelView)
8       app.router.add_view("/listSTLModels", views.ListSTLModelView)
9       app.router.add_view("/addSTLModel", views.AddSTLModelView)
10      app.router.add_view("/deleteSTLModel", views.DeleteSTLModelView)
```

Рисунок 4.6 – Приклад реєстрації обробників HTTP запитів

Ресурс також може мати змінні у своєму URL. Наприклад, ресурс зі шляхом ' a/{name}/c' буде відповідати всім вхідним запитам із такими шляхами, як '/a/b/c', '/a/1/c' та '/a/etc/c '.

Змінна частина задається у формі {ідентифікатор}, де ідентифікатор може бути використаний пізніше в обробнику запиту для доступу до відповідного

значення для цієї частини. Це робиться шляхом пошуку ідентифікатора в відображенні Request.match_info.

Оскільки aiohttp не диктує жодних деталей реалізації, розробники додатків можуть організовувати обробники в класи (рис 4.7).

Aiohttp забезпечує потужний механізм для налаштування оброблювачів запитів через middleware.

Middleware – це підпрограма, яка може змінювати об’єкти запитів та відповідей.

Middleware зазвичай передають запит обробнику, але вони можуть вирішити ігнорувати його, наприклад повернути HTTP відповідь з кодом 403, якщо користувач не має дозволу на доступ до ресурсу. Вони також можуть відображати помилки, викликані обробником, виконувати деякі перед- або післяобробки, наприклад, обробка CORS тощо.

Оскільки middleware самі по собі є підпрограмами, вони можуть виконувати додаткові await виклики під час створення нового обробника, наприклад, запити в базу даних, тощо.

```
class Handler:

    def __init__(self):
        pass

    async def handle_intro(self, request):
        return web.Response(text="Hello, world")

    async def handle_greeting(self, request):
        name = request.match_info.get('name', "Anonymous")
        txt = "Hello, {}".format(name)
        return web.Response(text=txt)

handler = Handler()
app.add_routes([web.get('/intro', handler.handle_intro),
                web.get('/greet/{name}', handler.handle_greeting)])
```

Рисунок 4.7 – Організація обробників в класи

Aiohttp також підтримує представлення на основі класів (рис. 4.8). Для цього необхідно створити клас-нащадок від базового класу View та визначити методи для обробки HTTP запитів. Обробники повинні бути підпрограмами, що приймають лише аргумент self та повертають об'єкт відповіді як звичайний веб-обробник. Об'єкт запиту можна отримати за допомогою властивості View.request.

Досить часто функції-представлення необхідно повернути відповідь у форматі JSON, для цього aiohttp надає допоміжну функцію json_response(), що що повертає об’єкт відповіді та встановлює необхідні заголовки HTTP відповіді.

Aiohttp має вбудовану підтримку для обробки файлів, завантажених із браузера. Для цього, по-перше, необхідно переконатися, що в HTML формі встановлений атрибут enctype та має значення enctype = "multipart / form-data".

```
11  class BaseView(web.View, CorsViewMixin):
12      pass
13
14
15  class GetSTLModelView(BaseView):
16      async def get(self):
17          name = self.request.query["name"]
18          stl = GetSTLModel(self.request.app["config"])(name)
19          return web.Response(
20              body=stl, headers={"Content-Type": "application/vnd.ms-pki.stl"}
21          )
22
```

Рисунок 4.8 – Представлення на основі класу для отримання вмісту STL моделі

Потім в обробнику запиту можна отримати доступ до поля введення файлу як екземпляр FileField. FileField – це просто контейнер для файла, а також деяких його метаданих. Це дозволяє отримати файл цілком або по частинах. Метод отримання файлів по частинах є більш безпечним, оскільки при отриманні всього вмісту файла за раз, є небезпека що він займе усю доступну ОЗУ. Приклад коду обробника HTTP запиту, що зберігає отриманий файл частинами на рисунку 4.9.

Також, існує можливість завантажити файл, як частину запиту у форматі JSON. Оскільки JSON є текстовим форматом, а вміст файлу представляю собою байти, то ці байти необхідно закодувати. Найпоширенішим способом кодування вмісту файлів для подальшої їх передачі є кодування за допомогою base64.

Base64 – позиційна система числення з основою 64. Система широко застосовується в електронній пошті для передачі бінарних файлів у тексті листа (транспортне кодування). Всі широко відомі варіанти, відомі під назвою Base64, використовують символи A…Z, a…z і 0…9, що становить 62 знаки, для інших двох знаків в різних системах використовуються різні символи [38].

