МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра загальної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ ВИДАЛЕННЯ ПРИХОВАНИХ ЛІНІЙ»**

Виконав(ла): студент(ка) 2 курсу, групи 8.1229

спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)

освітньої програми комп'ютерні науки
(назва освітньої програми)

Д.Л. Волковський
(ініціали та прізвище)

Керівник доцент кафедри комп'ютерних наук, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри програмної інженерії, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 комп'ютерні науки

(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук, к.т.н., доцент

_____ Борю С.Ю.

(підпис)

« ____ » _____ 2020 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Волковському Давіду Леонідовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Програмна реалізація та дослідження алгоритмів видалення прихованих ліній.

керівник роботи (проекту) Решевська Катерина Сергіївна, к.т.н., доцент

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

затверджені наказом ЗНУ від ««20» травня 2020 року № 576-с

2. Строк подання студентом роботи 01.12.2020

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 алгоритму та його експериментальна оцінка.

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

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.	25.05.2020	
2.	Збір вихідних даних.	28.05.2020	
3.	Обробка методичних та теоретичних		
	джерел.	30.06.2020	
4.	Розробка першого розділу.	07.07.2020	
5.	Розробка другого розділу.	18.09.2020	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи.	05.12.2020	
7.	Захист кваліфікаційної роботи.	14.12.2020	

Студент _____ Д.Л. Волковский
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ К.С. Решевська
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра: «Програмна реалізація та дослідження алгоритмів видалення прихованих ліній»: 64 с., 32 рис., 2 табл., 24 джерел, 5 додатків.

ECLIPSE, JAVA, NETBEANS, АЛГОРИТМ Z-БУФЕРУ, АЛГОРИТМ ВАРНОКА, АЛГОРИТМ ВЕЙЛЕРА-АЗЕРТОНА, АЛГОРИТМ НЬЮЕЛЛА, АЛГОРИТМ ПЛАВАЮЧОГО ГОРИЗОНТУ, АЛГОРИТМ РОБЕРТСА, АЛГОРИТМ ХУДОЖНИКА, АЛГОРИТМИ ВИДАЛЕННЯ ПРИХОВАНИХ ЛІНІЙ, КОМП'ЮТЕРНА ГРАФІКА, ПРОХОДЖЕННЯ ПРОМЕНІВ.

Об'єкт дослідження – алгоритми для видалення прихованих ліній та граней.

Мета роботи – розробка та аналіз алгоритмів видалення прихованих ліній графічних об'єктів.

Метод дослідження – описовий, порівняльний, алгоритмізація, програмування.

Апаратура – персональний комп'ютер, програмне забезпечення.

Результат кваліфікаційній роботи – висновки з аналізу алгоритмів прихованих ліній, алгоритм видалення прихованих ліній та рекомендації стосовно його використання.

У сучасній графіці дуже важливу позицію займають оптимізуючі процеси, особливо у 3D-графіці. Важко уявити собі сучасний 3D-движок без оптимізаційних заходів та процесів, які полегшують навантаження на фізичну та програмну частину пристрою. Один із нишових напрямів – це використання алгоритмів видалення прихованих ліній, граней та площин. У кваліфікаційній роботі присутній огляд відомих алгоритмів видалення прихованих ліній та поверхонь, більш детально розглянуто алгоритми Художника та Z-буферу. Також описується процес розробки додатку з використанням алгоритма видалення прихованих ліній.

SUMMARY

Master's Qualification Thesis: "Software implementation and research of hidden line removal algorithms": 64 pages, 32 figures, 2 tables, 24 references, 5 supplements.

ECLIPSE, JAVA, NETBEANS, Z BUFFER ALGORITHM, WARNOCK'S ALGORITHM, WEILER-ATHERTON ALGORITHM, NEWELL'S ALGORITHM, FLOATING HORIZON ALGORITHM, ROBERTS ALGORITHM, PAINTER'S ALGORITHM, HIDDEN LINE REMOVAL ALGORITHMS, COMPUTER GRAPHICS, RAY CASTING.

The object of the study is algorithms for removing hidden lines and faces.

The purpose of the study is to analyse different algorithms and develop an algorithm for removing hidden lines.

The methods of research are descriptive, comparative, algorithmization, programming.

The hardware are personal computer and software.

The result of the diploma project is an analysis result and algorithm for removing hidden lines.

In modern graphics, optimization processes occupy a very important position, especially in 3D graphics. It is difficult to imagine a modern 3D engine without optimization measures and processes that ease the load on the physical and software part of the device. One of the niche areas is the use of algorithms for removing hidden lines, faces and planes. In the qualification work there is a review of known algorithms for removing hidden lines and surfaces, the algorithms of the Artist and Z-buffer are considered in more detail. It also describes the process of developing an application using an algorithm for removing hidden lines.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 6
Summary 7
Перелік умовних позначень 10
Вступ 11
1 Аналіз алгоритмів видалення прихованих ліній 13
1.1 Алгоритми видалення невидимих частин графічних об’єктів 13
1.1.1 Способи і методи видалення прихованих ліній та поверхонь 14
1.1.2 Вирішення проблеми видалення прихованих ліній та поверхонь за допомогою різних підходів та алгоритмів 17
1.2 Порівняння алгоритмів видалення прихованих ліній та поверхонь 25
1.3 Висновок 27
2 Розробка додатку для реалізації алгоритму видалення прихованих ліній 28
2.1 Обрання бази реалізації та язика програмування 28
2.1.1 Критерії вибору та перелік вимог 28
2.1.2 Основні можливості Java 29
2.2 Опис обраних алгоритмів видалення прихованих ліній 30
2.2.1 Алгоритм Z-Buffer і алгоритм Художника 30
2.2.2 Переваги і особливості обраних алгоритмів 32
2.3 Реалізація на Java 33
2.3.1 Вибір програмного середовища 33
2.3.2 Програмна реалізація 38
2.4 Інструкція до додатку 40
2.4.1 Підготовка робочого місця 41
2.4.2 Редагування коду 41
2.5 Висновок 42

3 Експериментальна оцінка алгоритму видалення прихованих ліній та поверхонь 43
3.1 Аналіз часу роботи розробленого алгоритму 43
3.2 Аналіз відображення фігури розробленого алгоритму 44
3.2.1 Визначення еталонної оцінки відображення фігури 46
3.3 Результати експеримента та їх аналіз 47
3.3.1 Аналіз реалізованих алгоритмів видалення прихованих ліній та поверхонь 47
3.3.2 Аналіз візуального представлення після застосування розробленого алгоритму видалення прихованих ліній та поверхонь 53
3.4 Висновок 54
4 Висновки та майбутня робота 55
Перелік посилань 57
Додаток А ValueResolver.java 59
Додаток Б ValueWorker.java 60
Додаток В ZBuffer.java 62
Додаток Г ZBufferSort.java 65
Додаток Д ZBufferMain.java 68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

3D	Three-dimensional
2D	Two-dimensional
IDE	Integrated development environment
JDK	Java Development Kit

ВСТУП

Проблема видалення прихованих ліній та поверхонь з зображення є частково вирішеною. Існуючі алгоритми які використовуються у програмних продуктах не завжди є ідеальними, а також можуть не підходити для використання у конкретному випадку. Ця проблема найбільш помітна при роботі з невеликими додатками, або при роботі з не оптимізованими програмами.

Даний дипломний проект спрямований на дослідження алгоритмів видалення прихованих ліній та поверхонь, розробку й програмну реалізацію алгоритму, результатом якого є зображення об’єкту без прихованих ліній.

Мета роботи – реалізація алгоритму видалення прихованих ліній та поверхонь.

Завдання:

– ознайомитися з проблемою та існуючими рішеннями;

– проаналізувати існуючі засоби рішення задачі з видалення прихованих ліній та поверхонь;

– підготувати перелік вимог до сучасного рішення задачі з видалення прихованих ліній та поверхонь;

– розглянути існуючі засоби реалізації видалення прихованих ліній й обґрунтувати доцільність вибору тих чи інших інструментів;

– виконати програмну реалізацію алгоритму видалення прихованих ліній за допомогою обраних засобів;

– розробити інструкцію з експлуатації створеного програмного продукту.

Розділ 1 присвячений огляду способів видалення прихованих ліній та поверхонь, роздивлялися різні алгоритми та методи видалення прихованих ліній та поверхонь. Був проведений аналіз трьох розроблених експертами галузі алгоритмів.

У розділі 2 описується вибір програмного середовища для розробки додатку, обґрунтовується вибір алгоритмів. Також детально розбирається його програмна реалізація.

Розділ 3 спрямований на експериментальну перевірку роботи розробленого алгоритму.

1 АНАЛІЗ АЛГОРИТМІВ ВИДАЛЕННЯ ПРИХОВАНИХ ЛІНІЙ

При розробці функціоналу видалення прихованих ліній та поверхонь основним центром реалізації є алгоритм. Цей розділ буде присвячений аналізу існуючих алгоритмів, складностей при роботі з ними, їх характеристиці.

1.1 Алгоритми видалення невидимих частин графічних об’єктів

Видалення прихованих ліній та поверхонь стосується геометричного або відображеного у 3D представлення графів та об’єктів і використовується для покращення швидкості та зменшенню використовуваних ресурсів програми.

Задача видалення прихованих ліній та поверхонь є надзвичайно актуальною й її важливим аспектом є подолання розриву між теоретичними досягненнями і реалізованими рішеннями. Адже разом із розповсюдженням Інтернет-технологій, створенням нових напрямків у науці та впровадженням розвиваючихся підходів, все частіше і частіше виникає потреба у оптимізації графічного відображення об’єктів. Особливо актуальною ця задача стає при потребі візуалізації об’єктів значних розмірів — зокрема при вирішенні різнорідних проблем наукового, технічного та навіть військового характеру. Актуальність даної теми додатково підсилюється тим, що вона не є вузько спеціалізованою, тобто не орієнтована для застосування тільки у певній галузі або сфері, а навпаки повсемісно застосовується в декількох галузях людської діяльності. Програмне забезпечення з алгоритмами видалення прихованих ліній створюється з метою вирішення цих проблем.

1.1.1 Способи і методи видалення прихованих ліній та поверхонь

Через складність задачі по видаленню прихованих ліній та поверхонь було створено багато різноманітних алгоритмів та методів до цих задач. Значна частина всього цього орієнтована на спеціалізоване програмне забезпечення. Найкращого рішення загальної задачі видалення прихованих ліній та поверхонь не існує [1]. Наприклад, для авто- та авіа- тренажерів потрібен швидкий алгоритм, який буде прискорювати частоту кадрів програми. Або для важкого та красивого рендерінгу 3D об’єктів на перший план виходить якість зображення, де алгоритму треба враховувати тіні, прозорість, відбиття світла, паралакс, тощо. Подібні алгоритми, крім спеціалізованих, працюють у доволі повільно, а на обчислення потребують декілька хвилин або навіть годин. Треба відмітити, що врахування впливу ефектів на зображення не входить до завдання алгоритмів видалення прихованих ліній та поверхонь. Краще вважати їх частиною процесу візуалізації зображення. Проте багато з цих ефектів присутні в алгоритмах видалення прихованих ліній та поверхонь.

Існує пряма залежність між швидкістю роботи алгоритму і якістю його результату. Жоден з алгоритмів не може досягти ідеальних результатів для цих двох показників одночасно. Після появи все більш швидких алгоритмів з’являється можливість будувати більш детальні зображення. Реальні або реалістичні задачі, проте, завжди вимагатимуть урахування більшої кількості деталей.

Алгоритми видалення прихованих ліній та поверхонь використовують для визначення ліній, поверхонь чи фігур, які приховані та неприховані для спостерігача або екрану, який знаходиться у певній точці в просторі(дивись рисунок 1.1).