```
async def store_mp3_handler(request):

    reader = await request.multipart()

    # /!\ Don't forget to validate your inputs /!\

    # reader.next() will `yield` the fields of your form

    field = await reader.next()
    assert field.name == 'name'
    name = await field.read(decode=True)

    field = await reader.next()
    assert field.name == 'mp3'
    filename = field.filename
    # You cannot rely on Content-Length if transfer is chunked.
    size = 0
    with open(os.path.join('/spool/yarrr-media/mp3/', filename), 'wb
        while True:
            chunk = await field.read_chunk()  # 8192 bytes by defaul
            if not chunk:
                break
            size += len(chunk)
            f.write(chunk)

    return web.Response(text='{} sized of {} successfully stored'
                             ''.format(filename, size))
```

Рисунок 4.9 – Отримання завантаженого файлу частинами

Base64 – спосіб перетворення бінарної послідовності байтів на послідовність друкованих символів ASCII. Використовується для передачі будь-яких даних, що можуть мати спеціальні або не друковані символи.

Повна специфікація цієї форми base64 міститься в RFC 1421 та RFC 2045. Ця схема застосовується для кодування послідовності октетів (байт). Це відповідає визначенню файлів майже у всіх системах. Закодовані за допомогою base64 дані мають довжину, більшу за оригінальну (у співвідношенні 4:3), виглядають як випадкові символи.

Щоб перекодувати файл у base64, перший байт файлу вміщується в найстарші вісім біт 24-бітного буфера, другий байт – у середні вісім біт, і третій – у молодші вісім біт.

Якщо кодується менш, ніж три байти, то відповідні біти буфера встановлюються в нуль. Далі кожні шість біт буфера, починаючи з найстарших, використовуються як індекс у послідовності ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/. Символ, на який вказує індекс, вміщується у вихідний код. Якщо кодуються тільки один або два байти, використовуються тільки перші два або три символи рядка і результат доповнюється двома або одним символом =. Це запобігає додаванню додаткових бітів для відновлення початкових даних. Процес повторюється над рештою вхідних даних [38].

Кодування Base64 може використовуватися лише у тих випадках, коли відома довжина інформації, яку необхідно закодувати.

```python
import base64

from backend.actions.base import BaseAction
from backend.models import STLModel


class AddSTLModel(BaseAction):
    def __call__(self, name: str, content: str) -> dict:
        stl_model = self.create(name, content)
        return self.to_dict(stl_model, only=[STLModel.name])

    def create(self, name: str, content: str) -> STLModel:
        return STLModel.create(name=name, blob=self.load_b64(content))

    def load_b64(self, content: str) -> bytes:
        return base64.b64decode(content)
```

Рисунок 4.10 – Декодування STL моделі

Для того, щоб зберегти на бекенді файл, що був переданий подібним чином, його спершу необхідно декодувати. До стандартної бібліотеки Python включений модуль, що дозволяє працювати з цим методом кодування. Код, що відповідає за декодування файлу, вміст якого є STL моделлю наведено на рис. 4.10.

4.3 Створення компонентів Vue.js

Vue.js – JavaScript-фреймворк з відкритим вихідним кодом для створення користувацьких інтерфейсів. Легко інтегрується в проекти з використанням інших JavaScript-бібліотек. Може функціонувати як веб-фреймворк для розробки односторінкових додатків в реактивному стилі [39].

Vue.js має поступово прийнятну архітектуру, яка зосереджена на декларативній візуалізації та композиції об’єктів. Він включає такі функції, як маршрутизація, управління станом та інструменти побудови проектів.

Основні складові проекту на Vue.js:

– компоненти – розширюють основні елементи HTML для інкапсуляції та багаторазового використання коду. На високому рівні компоненти це спеціальні елементи, до яких компілятор Vue приєднує певну поведінку (рис. 4.11);