Рисунок 1.1 – Приклад зображення з видаленням прихованих ліній

Всі алгоритми видалення прихованих ліній та поверхонь містять у собі алгоритм сортування[2]. Порядок, в якому проводиться це сортування координат об'єктів не впливає на ефективність цих алгоритмів. Але сортування може вестися по геометричній відстані від тіла, від поверхні, від ребра або від точки у просторі до точки спостереження.

Головна ідея сортування за відстанню полягає в тому, що чим далі розташований об'єкт від точки спостереження, тим більша ймовірність, що він буде повністю або частково закритий одним з об'єктів, ближчих до точки спостереження. Після визначення відстаней або пріоритетів по глибині залишається провести сортування по горизонталі і по вертикалі щоб з'ясувати чи буде даний об'єкт дійсно закритий іншим об'єктом, що розташований ближче до точки спостереження.

Ефективність будь-якого алгоритму видалення прихованих ліній та поверхонь значно залежить від ефективності процесу сортування. Для підвищення ефективності сортування використовується також когерентність сцени, тобто тенденція незмінності характеристик сцени у малому.

Алгоритми видалення прихованих ліній та поверхонь можна класифікувати за способом вибору системи координат та простору, в якому вони працюють [3]. Виділяють три класи алгоритмів видалення прихованих ліній та поверхонь:

а) Алгоритми, що працюють в об'єктному просторі;

б) Алгоритми, що працюють в просторі зображення (екрана);

в) Алгоритми, що формують список пріоритетів.

Алгоритми, що працюють в об'єктному просторі мають справу з фізичної системою координат, в якій описані ці об'єкти. При цьому виходять дуже точні результати, які обмежені лише точністю обчислень. Отримані зображення можна вільно збільшувати або зменшувати. Алгоритми, що працюють в об'єктному просторі особливо корисні в тих додатках, де необхідна висока точність.

Алгоритми, що працюють в просторі зображення починають працювати з системою координат того екрана, на якому об'єкти візуалізуються. При цьому точність обчислень обмежена роздільною здатністю екрану. Раніше дозвіл екрана був досить низьким, наприклад 512x512 точок, проте зараз з’являються все більші та більші екрани, наприклад 15360x8640 пікселів. Це, відповідно збільшує витрати ресурсів системи на створення зображення.

Результати, що були отримані в просторі зображення а потім збільшені у декілька разів не будуть відповідати початковій сцені. Наприклад, можуть не збігатися кінці відрізків. Алгоритми, що формують список пріоритетів, працюють в обох згаданих системах координат.

Обсяг обчислень для будь-якого алгоритму, працюючого в об'єктному просторі дорівнює порівнянню кожного об'єкта сцени із іншими об'єктами цієї сцени і зростає теоретично як квадрат числа об'єктів (n^2).

Аналогічно, обсяг обчислень для будь-якого алгоритму, що працює в просторі зображень і який порівнює кожен об'єкт сцени з позиціями всіх пікселів в системі координат екрана, зростає теоретично, як nN. Де n – це кількість об'єктів на сцені, а N – число пікселів. Теоретично складність алгоритмів, що працюють в об'єктному просторі, менше складності алгоритмів, що працюють в просторі зображення при n <N. Однак на практиці це не так. Справа в тому, що алгоритми, які працюють в просторі зображення більш ефективні ніж алгоритмів, які працюють в об'єктному просторі, тому що для перших легше скористатися перевагою когерентності при растрової реалізації.

1.1.2 Вирішення проблеми видалення прихованих ліній та поверхонь за допомогою різних підходів та алгоритмів

Алгоритми видалення прихованих ліній та поверхонь у значній степені відрізняються один від одного, спираючись на ціль, для якої вони й розроблялися. Нижче будуть описані декілька відомих алгоритмів та їх особливості.

Алгоритм плаваючого горизонту. Його можна віднести до класу алгоритмів, що працюють в просторі зображення. Алгоритм плаваючого горизонту краще за все використовується для видалення прихованих ліній тривимірного уявлення функцій, що описують поверхню у вигляді формули F(x,y,z) = 0[4]. Ідея даного методу полягає в зведенні тривимірної задачі до двовимірної шляхом перетину вихідної поверхні послідовністю паралельних січних площин, що мають постійні значення координат x, y або z. Нижче зображено типовий результат роботи алгоритму плаваючого горизонту для функції(дивись рисунок 1.2):

$$y = \frac{1}{5}\sin(x)\cos(z) - \frac{3}{2}\cos\left(\frac{7\alpha}{4}\right)e^{(-\alpha)}$$

$$\alpha = (x-\pi)^2 + (z-\pi)^2 \text{ на інтервалі } \{0; 2\pi\}$$

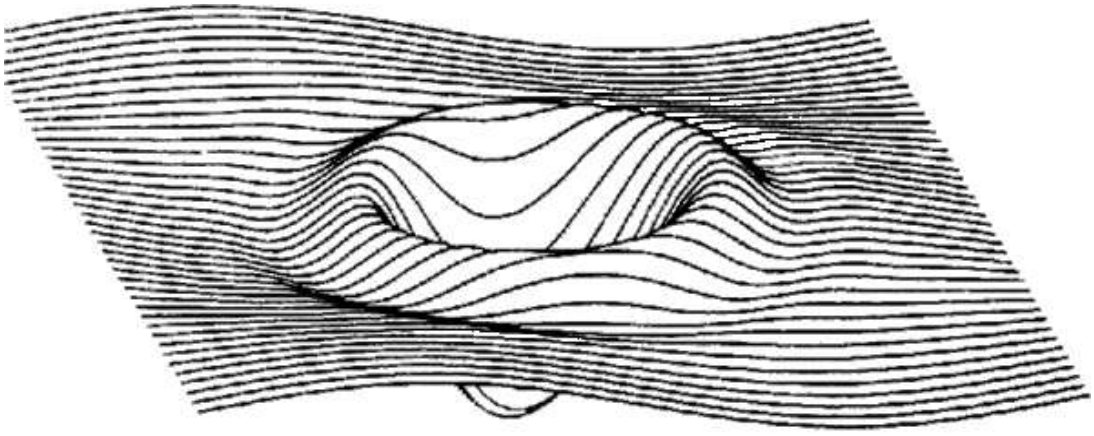

Рисунок 1.2 – Результат роботи алгоритму плаваючого горизонту

Алгоритм Робертса. Цей алгоритм являє собою перше запропоноване рішення задачі про видалення прихованих ліній[5]. Цей представлений математичний метод працює в об'єктному просторі. Алгоритм видаляє з кожного тіла ті ребра чи площини, які екрануються самим тілом. Потім кожне з видимих ребер кожного тіла порівнюється з усяким із решти тіл для визначення того, яка його частина або частини, екрануються цими тілами, якщо вони існують. Тому обчислювальна складність алгоритму Робертса зростає як квадрат числа об'єктів. Математичні методи, що використовуються в цьому алгоритмі прості і точні. Більш пізні реалізації алгоритму, що використовують попереднє пріоритетне сортування вздовж осі z демонструють лінійну залежність від числа об'єктів (дивись рисунок 1.3).

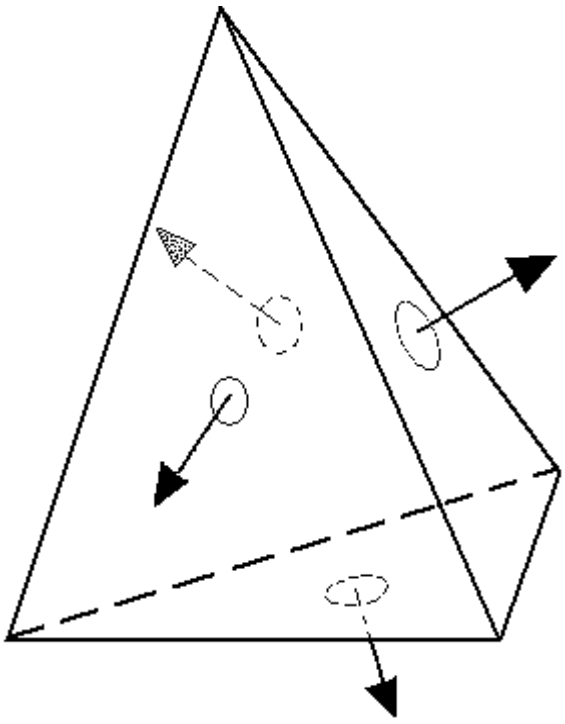

Рисунок 1.3 – Алгоритм Робертса, визначення не лицьових граней

Алгоритм, що використовує z-буфер. Вперше він був запропонований Кетмулом. Працює цей алгоритм в просторі зображень. Ідея z-буфера є узагальненням ідеї про буфер кадру[6]. Він використовується для

запам'ятовування атрибутів (z-координати) кожного пікселя в просторі зображення. Z-буфер – це окремий буфер глибини, який використовують для запам'ятовування z координати, інакше кажучи глибини кожного видимого пікселя в просторі зображення.

У процесі роботи глибина чи z координата кожного нового пікселя, який потрібно занести в буфер кадру, порівнюється з глибиною того пікселя, який вже занесений в z-буфер. Якщо це порівняння показує, що новий піксель знаходится перед пікселем, який вже розміщений в буфері кадру, то новий піксель заноситься в цей буфер і також проводиться коригування z-буфера новим значенням z. Якщо ж порівняння дає протилежний результат, то ніяких дій не проводиться. По суті, алгоритм є пошуком по x і y найбільшого значення функції z (x, y).

Головна перевага алгоритму – його простота. Він вирішує задачу про видалення прихованих ліній та поверхонь і робить тривіальним виконання візуалізації перетину складних поверхонь. Сцени можуть бути будь якої складності, оскільки розмір простору зображення фіксований. Оцінка обчислювальної трудомісткості алгоритму лінійна. Оскільки елементи сцени чи картинки можна заносити в буфер кадру або в z-буфер в довільному порядку, їх не потрібно попередньо сортувати по пріоритету глибини при статичній сцені. Це може зменшити обчислювальний час, що витрачається на сортування.

Основний недолік алгоритму z-буферу – великий обсяг необхідної пам'яті. Якщо сцена піддається видовому перетворенню і відсікається до фіксованого діапазону значень координати z, то можна використовувати z-буфер з фіксованою точністю. Інформацію про глибину потрібно обробляти з більшою точністю, ніж координатну інформацію на площині (x, y); Інший недолік алгоритму z-буфера полягає в складності і тяжкій реалізації ефектів прозорості та просвічування.

Метод проходження променів (ray casting). У цьому методі для кожного пікселя на картинній площині визначається найближча грань. Для

цього через цей піксель випускається промінь, який знаходить всі його перетени з гранями і серед цих перетинів вибирається найближча до нього грань[7]. Цей метод проходження променів дещо схожий на алгоритм z-буфера, проте тут цикли по пікселям і по об'єктам змінені місцями.

Алгоритми, що використовують список пріоритетів. При реалізації алгоритмів видалення прихованих ліній та поверхонь, що ми розглянули вище, встановлювалися пріоритети, тобто глибина об'єктів сцени або їх відстані від точки спостереження чи екрану. Алгоритми, що використовують список пріоритетів намагаються отримати перевагу за допомогою сортування по глибині або по пріоритету. Це сортування може проходити перед виконанням алгоритму видалення прихованих ліній та поверхонь. Мета такого сортування полягає в отриманні остаточного списку об'єктів сцени, що упорядковані по пріоритету глибини, заснованому на відстані від точки спостереження. Якщо такий список буде статичним, то ніякі два об'єкта не будуть взаємно перекривати один одного. Тоді можна буде записати всі об'єкти в буфер кадру по черзі, починаючи з найбільш віддаленого від точки спостереження до найближчого. Ближчі до спостерігача об'єкти будуть заміняти інформацію про більш далекі об'єкти в буфері кадру. Тому завдання про видалення прихованих ліній та поверхонь вирішується тривіально. Ефекти прозорості можна включити до складу алгоритму шляхом часткового коригування вмісту буфера кадру з урахуванням атрибута прозорості об'єктів.

Алгоритм Художника. Для простих об'єктів сцени, наприклад, для багатокутників, цей алгоритм іноді називають алгоритмом Художника[8], оскільки він тотожній тому способу, яким художник створює свою картину. Спочатку художник малює задній фон, потім предмети і елементи, що лежать на середній відстані, і, нарешті, передній план, воду чи дерева. Тим самим художник вирішує задачу видалення прихованих поверхонь, або задачу видимості, через побудову картини в порядку зворотної пріоритету (дивись рисунок 1.4). Також алгоритм Художника має іншу назву – Z-сортування.

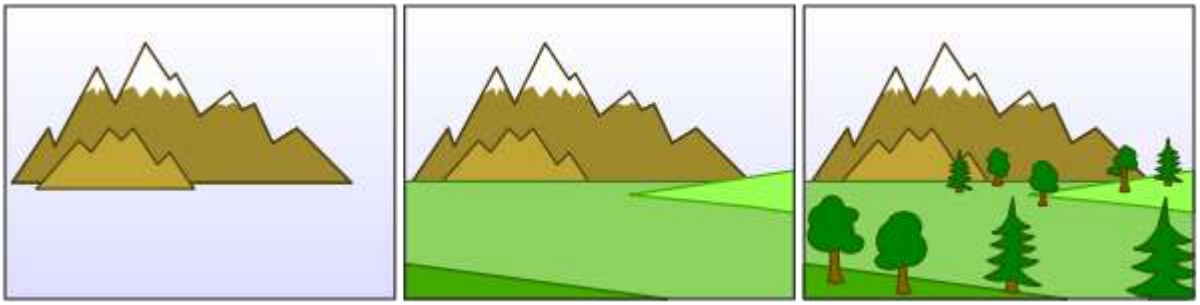

Рисунок 1.4 – Алгоритм Художника в дії

Алгоритм Ньюелла. Його ще можуть називати алгоритмом Ньюела-Ньюела-Санча, бо він був запропонований в 1972 році Мартіном Ньюеллом, Діком Ньюеллом і Т. Санчей[9]. Мета цього алгоритму – вирішити проблему циклічного перекриття багатокутників, яка притаманна алгоритму Художника(дивись рисунок 1.5).

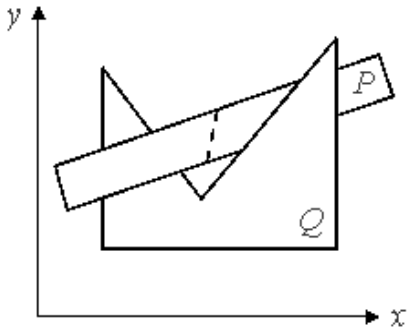

Рисунок 1.5 – Перекриття багатокутників

Починається все з формування списку багатокутників, які впорядковуються по значенню z-min для кожної такої фігури. Першим у списку буде такий багатокутник, який має найменше значення z-min. Він буде найбільш віддаленим по z-напряму. Позначимо цей багатокутник через Q, а наступний за ним багатокутник через P. Сортування цих багатокутників, позначенних P і Q відбувається в кілька етапів, де з'ясовується їх взаємне розташування. Ми можемо виділити декілька ситуацій розташування, у які P і Q можуть потрапляти відносно один одного:

– Якщо крайні значення z-координат усіх вершин Q лежать далі, ніж P, сортування тривіальне. Q буде сортуватися пізніше;

– Якщо багатокутники, що порівнюються, не перекриваються крайніми значеннями в площині X, У, їх не потрібно сортувати, оскільки вони не перетинаються один з одним;

– Якщо всі вершини Q більш віддалені від глядача, ніж рівень вершин в P, то Q упорядкований;

– Якщо всі вершини P до глядача ближче, ніж вершини на рівні Q, то Q упорядкований;

– Якщо X, Y значення P і Q перетинаються, то їх не потрібно сортувати;

– Якщо досі немає сортування і не вдалося циклічно перекрити багатокутники, то у цьому випадку, вони повинні виділятися і сортування буде продовжене з не циклічних частин. Поділ відбувається на одному з багатокутників на розрізаній кромці з іншого багатокутника.

Багатокутники повинні бути плоскі, тобто, всі вершини повинні лежати на одній площині. Порядок кроків обирається таким чином, щоб прості тести спочатку відбувалися першими, а більш складні тести виконувались в кінці. Це зроблено з метою мінімізації часу на обчислення.

Алгоритм Варнока. Алгоритм Варнока є одним із прикладів алгоритму, заснованого на розбитті картинної площини на деякі частини, кожна з яких у результаті містить таку задачу, яка може бути вирішена досить просто. Оскільки алгоритм Варнока націлений на обробку створеного рисунку, він працює в просторі зображення. У просторі зображення спершу розглядається деяке обране вікно і вирішується питання про те, чи порожньо воно, чи його вміст досить простий для візуалізації(дивись рисунок 1.6). Якщо це не так, то вікно розбивається на фрагменти і буде розбиватися до тих пір, поки вміст фрагмента не стане досить простим для візуалізації або його розмір не досягне встановленої межі розбиття[10].

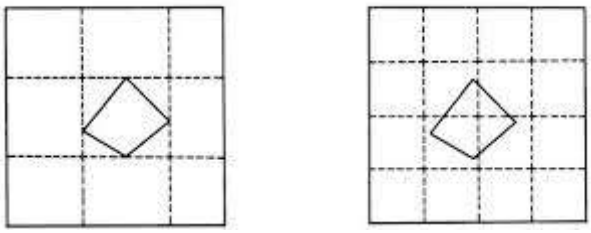

Рисунок 1.6 – Приклади розбиття області на квадрати

Як результат, можливі чотири різних випадка:

– Проекція грані повністю накриває область;

– Проекція грані перетинає область, але не міститься в ній повністю;

– Проекція грані цілком міститься всередині області;

– Проекція грані не має загальних внутрішніх точок з розглянутою областю.

Зазначимо, що в останньому випадку грань взагалі ніяк не впливає на те, що видно в даній області. Порівнюючи область з проекціями всіх граней можна виділити випадки, коли зображення, яке входить в дану область, визначається відразу:

– Коли проекція жодної грані не потрапляє в область.

– Коли проекція тільки однієї грані міститься в області або перетинає область. У цьому випадку проекції грані розбивають всю область на дві частини, одна з яких і відповідає цій проекції.

– Коли існує грань, проекція якої повністю накриває дану область і ця грань розташована до картинної площини ближче, ніж всі інші грані, проекції яких перетинають дану область. В даному випадку область відповідає цій встановленій межі.

Якщо жоден з розглянутих трьох випадків не відбувся, то знову розбиваємо область на чотири рівні частини і перевіряємо виконання цих умов для кожної з нових частин. Ті частини, для яких не вдалося встановити видимість, розбиваємо знову, тощо. Звісно, виникає питання про критерії, на

підставі якого слід зупиняти розбиття області на квадрати. У якості першого критерію можна взяти розмір області. Як тільки розмір області стане не більше розміру, наприклад, одного пікселя, то проводити подальше розбиття не має сенсу і для даної області найближча до неї грань визначається явно, як в методі проходження променів. Іноді, для усунення сходового ефекту, процес роздроблення проводиться до розмірів менших, ніж дозвіл екрана на один піксель. При цьому усереднюються атрибути виникнених подпікселей, щоб визначити атрибути самих пікселів. Якщо у вікні міститься тільки один багатокутник і якщо він цілком охоплений цим вікном, то його легко зобразити, не проводячи подальшого розбиття. Такий спосіб розбиття корисний, зокрема, при мінімізації кількості кроків розбиття для простих сцен. Однак зі зростанням складності сцени цей алгоритм втрачає свою перевагу.

Алгоритм Вейлера-Азертона. Розбиття на картинній площині можна робити не тільки лініями, що паралельні координатним осям, але й по кордонах проекцій граней. У результаті виходить точне рішення задачі[11]. Цей метод працює з проекціями граней на картинну площину.

В якості першого кроку проводиться сортування всіх граней по глибині. Потім зі списку залишившихся граней береться найближча грань (грань А), а всі інші грані обрізаються по цій межі. Якщо проекція грані В перетинає проекцію грані А, то грань В розбивається на частини. Частини розбиті так, що кожна частина або міститься в межі А, або не має з нею загальних внутрішніх точок (дивись рисунок 1.7).

Таким чином отримуються дві множини граней: Fa – грані, проекції яких містяться в проекції грані А (сюди входить і сама грань А), і Fb – межі, проекції які не мають спільних внутрішніх точок з проекцією межею А. Множину Fa зазвичай називають множиною граней, які внутрішні по відношенню до А.

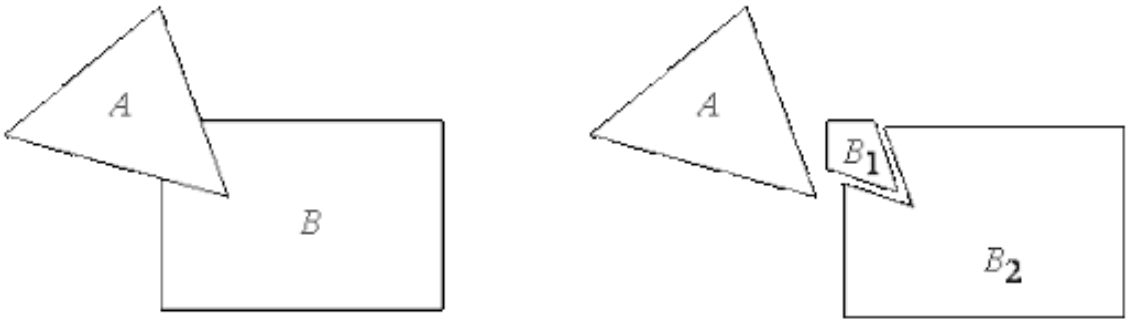

Рисунок 1.7 – Приклад роботи алгоритму Вейлера-Азертона

Однак у множині Fa можуть міститися грані, які знаходяться до спостерігача ближче, ніж сама грань A (це можливо, наприклад, при циклічному накладенні граней). У цьому випадку кожна така грань використовується для розбиття всіх інших граней з множини Fa (включаючи вихідну грань A). Коли рекурсивне розбиття завершиться, то всі грані з першої множини виводяться з набору цих залишившихся граней. Потім з набору Fb граней береться наступна грань і процедура повторюється.

1.2 Порівняння алгоритмів видалення прихованих ліній та поверхонь

Давайте подивимось на порівняння алгоритмів, що ми вже розглянули та з’ясуємо, коли ці алгоритми краще застосовувати(дивись таблицю 1.1). Пояснення для складності алгоритмів:

– N – кількість граней;

– C – кількість пікселів.

До таблиці будуть входити назва алгоритму, простір застосування, його складність та коли цей алгоритм краще застосовувати.