```
<template>
  <model-stl :src="modelSrc"
        :width="600"
        :height="600"
         class="stlviwer">
  </model-stl>
</template>
<script>
  import {ModelStl} from 'vue-3d-model'
  export default {
    name: "STLViewer",
    components: {
      ModelStl,
    },
    props: {
      modelSrc: {
        required: true,
        type: String,
      },
    },
  }
</script>
<style>
  .stlviewer{
    border: 5px black;
  }
</style>
```

Рисунок 4.10 – Компонент Vue.js для відображення STL моделі

– шаблони – Vue використовує синтаксис шаблонів на основі HTML, який дозволяє прив’язувати відображений DOM до даних екземпляра класу. Усі шаблони Vue є валідним HTML, який можна проаналізувати за допомогою веб-переглядачів та HTML-аналізаторів. Vue компілює шаблони у віртуальні функції візуалізації DOM. Віртуальна модель об'єкта документа (або "DOM") дозволяє Vue відтворювати компоненти в пам'яті перед оновленням браузера [40];

– реактивність – Vue має систему реактивності, яка використовує звичайні об'єкти JavaScript та оптимізує повторну візуалізацію. Кожен компонент відслідковує свої реактивні залежності під час його візуалізації, тому система точно знає, коли потрібно виконувати повторну візуалізацію, та які компоненти повторно візуалізувати;

– переходи – Vue надає різноманітні способи застосування ефектів переходу, коли елементи вставляються, оновлюються або видаляються з DOM;

– роутинг – традиційним недоліком односторінкових програм (SPA) є неможливість ділитися посиланнями на точну позицію на сторінці в межах конкретної веб-сторінки. Щоб вирішити цю проблему, багато клієнтських маршрутизаторів розмежовують свої динамічні URL-адреси за допомогою "hashbang" (#!), Наприклад page.com/#!/. Однак у HTML5 більшість сучасних браузерів підтримують маршрутизацію без цих елементів. Vue надає інтерфейс для зміни того, що відображається на сторінці, залежно від поточного шляху до URL-адреси – незалежно від того, як це було змінено (чи за посиланням електронною поштою, оновлення або за допомогою переходу на посилання, що знаходяться на сторінці) [39].

4.4 Створення тривимірної графіки за допомогою бібліотеки Three.js

Three.js – легка кросбраузерна бібліотека JavaScript, що використовується для створення та відображення анімованої комп'ютерної 3D графіки при розробці веб-додатків. Three.js скрипти можуть використовуватися спільно з елементами HTML5 CANVAS, SVG або WebGL [41].

Three.js дозволяє створювати прискорену на GPU 3D графіку, використовуючи мову JavaScript як частина сайту без підключення пропрієтарних плагінів для браузера. Це можливо завдяки використанню технології WebGL.

Особливості:

– рендерер: Canvas, SVG або WebGL;

– сцена: додавання і видалення об'єктів в режимі реального часу; туман;

– камери: перспективна або ортографічна;

– анімація: каркаси, пряма кінематика, інверсна кінематика, покадрова анімація;

– джерела світла: зовнішній, спрямований, точковий; тіні: кинуті і отримані;

– шейдери: повний доступ до всіх OpenGL-шейдерам (GLSL);

– об'єкти: мережі, частинки, спрайт, лінії, скелетна анімація і інше;

– геометрія: площину, куб, сфера, тор, 3D текст і інше; модифікатори: тканина, видавлювання;

– завантажники даних: двійковий, зображення, JSON і сцена;

– експорт та імпорт: утиліти, що створюють Three.js-сумісні JSON файли з форматів: Blender, openCTM, FBX, 3D Studio Max, і Wavefront .obj файл [41].

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розглянуто основні принципи і поняття реалізації та використання розподілених обчислювальних систем, методи їх масштабування. Досліджено історію розвитку таких систем. Розглянута їх класифікація. Наведені поширені ситуації, коли використання подібних систем дозволяє побудувати відмовостійкий високонавантаженій додаток.

Розглянуто існуючі інструменти візуалізації тривимірних моделей, наведено опис та особливості деяких популярних бібліотек, виконано їх порівняння у вигляді таблиці. Описані етапи створення тривимірної анімації, технології, що дозволяють виконувати відображення, моделювання та анімацію тривимірної графіки у веб. Абсолютно всі інструменти візуалізації тривимірної графіки у веб використовують WebGL, що дозволяє візуалізувати інтерактивну 2D та 3D графіку в будь-якому сумісному веб-браузері без використання плагінів.

Наведено характеристики полігональних моделей, їх складові, методи представлення та структури даних, що використовуються для їх зберігання. Вибір структури даних визначається застосуванням, необхідною продуктивністю, розміром даних, операціями, які будуть виконуватися.

Виконане практичне завдання, а саме, було розроблено розподілену систему для візуалізації полігональних моделей. Завдяки тому, що розроблена система не зберігає інформації про поточний стан виконання, вона дозволяє легко виконувати горизонтальне масштабування. А використання асинхронного програмування на стороні бекенду дозволило значно підвищити швидкодію та утилізацію ресурсів при надходженні великої кількості конкурентних запитів від клієнтів на отримання змісту полігональних моделей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Таненбаум Э. Распределенные системы. Принципы и парадигмы. Санкт-Петербург : Питер, 2003. 877 с.

2. Magnoni L. Modern Messaging for Distributed Sytems (sic) // *Journal of Physics: Conference Series*, 2015. №608.

3. Computer cluster. URL: https://en.wikipedia.org/wiki/Computer_cluster (дата звернення: 22.03.2018).

4. Niedermair A. Mastering ServiceStack URL: https://subscription.packtpub.com/book/application_development/9781783986583 (дата звернення: 11.11.2019).

5. Distributed Application Architecture, Sun Microsystems, URL: http://java.sun.com/developer/Books/jdbc/ch07.pdf (дата звернення: 11.11.2019).

6. Benatallah B. Web service conversation modeling: A cornerstone for e-business automation // *IEEE Internet Computing*, 2004. №5. С. 46–54.

7. Dustdar S. A survey on web services composition // *International Journal of Web and Grid Services*, 2005. №1.

8. Фаулер М. Архитектура корпоративных программных приложений. Киев : Диалектика, 2019. 544 с.

9. Eckerson W. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications // *Open Information Systems*, 1995. №3.

10. Quang Hieu Vu. Peer-to-Peer Computing: Principles and Applications, 2010. 336 с.

11. Schollmeier R. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications URL: https://www.researchgate.net/publication/3940901_A_Definition_of_Peer-to-

Peer_Networking_for_the_Classification_of_Peer-to-Peer_Architectures_and_Applications (дата звернення: 11.11.2019).

12. Kamel M. Optimal Topology Design for Overlay Networks // *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*. Atlanta, 2007. С. 714–725.