Таблиця 1.1 – Порівняння алгоритмів

Назва	Коли краще застосовувати	Простір застосування	Складність
Алгоритм Робертса	для зображення безлічі опуклих багатогранників на одній сцені у вигляді дротяної моделі.	створен для в об'єктного простору	O(N^3)
Алгоритм z-буфера	для зображення динамічних та статичних сцен, особливо якщо є невикористані резерви оперативної пам’яті.	створен для простору зображень	O(CN)
Алгоритм Художника	для зображення статичних або малодинамічних сцен.	використовує список пріоритетів	O(N+C)
Метод проходження променів	дещо схожий на алгоритм z-буфера, однак тут цикли по пікселям і по об'єктам змінені місцями.	створен для простору зображень	O(CN)
Алгоритм плаваючого горизонту	для видалення невидимих ліній тривимірного уявлення функцій, що описують поверхню у вигляді формули F(x,y,z) = 0.	створен для простору зображень	O(CN)
Алгоритм Ньюелла	як і алгоритм Художника, проте з успіхом може відображати циклічне перекриття багатокутників.	використовує список пріоритетів	O(CN^5)
Алгоритм Варнока	при мінімізації числа розбиття, тобто зображенні простих сцен.	створен для простору зображень	O(CN)
Алгоритм Вейлера-Азертона	коли є можливість працювати з проекціями граней на картинну площину.	використовує список пріоритетів	O(CN^3)

1.3 Висновок

У цьому розділі ми зробили огляд існуючих рішень видалення прихованих ліній та поверхонь, подивилися на те, як інші фахівці мають справу з цією проблемою та порівняли відомі алгоритми.

2 РОЗРОБКА ДОДАТКУ ДЛЯ РЕАЛІЗАЦІЇ АЛГОРИТМУ ВИДАЛЕННЯ ПРИХОВАНИХ ЛІНІЙ

У цьому розділі представлений короткий огляд можливостей мови програмування, обраної IDE, дослідження роботи поточного алгоритму з визначенням його сильних і слабких сторін. Також детально описується процес розробки додатку. Наприкінці присутня інструкція для нього.

2.1 Обрання бази реалізації та язика програмування

Обрання мови програмування - це важливий крок, від якого буде залежати усякий подальший розвиток додатку, адже особливості кожної мови програмування можуть як сприяти покращенню якості, так і навпаки. Спершу треба скласти критерії вибору мов програмування.

2.1.1 Критерії вибору та перелік вимог

Перш за все треба відзначити саме для чого потрібна мені мова програмування: для роботи з графікою, тому підтримка графічних пакетів та бібліотек, що працюють з 3D та 2D графікою є першою вимогою до мови програмування. Також із тривіальних вимог є можливість написання свого додатку на цій мові програмування, імплементування додатку та запуск його у спеціальному програмному середовищі. Сюди входить й особисті навички та зручність використання певною мовою програмування.

Список вимог не дуже великий та прискіпливий, тому ми вже можемо виділити декілька мов програмування:

– Python;
– Java;
– JavaScript;
– C#;
– C++;
– Delphi;
– OpenGL.

Спираючись на текст вище та описані мною критерії, я обрав для себе мову програмування Java. Головною причиною вибору стало те, що це мова, з якою я знайомий найкраще серед інших представлених мною мов програмування або мов, що підходять під критерії вибору.

2.1.2 Основні можливості Java

Я буду описувати можливості Java Runtime Environment та JDK яку я саме і використовую. На цей час останньою версією була Java Standard Edition (далі Java SE) 15.0.1 яка в мене встановлена. Java SE є частиною JDK тієї ж версії.

Java[12] – строго типизований об’єктно-орієнтований язик програмування, розроблений компанією Sun Microsystems, в подальшому купленний Oracle. Язик програмування та основні його реалізації розповсюджуються по ліцензії General Public License.

Додатки Java транслюються у спеціальний байт-код, тому вони можуть працювати на будь-якій комп’ютерній архітектурі, для якої існує реалізація віртуальної Java-машини. Дата офіційного виходу – 23 травня 1995 року. На 2020 рік Java – одна з найпопулярніших мов програмування.

2.2 Опис обраних алгоритмів видалення прихованих ліній

У цій частині буде наданий детальний опис обраних алгоритмів та надані їх порівняльні характеристики.

2.2.1 Алгоритм Z-Buffer і алгоритм Художника

Алгоритм, який буде застосовуватись у моєму дипломному проекті являє собою поєднання добре відомого алгоритму Z-Buffer з деякими частинами алгоритму Художника.

Алгоритм Z-Buffer – алгоритм, що зберігає інформацію про вже оброблені об’єкти на сцені у двох створених буферах: буфері кадру і z-буфері (дивись рисунок 2.1)[13].

– У буфері кадру для кожного пікселя зберігається інформація про колір об'єкта, що відображається ним на даний момент;

– В z-буфері для кожного пікселя зберігається z-координата видимого на даний момент об'єкта.

Припустимо, що ми вибрали піксель і перетворюємо об'єкт (дивись рисунок 2.2)[14]

– Якщо z-координата об'єкта в цьому пікселі менше, ніж z-координата, що зберігається в z-буфері, тоді новий об'єкт лежить перед видимим на даний момент об'єктом. Тоді занесемо колір нового об'єкта в буфер кадру, а його координату – в z-буфер;

– Якщо z-координата об'єкта в цьому пікселі більше, ніж z-координата, що зберігається в z-буфері, то новий об'єкт прихований і буфери залишаться без змін.

У цього алгоритму є свій недолік: для зберігання z-буфера потрібна додаткова кількість пам'яті та потрібна додаткова перевірка кожного пікселя.

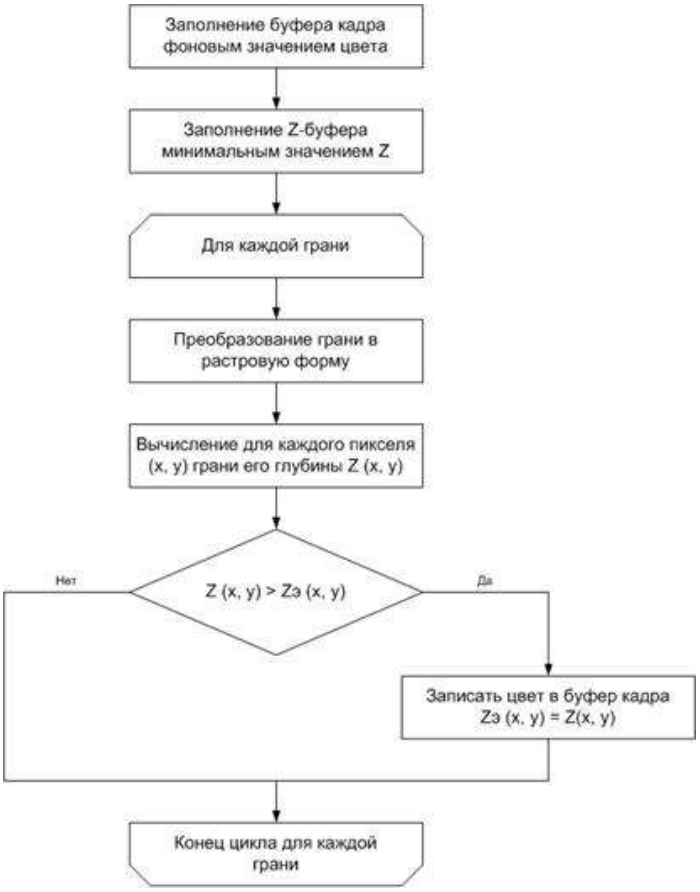

Рисунок 2.1 – Блок-схема алгоритму z-буфера.

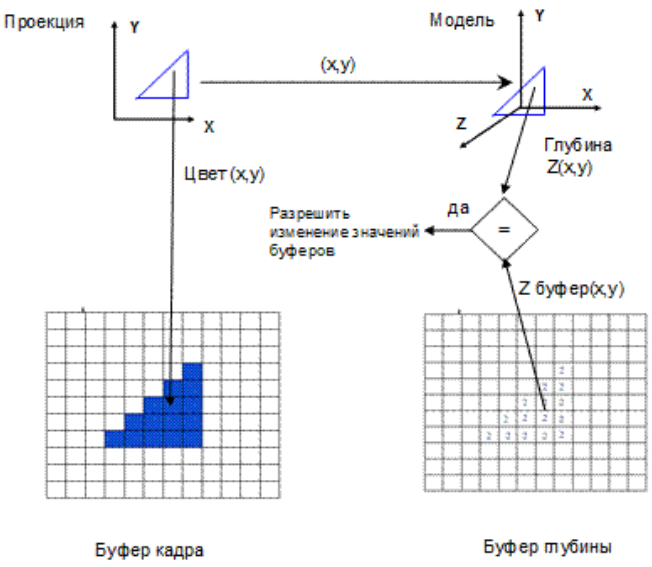

Рисунок 2.2 – Схема роботи алгоритму z-буфера.

Алгоритм Художника уникає додаткових витрат пам’яті, спочатку сортуючи об’єкти по відстані від них самих до точки огляду. Тоді об’єкти перевіряються в так званому порядку глибини, починаючи від самого

далекого. У такому випадку при розгляді об'єкта вже не потрібна перевірка його z-координати, ми завжди будемо записувати колір в буфер кадру. Значення, що зберігалися в буфері раніше, просто перезаписуються (дивись рисунок 2.3)[15].

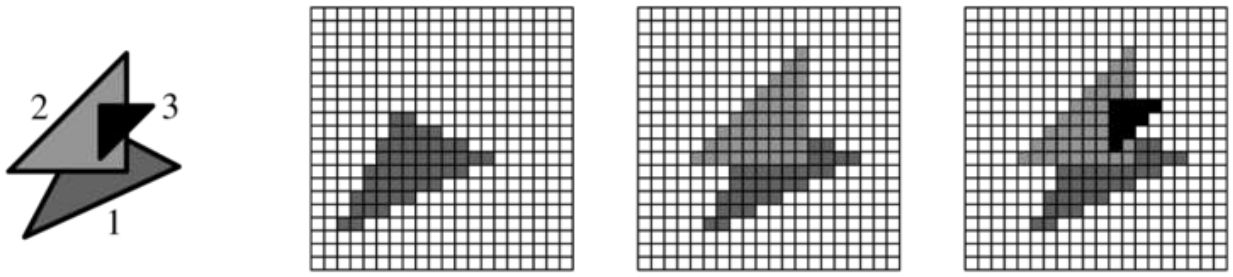

Рисунок 2.3 – Кроки роботи алгоритму з впорядкованими фігурами

2.2.2 Переваги і особливості обраних алгоритмів

Z-Buffer краще підходить у тих ситуаціях, коли:

– на сцені велика кількість об’єктів / полігонів;

– динамічна сцена (об’єкти чи камера рухаються / повертаються).

Особливість алгоритму Z-Buffer: Йому потрібен додатковий буфер у тому ж форматі, що й в намальованого зображення. Також алгоритм виконує одну дію на фрагмент (намальований піксель кожного полігону, навіть прихованого). Після рендерінга в розпорядженні мається координата z кожного видимого пікселя, яка необхідна для багатьох сучасних методів рендерінга.

Алгоритм Художника краще підходить у тих ситуаціях, коли:

– нерухомі об’єкти чи невелика кількість об’єктів / полігонів;

– якщо у распоряджені не багато пам'яті (в 8-бітних комп’ютерах більшість 3D додатків використовували алгоритм Художника тому що не було пам’яті для реалізації алгоритму Z-Buffer).

Особливість алгоритму Художника: йому потрібен індексний буфер для зберігання всіх індексів об'єкта / полігону і він виконує сортування при кожній динамічній зміні сцени (сортування багатьох об’єктів по координаті Z може бути більш повільним). Інколи ці дві техніки використовують разом.

2.3 Реалізація на Java

Даний розділ містить опис імплементації алгоритму видалення прихованих ліній. Однією з важливих частин є детальне описання процесу впровадження функціоналу в обраній IDE дотримуючись його архітектури. У вигляді фрагментів наведені приклади коду та надані відповідні коментарі.

2.3.1 Вибір програмного середовища

Оскільки я обрав мову програмування Java, то треба обрати й підходяще програмне середовище для реалізації алгоритмів. Гарна IDE сприяє покращенню розробки коду, додавання коментарів, підвищенню чистоти коду, тощо. Усяк краще, ніж працювати у консолі чи в блокноті.

Звісно, мій вибір пав на вже знайомі нам всім NetBeans та Eclipse.

Почнемо з NetBeans. NetBeans – це інтегроване середовище розробки, яке підтримує масу мов програмування, включаючи найбільш популярні, такі як Python, Java, C / C ++, обробку XML, взаємодію з базами даних та інші функції, характерні для сучасної IDE[16]. Його логотип – один із тих речей, які надовго запам’ятовуються. Логотип NetBeans 8.2(Рисунок 2.4).

Рисунок 2.4 – Логотип NetBeans 8.2.

Підтримка і вливання коштів на розробку відбувалося компанією Oracle, яка таким чином забирає собі у власність масу open-source продуктів. Реальна підтримка відбувається серед звичайних ентузіастів, які називають себе NetBeans Community, а також компанією, яка називається NetBeans Org. Як і в будь-який інший подібної системи, в NetBeans реалізована підтримка рефакторинга коду, його профілювання, а також колірне виділення і генерація ділянок коду. Ще одна схожість для всіх основних середовищ розробки – це необхідність попередньої установки Java Developer Kit для запуску NetBeans. IDE підтримує основні платформи для малих, середніх і великих підприємств: Java Enterprise та Standard Edition. З огляду на розвиток мобільних пристроїв, нові версії працюють і з платформою Java Micro Edition, яка призначена для створення додатків на девайсах, ресурси яких дещо обмежені. Дивлячись на істотне навантаження і концентрацію на більш комерційних проектах, компанія Oracle прийняла рішення про передачу NetBeans в руки іншої компанії. Так, починаючи з 2016 року, середовище розробки ПО підтримується фондом Apache Software Foundation.

Серед головних переваг цього програмного середовища[17]:

– встановлюй і користуйся. Якщо ви точно знаєте, якою мовою будете програмувати, то вкажіть необхідні пакети відразу при відкритті файлу інсталяції;

– open-source. Ніколи не зайве глянути, як все працює. Чи внести необхідні правки. До того ж, велика кількість таких же розробників готові в будь-яку хвилину подумати над запропонованою вами проблемою;

– спільна розробка. NetBeans має необхідні інструменти для створення додатків в команді. Таким чином, вона стає вибором, коли мова заходить про установку IDE для великих підприємств і просто взаємодії між програмістами в рамках одного проекту;

– відлагодження. Завдяки вбудованим рішенням NetBeans може швидко знайти помилки і ті місця, де код стає малопродуктивним. Причому, все це відбувається в режимі реального часу, без переривання штатної роботи програміста;

– широка підтримка мов. Як тільки ви вирішите написати щось на іншій мові – просто укажіть це під час створення нового проекту. Немає потреби використовувати додаткове програмне забезпечення.

Загальним недоліком для всіх середовищ, які запускаються через JDK / SDK буде істотне споживання ресурсів пристрою. Як Eclipse так і NetBeans не застраховані від цього.

На цей час я використовую Apache NetBeans IDE 12.1. Зовнішний вигляд ви можете побачити на рисунках нижче. Картинка при запуску(дивись рисунок 2.5) та сам інтерфейс, який бачить програміст(дивись рисунок 2.6).

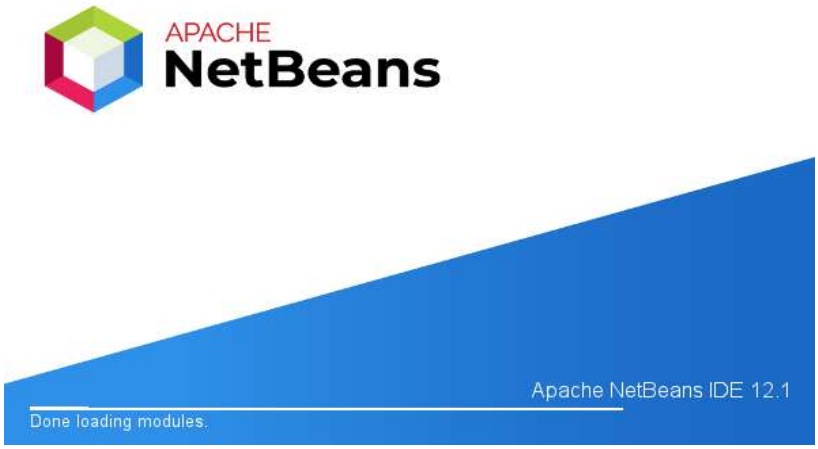

Рисунок 2.5 – Картинка при запуску Apache NetBeans

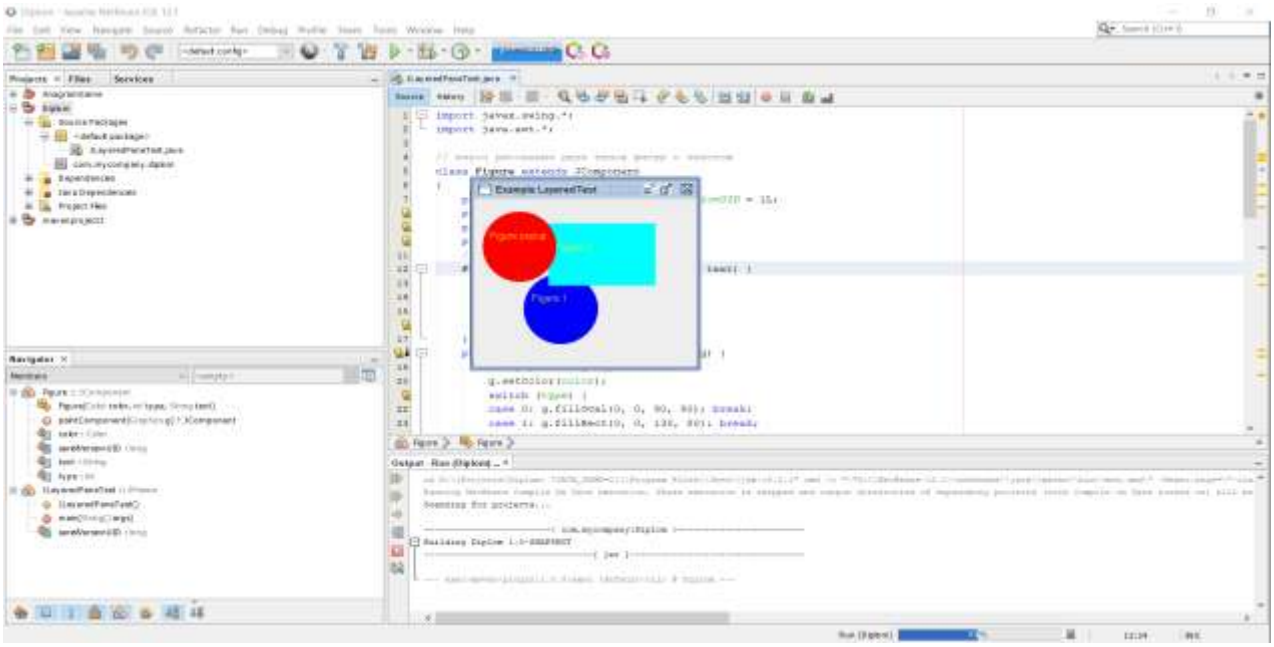

Рисунок 2.6 – Інтерфейс Apache NetBeans

Не будемо забувати й про IDE Eclipse**.** Eclipse[18] був розроблений IBM. На початку 2000-х, коли відбувалося сходження майбутньої зірки від Microsoft (Visual Studio), компанія International Business Machines вирішила зробити свою відповідь на це програмне забезпечення. Лого Eclipse можно побачити тут (дивись рисунок 2.7).

Рисунок 2.7 – Лого Eclipse

Уже кілька років поспіль Eclipse не належить IBM. Над цим проектом працює Eclipse Foundation. Це колаборація IT-компаній. Серед гучних імен

тут можна назвати Google, Oracle, IBM. А з недавнього часу навіть Microsoft вирішили приєднатися до розробки цього програмного забезпечення.

Серед головних переваг цього програмного середовища:

– Найчастіше на Eclipse переходять, коли необхідна розробка зовнішнього вигляду програми. Для цього служить бібліотека Swing. До того ж, Eclipse оснащена Standard Widget Toolkit;

– Часто рятує і документація Eclipse, яка розвинена не так добре в NetBeans. У ній і справді можна знайти відповіді на більшість питань, що цікавлять. Та й кількістю людей не сильно поступається – інші програмісти завжди прийдуть на допомогу.

На цей час я використовую Eclipse 4.17.0. Також ви можете ознайомитися з картинкою при запуску (дивись рисунок 2.8) та інтерфейсом Eclipse (дивись рисунок 2.9).

Рисунок 2.8 – Картинка при запуску Eclipse

Врешті решт я обрав Eclipse для реалізації свого проекту через надані IDE переваги, Проте головною причиною стало те, що у мене були деякі проблеми з буфером у IDE Apache NetBeans .

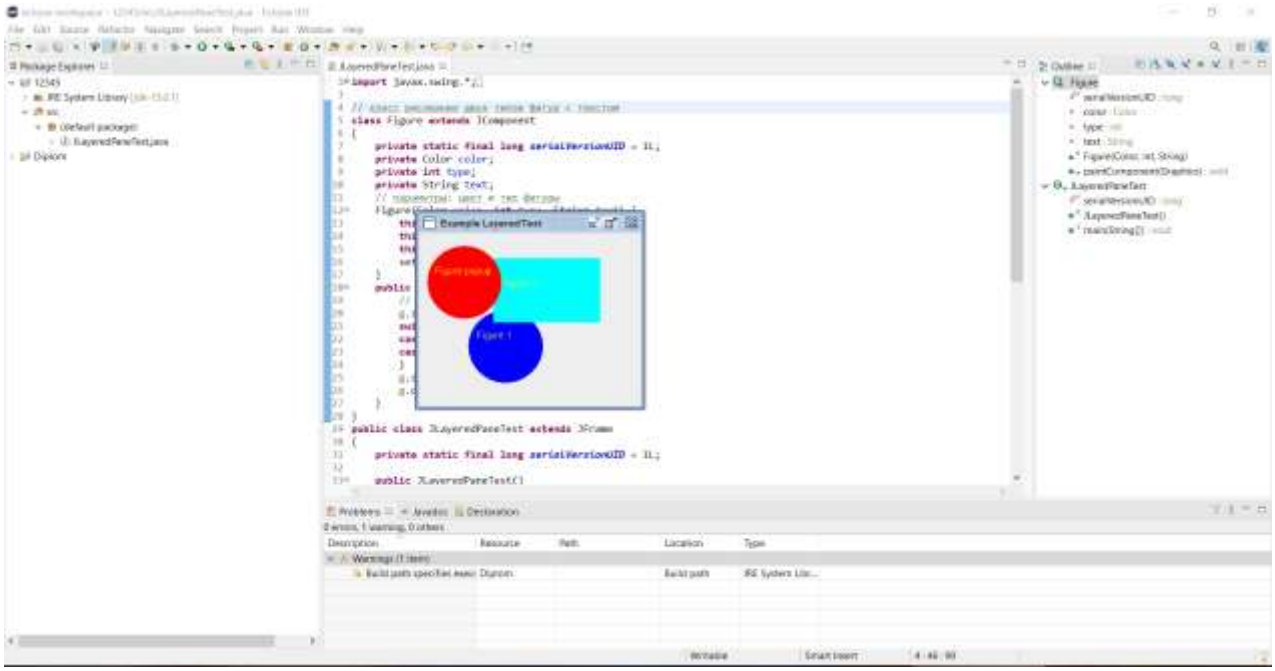

Рисунок 2.9 – Інтерфейс Eclipse

2.3.2 Програмна реалізація

Для створення нового алгоритму видалення прихованих ліній та поверхонь було створено п’ять класів та один інтерфейс – `ZBuffer`, `ZBufferSort`, `ValueResolver`, `ValueWorker`, `ZBufferMain` та `ResolverInterface`. Спершу приділю увагу інтерфейсу `ResolverInterface` – він містить у собі параметр для отримання координат x та y. Клас `ValueResolver`, що імплементує `ResolverInterface`, використовує класс `ValueWorker` для того, щоб отримати координату Z фігури. `ZBuffer` – це клас, що імпементує `Composite` та містить функції для роботи з Z-буфером, а клас `ZBufferSort` імпементує `CompositeContext` та виконує присвоєння по Z-координати нового кольору пікселя. Останній клас `ZBufferMain` – це main клас, де створюється `JFrame` і задаються параметри вікна та подаются координати фігур для обробки до класів `ZBuffer` та `ValueWorker`.