13. Дейт К. Введение в системы баз данных. Москва : Вильямс, 2001. 1328 с.

14. Розподілена база даних. URL: https://uk.wikipedia.org/wiki/Розподілена_база_даних (дата звернення: 11.10.2019).

15. Реплікація (бази даних). URL: https://uk.wikipedia.org/wiki/Реплікація_(бази_даних) (дата звернення: 11.11.2019).

16. Кузнецов М. В. Самоучитель MySQL. Москва : Вильямс 5, 2007. 560 с.

17. Ли Д. Трёхмерная графика и анимация. Москва : Вильямс, 2002. 640 с.

18. An Historical Timeline of Computer Graphics and Animation. URL: https://studylib.net/doc/8928055/an-historical-timeline-of-computer-graphics-and-animation (дата звернення: 12.11.2019).

19. 3D computer graphics URL: https://www.sciencedaily.com/terms/3d_computer_graphics.htm (дата звернення: 11.11.2019).

20. Трёхмерная графика. URL: https://ru.wikipedia.org/wiki/Трёхмерная_графика (дата звернення: 11.11.2019).

21. Иванов В. П. Трёхмерная компьютерная графика. Москва : Радио и связь. 1995. 224 с.

22. Трехмерная графика в вебе. URL: https://habr.com/ru/post/325646/ (дата звернення: 20.11.2019).

23. Parisi T. WebGL: Up and Running Building 3D Graphics for the Web, 2012. 230 с.

24. WebGL Specification. URL: https://www.khronos.org/registry/webgl/specs/latest/1.0/ (дата звернення: 20.11.2019).

25. WebGL 2.0 Specification URL: https://www.khronos.org/registry/webgl/specs/latest/2.0/ (дата звернення: 21.11.2019).

26. Полігональне моделювання. URL: uk.wikipedia.org/wiki/ Полігональне_моделювання (дата звернення: 20.11.2019).

27. Smith C. On vertex-vertex systems and their use in geometric and biological modelling. Calgary, 2006. 204 с.

28. Baumgart B. Winged Edge Polyhedron Representation. Stanford, 1972.

29. Baumgart B. A polyhedron representation for computer vision. URL: https://people.cs.clemson.edu/~dhouse/courses/405/papers/p589-baumgart.pdf (дата звернення: 23.11.2019).

30. «Крилате» представлення. URL: https://uk.wikipedia.org/wiki/«Крилате»_представлення (дата звернення: 24.11.2019).

31. Python – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 24.11.2019).

32. Лутц М. Программирование на Python Москва: Диалектика, 2015. 992 с.

33. Лутц М. Изучаем Python. Москва: Диалектика, 2011. 832 с.

34. Асинхронный ввод-вывод. URL: https://www.wikiwand.com/ru/Асинхронный_ввод-вывод (дата звернення: 26.11.2019).

35. Microsoft. Synchronous and Asynchronous I/O. URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa365683(v=vs.85).aspx (дата звернення: 26.11.2019).

36. Asynchronous I/O. URL: https://docs.python.org/3/library/asyncio.html (дата звернення: 26.11.2019).

37. Web Server Quickstart. URL: https://aiohttp.readthedocs.io/en/stable/web_quickstart.html (дата звернення: 26.11.2019).

38. Base64. URL: https://uk.wikipedia.org/wiki/Base64 (дата звернення: 26.11.2019).

39. Vue.js. URL: https://en.wikipedia.org/wiki/Vue.js (дата звернення: 26.11.2019).

40. Introduction – Vue.js .URL: https://vuejs.org/v2/guide/ (дата звернення: 26.11.2019).

41. Three.js. URL: https://en.wikipedia.org/wiki/Three.js (дата звернення: 26.11.2019).

ДОДАТОК А

Скріншоти веб інтерфейсу

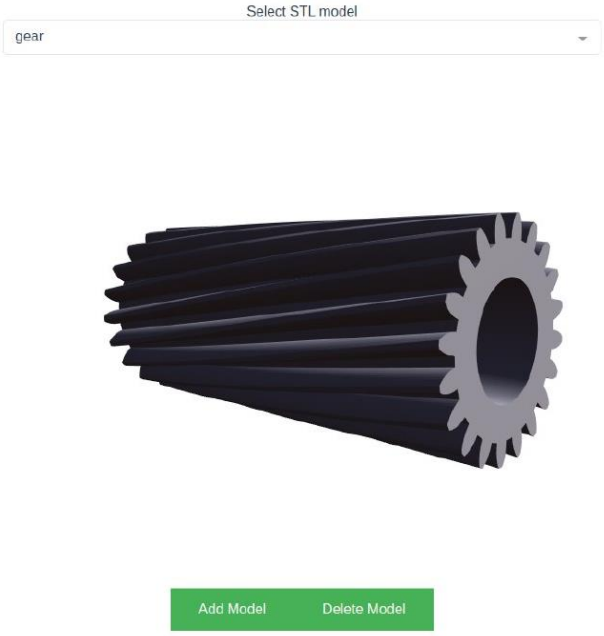

Рисунок А.1 – Відтворення полігональної моделі (зубчата передача)

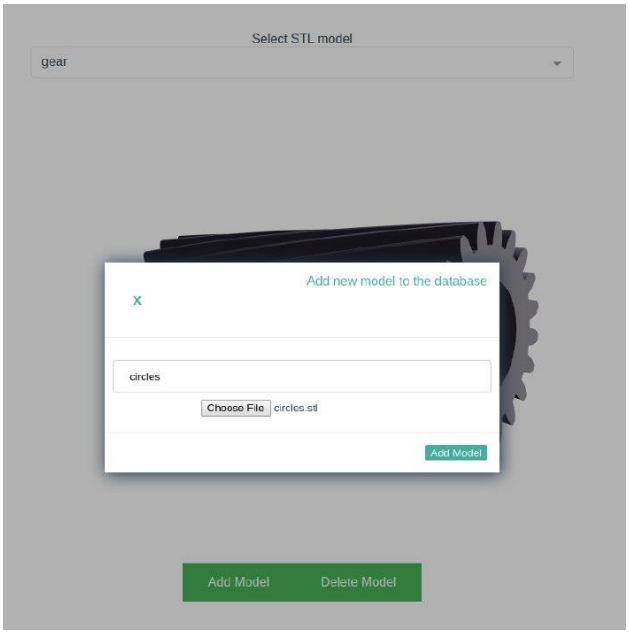

Рисунок А.2 – Додавання нової моделі у БД