Розглядаючи програмну реалізацію на більш детальному рівні, інтерфейс `ResolverInterface` складається всього з `double resolve(double x, double y);` Повний код класу:

```
public interface ResolverInterface {
double resolve(double x, double y);
}
```

У свою чергу клас `ValueResolver` має свою `protected double[] plane` та декілька методів для роботи з площиною. Метод `ValueResolver` може викликатися за допомогою координатів фігур, тоді він буде створювати площину за допомогою класу `ValueWorker` або, якщо він буде отримувати площину, то буде застосовувати до неї свій наступний метод `setPlane`. Цей метод спочатку перевірить довжину площини і якщо вона буде надто короткою то видасть `IllegalArgumentException`. Потім присвоїть подану `plane` до своєї. І останній `resolve` метод повинен отримати double x і double y та повернути `ValueWorker.resolveZ(x, y, plane)`. Повний код класу `ValueResolver` наведений у **Додатку А**.

Наступний клас який я опишу – `ValueWorker`. Він містить три методи, що опрацьовують подані дані. Перший метод це `createPolygonResolver` який створює елемент методу `ValueResolver` класу `ValueResolver` та повертає його. Другий метод називається `createPlane`. Він за допомогою рівняння площини по трьом точкам[19] отримує і повертає площину. Останній метод `resolveZ` отримує Z-координату та повертає її. Повний код класу `ValueWorker` наведений у **Додатку Б**.

Клас `ZBuffer` містить свої методи та змінні. До змінних належать:

- `buffer` – використовується як Z-буфер;
- `clearBuffer` – використовується для очистки змінної `buffer`;
- `int width` та `height` вікна;
- `ResolverInterface` для `ResolverInterface`.

До методів належать: `ZBuffer`, `setSize`, `createContext`, `clearBufferBit`, `setZOf`, `getBuffer`, `setResolverInterface`,

getResolverInterface, getZOf. Метод ZBuffer отримує width та height, збільшує width на 1 та передає їх у метод setSize. Він в свою чергу ініціює buffer та clearBuffer. Метод createContext використовує ZBufferSort класу ZBufferSort для обробки та повернення даних. Метод clearBufferBit копіює у buffer Double.MAX_VALUE з clearBuffer. Метод setZOf встановлює значення Z-координати. Метод getBuffer повертає buffer. Метод setResolverInterface встановлює ResolverInterface. Метод getResolverInterface повертає ResolverInterface. Метод getZOf повертає значення Z-координати. Повний код класу ZBuffer наведений у **Додатку В.**

Клас ZBufferSort містить чотири змінні та три метода. До змінних належать RED, GREEN, BLUE та ZBuffer. До функцій належать ZBufferSort що викликає Z-буфер, compose і dispose для нього. У методі compose і відбувається уся магія: він отримує дані фігури та Z-буфера, порівнює їх між собою та якщо це порівняння показує, що новий піксель розташований перед пікселем, що знаходиться в буфері кадру, то новий піксель рисується поверх старого. Повний код класу ZBufferSort наведений у **Додатку Г**.

Клас ZBufferMain містить main клас. У ньому створюються **JPanel** та MouseAdapter, який дозволяє відстежувати положення миши та заносить ці дані у назву вікна. У саму JPanel вносятся дані як розмір вікна, розмір Z-буферу та координати фігур, їх колір, тощо. Код класу ZBufferMain, який буде малювати дві фігури наведений у **Додатку Д**.

2.4 Інструкція до додатку

Даний розділ присвячений інструкції до розробленого додатку з особливостями редагування коду.

2.4.1 Підготовка робочого місця

Для запуску цього додатку спочатку треба встановити JDK 15.0.1 чи вище. Потім встановити IDE для роботи з додатком (я надаю перевагу Eclipse). Після запуску та налаштування IDE[20], створіть новий проект, створіть у ньому новий package якщо він не був створений за замовчуванням, створіть п’ять Java файлів у цьому package. У кожний файл внесіть код із додатку. Наприклад, для першого файлу внесіть код із додатку А, до другого файлу – із додатку Б. Заповніть так усі файли. Потім виправте назву файлів, щоб назва файлу була такою ж як і назва класу у цьому файлі. Це можно зробити за допомогою IDE – вона відмітить не підходящі місця як помилки і клацнувши на них буде відкриватися контекстне меню, де можно буде обрати варіант з виправленням (дивись рисунок 2.10). Потім натисніть зелену кнопку “Run” або Ctrl+F11 для запуску додатку.

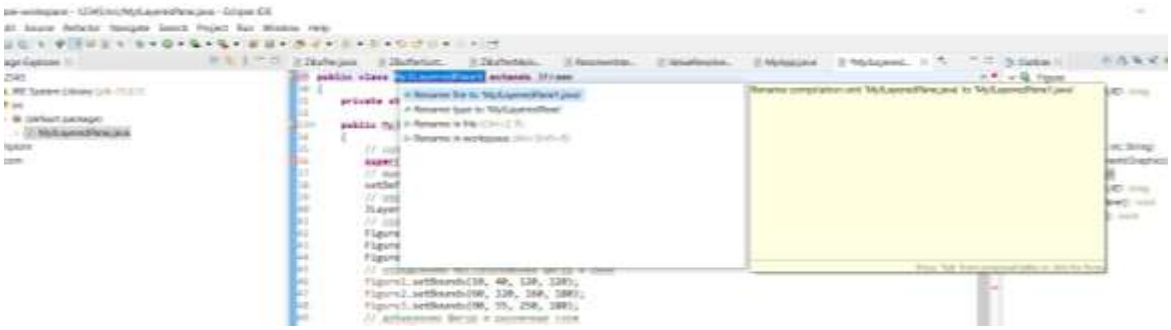

Рисунок 2.10 – Інструкції до використанні у Eclipse

2.4.2 Редагування коду

Якщо Ви бажаєте редагувати код або змінити параметри фігур, то спершу Вам необхідно перейти до класу `ZBufferMain`. Там створюється `JPanel` і вказуються дані вікна й фігур. Зазначу, що щоб марно не витрачати ресурси пропоную вказувати розмір Z-буферу ідентично розміру вікна. Розмір Z-буферу вказується у строчці `new ZBuffer(640, 480)`, а розмір

вікна `frame.setSize(640, 480)`. Щоб змінити розмір або форму фігури треба змінити `xpoints` та `ypoints`. Строка нище подає ці дані до обробки. Якщо змінити дані після `xpoints` та `ypoints` то можно змінити положення фігури у просторі (Z-координату).

```
composite.setResolverInterface(ValueWorker.createPolygonResolver
(  xpoints, ypoints, 100, 0, 0))
```

Ці дві строки відповідають за заливку об’єкта червоним кольором:

```
g2d.setColor(Color.red);
g2d.fillPolygon(xpoints, ypoints, xpoints.length);
```

Ці дві строки відповідають за колір контуру об’єкта:

```
g2d.setColor(Color.yellow);
g2d.drawPolygon(xpoints, ypoints, xpoints.length);
```

Якщо Ви бажаєте додати нову фігуру, то Вам треба додати до коду ці строки, починаючи з `xpoints` і закінчуючи `g2d.drawPolygon`. Якщо бажаєте, то можете скопіювати вже існуючу частину коду і змінити необхідні параметри.

2.5 Висновок

У цьому розділі було обрано мову програмування для реалізації додатку, був огляд обраної мови програмування, були розглянуті дві IDE та було обране середовище реалізації додатку, був огляд та порівняння алгоритмів Z-буфера та Художника й була зроблена та детально розглянута реалізація додатку з написанням до нього інструкції.

3 ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА АЛГОРИТМУ ВИДАЛЕННЯ ПРИХОВАНИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Цей розділ присвячений аналізу розроблених алгоритмів видалення прихованих ліній та поверхонь, описанню проведеного експерименту, його результатам та порівняння цих результатів з визначеним еталоном.

3.1 Аналіз часу роботи розробленого алгоритму

Метою експерименту є перевірка гіпотези про ефективність розробленого алгоритму. Основою перевірки є оцінка часу роботи алгоритму при відображенні трьох фігур. Малюнок трьох фігур буде представлений нижче

Апаратна та програмна середа виконання експерименту:

- Процесор: Intel (R) Core (TM) i7-6700HQ CPU 2.60 GHz;
- Оперативна пам’ять: 16,00 Гб;
- Тип системи: 64-розрядна ОС;
- Відеоадаптер: Intel(R) HD Graphics 530;
- IDE: Eclipse.

Для перевірки часу роботи розробленого алгоритму до коду було додано декілька `System.currentTimeMillis()` для отримання часу та `System.out.println` для його відображення. Нижче приведено результати перевірки часу роботи розробленого алгоритму (дивись рисунок 3.1).

Рисунок 3.1 – Результати перевірки часу роботи розробленого алгоритму

Одразу можно помітити, що методи у додатку витрачають значно відрізняючийся час на виконання своїх задач. Особливо багато часу витрачається на ініціалізацію графічної частини (`JFrame`) та її відображенні. Також хочу звернути увагу на збільшене витрачання часу для малювання першої фігури.

3.2 Аналіз відображення фігури розробленого алгоритму

Крім алгоритму, що виконує розрахунки для Z-буферу, треба оцінити якість малюванні фігур, тобто кінцевий результат виконання розробленого додатку. Нище представлений результат малювання фігур, наведених у коді додатку Д (дивись рисунок 3.2).

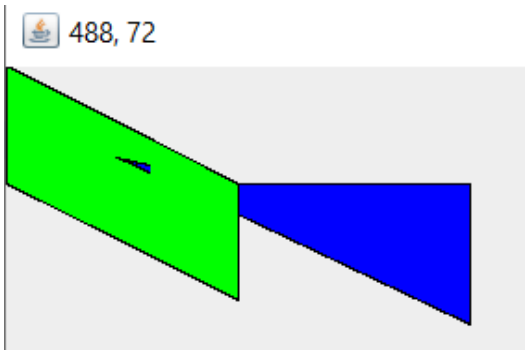

Рисунок 3.2 – Результат роботи розробленого алгоритму

Тепер спробуємо відобразити дві площини, що перетинають одна одну, наче піла ріже дерево (дивись рисунок 3.3).

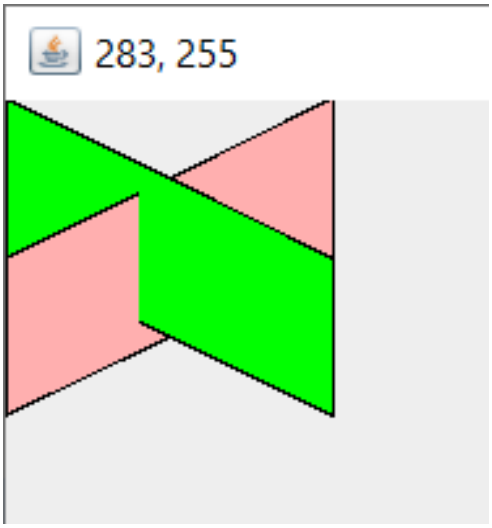

Рисунок 3.3 – Результат роботи розробленого алгоритму

Тепер додамо до зображення третю фігуру, що вже використовувалася вище. Саме на цьому коді з трьома фігурами (дивись рисунок 3.4) я й проводив аналіз часу роботи алгоритму у пункті 3.2.

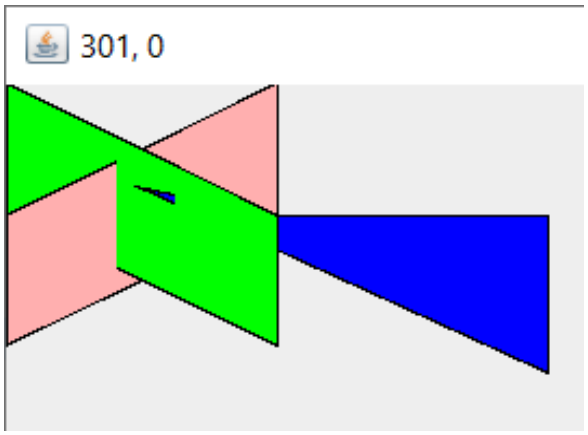

Рисунок 3.4 – Результат роботи розробленого алгоритму

Тепер зробимо так, щоб наш синій трикутник проткнув дві площини та подивимося на результат роботи алгоритму (дивись рисунок 3.5).

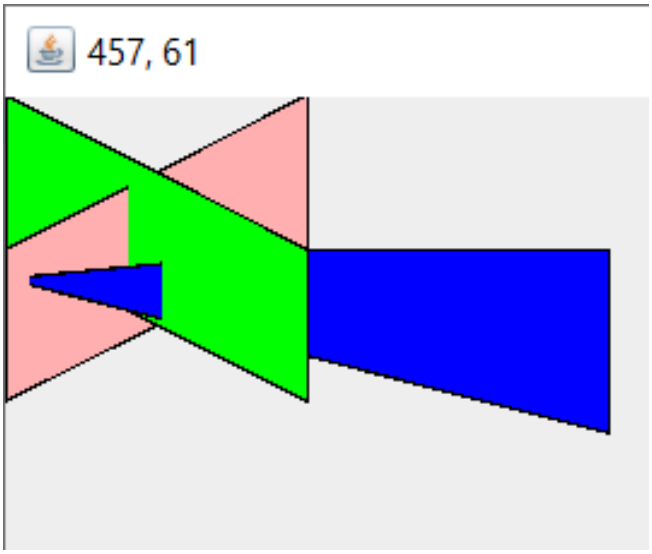

Рисунок 3.5 – Результат роботи розробленого алгоритму

3.2.1 Визначення еталонної оцінки відображення фігури

Простим способом оцінки правильності рішення є порівняння його результатів з еталонною оцінкою. Для завдання видалення прихованих ліній та поверхонь на даний момент еталонною оцінкою є малювання вручну або за допомогою спеціалізованих програм та допомогою експертів предметної галузі. Нижче представлений приклад роботи одного з експертів[21] (дивись рисунок 3.6).

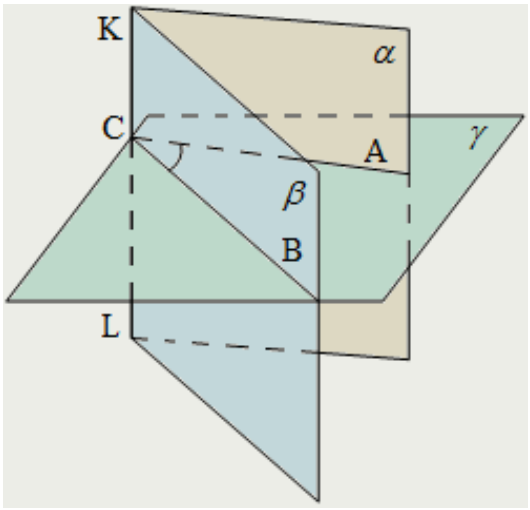

Рисунок 3.6 – Робота експерта

3.3 Результати експеримента та їх аналіз

У даному розділі ми проаналізуємо вже розроблені алгоритми, проаналізуємо отриманні результати при видаленні прихованих ліній та поверхонь та надамо оцінку виконаних дій при проведенні експерименту.

3.3.1 Аналіз реалізованих алгоритмів видалення прихованих ліній та поверхонь

Побачивши теоретичні відомості про алгоритми прихованих ліній та поверхонь, настав час приділити увагу й практичній частині. Нище ми проаналізуємо роботу експертів предметної галузі, а саме розроблені ними додатки або представлений ними код.

Встановлення критеріїв для порівняння алгоритмів видалення прихованих ліній та поверхонь. Спершу треба визначити критерії аналіза для обраних алгоритмів. Ми не будемо далеко відходити від канонів, тому візьмемо за критерії наступне:

- складність алгоритму;
- час виконання;
- якість вихідного зображення.

Треба зазначити, що якість вихідного зображення буде оцінюватися як результат роботи алгоритму видалення прихованих ліній, тобто на зображенні повинно бути видно всі видимі частини, а невидимі – не видно. Критерії будуть оцінюватися за можливістю.

Реалізація z-буферу. Спочатку розглянемо приклад реалізації[22] z-буферу на C++(дивись рисунок 3.7). Завдяки додатковим даним цієї реалізації ми можемо більш детально ознайомитися з результатами роботи алгоритму z-буферу.

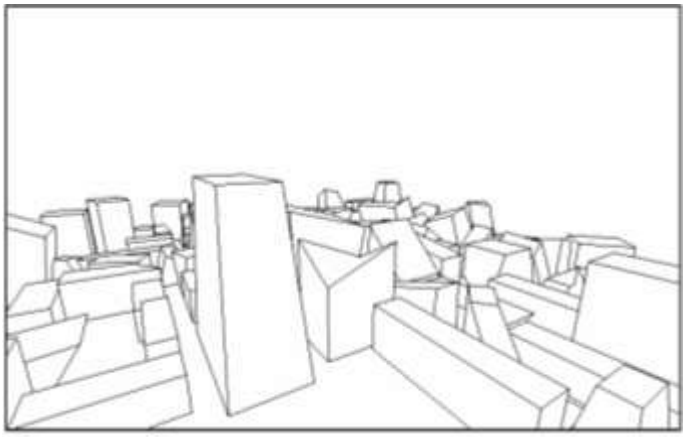

Рисунок 3.7 – Приклад роботи алгоритму z-буферу

Аналізуючи вихідне зображення реалізованого алгоритму можно сказати, що воно задовольняє встановленим критеріям. Роздивляючись цю програму більш детальніше, подивимося на складність роботи розробленого додатку (дивись рисунок 3.8).

Количество пикселей в изображении	Количество граней в сцене				
	113	229	348	463	581
1749	112374	135622	221268	268727	337186
2560	152924	199238	357141	409090	533160
4000	168974	299221	456591	646327	772428
6996	263801	510177	813610	1045630	1309974
16000	584997	1169266	1833199	2453510	3045587

Рисунок 3.8 – Складність роботи алгоритму z-буферу

За даними з рисунку 3.8 можна побачити, що кількість пікселів у зображенні значно впливає на кількість граней в сцені (дивись рисунок 3.9).

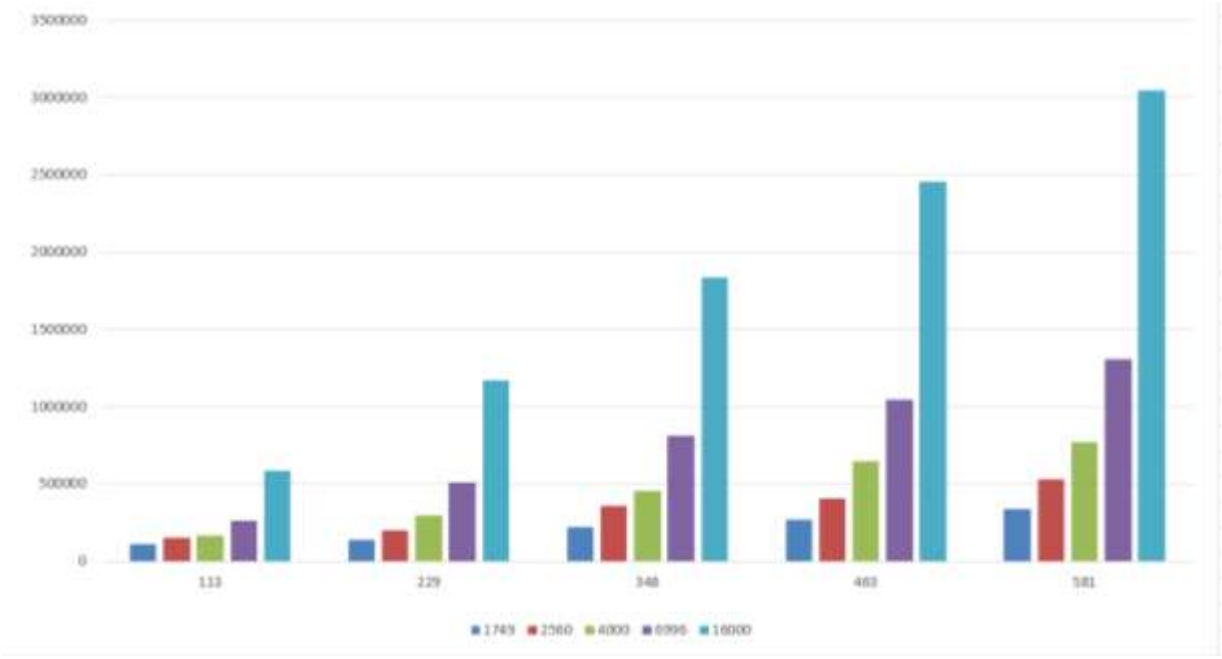

Рисунок 3.9 – Залежність пікселів від кількості граней на сцені

Далі розглянемо час роботи алгоритму. Одиниця виміру – одна тисяча тактов процесору. Нище зображена залежність часу роботи алгоритму від кількості граней на сцені(дивись рисунок 3.10).

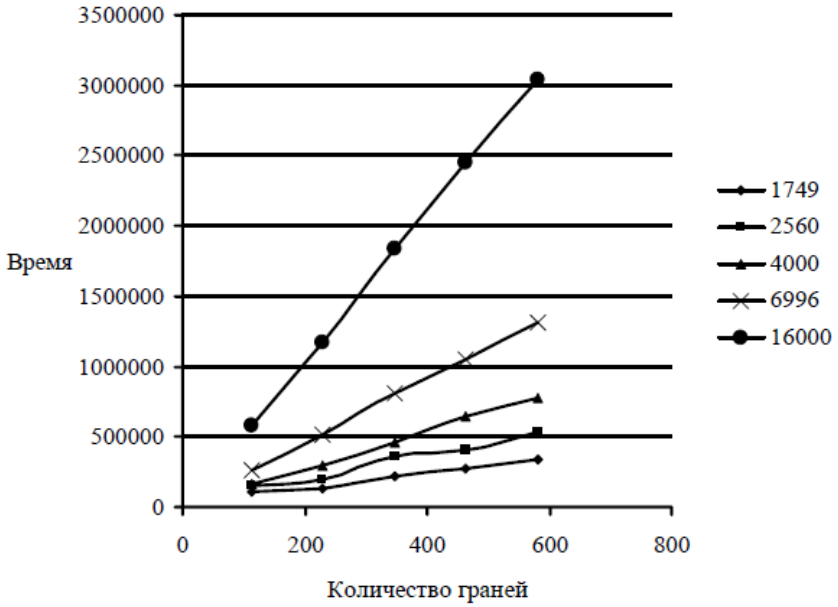

Рисунок 3.10 – Залежність часу роботи від кількості граней на сцені

Тут можно побачити значне підвіщеня потребуємого часу для кількості пікселів вище 4000.

Реалізація алгоритму Художника. Наступний реалізований алгоритм, який ми розглянемо – це алгоритм Художника[23], написаний на C (дивись рисунок 3.11).

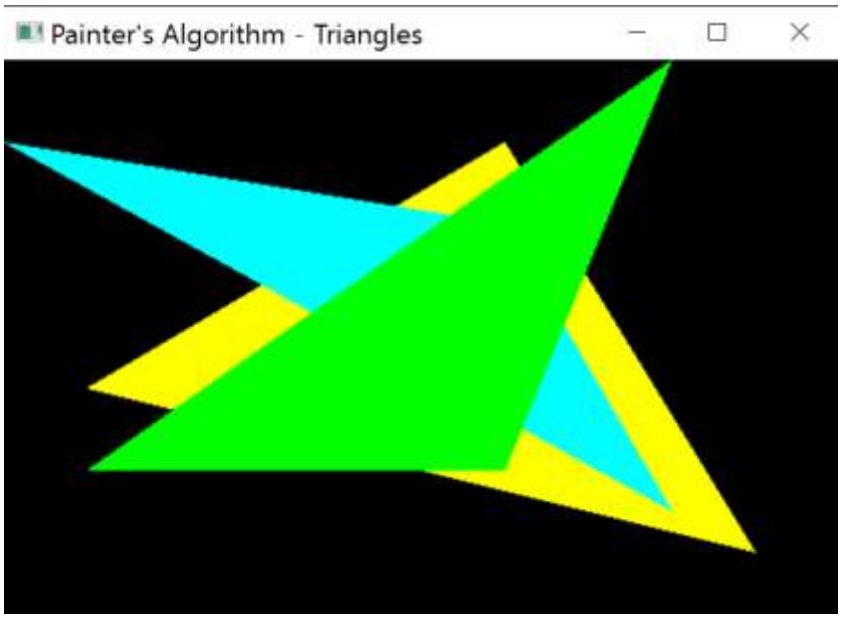

Рисунок 3.11 – Приклад роботи алгоритму Художника

Аналізуючи вихідне зображення реалізованого алгоритму можно сказати, що воно задовольняє встановленим критеріям. Одразу же перейдем до розглядання часу роботи алгоритму. Експерт пропонує нам розглядати роботу його алгоритму у секундах, чого ми і будемо притримуватися(дивись рисунок 3.12).

```
$ ./pixel
10 frame buffers of size 10 x 10 took 0.000000 seconds to update
10 frame buffers of size 100 x 100 took 0.015000 seconds to update
10 frame buffers of size 1000 x 1000 took 1.016000 seconds to update
10 frame buffers of size 10000 x 10000 took 158.156006 seconds to update
```

Рисунок 3.12 – Результати тесту алгоритму Художника

Тут також можна побачити значне збільшення витрачання часу з кожним збільшенням розміру буферу. Графік нище це відображає(дивись рисунок 3.13).

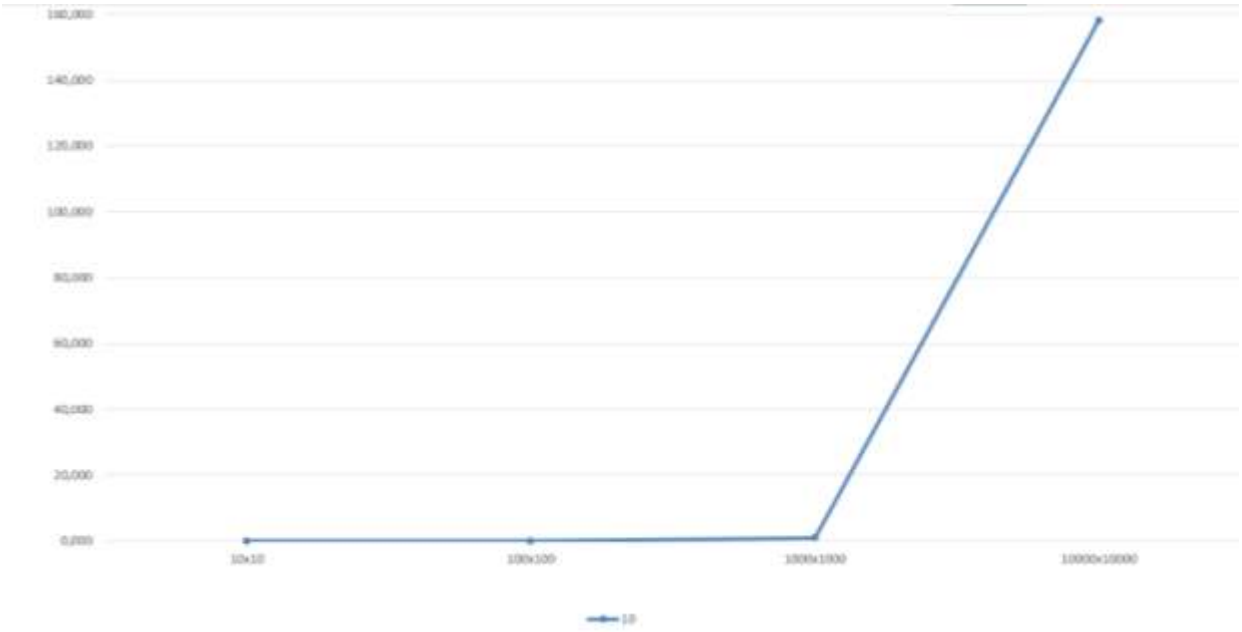

Рисунок 3.13 – Результати тесту алгоритму Художника

Проте, на мою думку, він повинен бути більше похожим на віддзеркалений y = 1/x. Що стосується складності алгоритму, то вона дорівнює O(N+C), де N – кількість граней, C – кількість пікселів.

Реалізація алгоритму Робертса. Останній реалізований алгоритм, який ми розглянемо – це алгоритм Робертса[24], написаний на JavaScript (дивись рисунок 3.14).

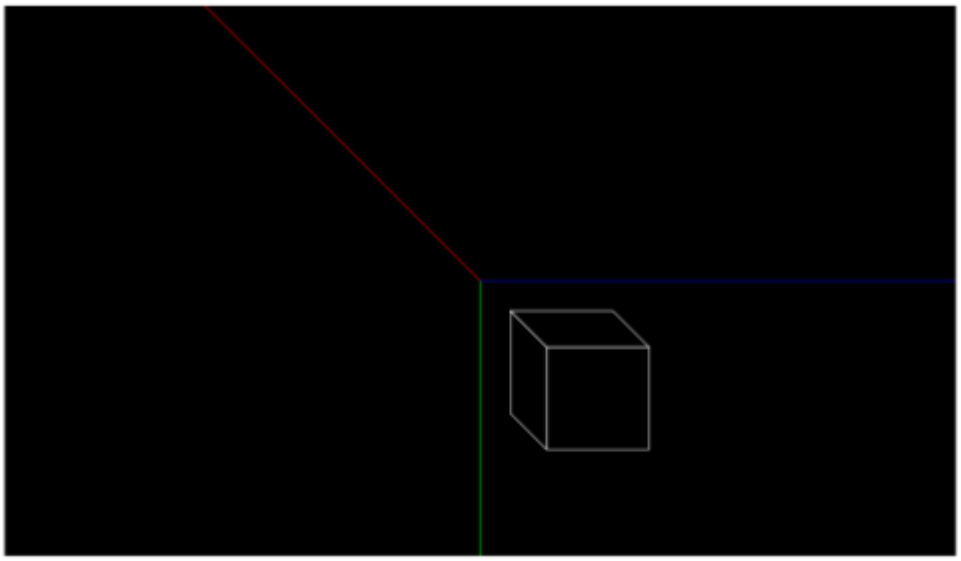

Рисунок 3.14 – Приклад роботи алгоритму Робертса

Аналізуючи вихідне зображення реалізованого алгоритму можно сказати, що воно задовольняє встановленим критеріям. Як і з іншими, ми будемо тестувати швидкість роботи реалізованого алгоритму. Для цього ми використаємо “PageSpeed Insights”, що розроблений компанією Google(дивись рисунок 1.16).

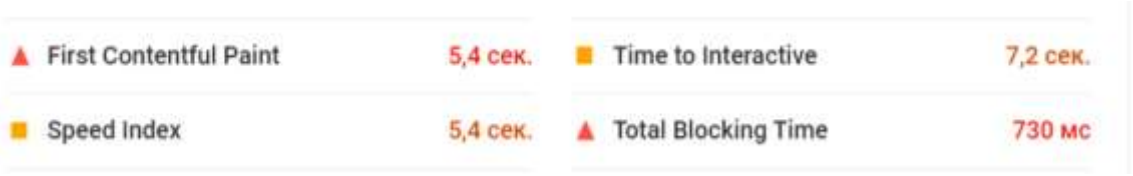

Рисунок 3.15 – Результати тесту алгоритму Робертса

Результат тесту показує, що є значна затримка відображення першого елемента. Також, між появою першого і останнього елемента проходить деякий час. Що стосується складності алгоритму, то вона дорівнює O(N^3), де N – кількість граней.

Результат аналізу реалізованих алгоритмів видалення прихованих ліній. Як результат аналізу, пропоную подивитися створену таблицю з результатами аналізу реалізованих алгоритмів (дивись таблицю 1.2).

Таблиця 1.2 – Результат аналізу алгоритмів

Умова\Алгоритм	Z-буферу	Художника	Робертса
Де рекомендовано використовувати	Для відображення сцен та об'єктів, якщо маємо високий об’єм пам’яті	Для відображення декількох графічних об’єктів або сцени з об’єктами	Для відображення одного об'єкта 3д графіки у веб
Переваги використання	Може працювати з великими об’ємами даних	Може швидко зобразити статичну сцену з кількома фігурами	Працює швидко та може застосовуватися у веб
Недоліки використання	Потребує додатковий об’єм пам’яті	Працює повільно з великим буфером даних	Ефективно відображає лише одну фігуру

3.3.2 Аналіз візуального представлення після застосування розробленого алгоритму видалення прихованих ліній та поверхонь

Що стосується загального результату після застосування алгоритму, його можна вважати позитивним. Також проаналізувавши вихідний малюнок можна сказати що результат відображення задовольняє критеріям, проте не є ідеальним. Можна побачити на рисунках деякі відхилення у малюванні кінця трикутника. Основною задачею на сьогодні є виправлення таких помилок.

3.4 Висновок

У цьому розділі був проведений експеримент з перевірки проекту. Метою експерименту була оцінка часу роботи розроблених алгоритмів та оцінка зовнішнього вигляду нарисованої фігури. Була знайдена робота експерта для проведення порівняння та були показані результати роботи розробленого алгоритму, був проведений аналіз результатів роботи, був зроблений аналіз вже створених та реалізованих алгоритмів, результатів їх роботи.

4 ВИСНОВКИ ТА МАЙБУТНЯ РОБОТА

Ця кваліфікаційна робота магістра була присвячена розробці алгоритму видалення прихованих ліній. Вибір такого напрямку роботи обумовлений його актуальністю та необхідністю нових розробок та досліджень у галузі.

У першому розділі був зроблен огляд існуючих рішень видалення прихованих ліній та поверхонь. Ми подивилися на те, як інші фахівці мають справу з цією проблемою та порівняли відомі алгоритми.

У другому розділі було обрано мову програмування для реалізації додатку, був огляд обраної мови програмування, були розглянуті дві IDE та було обране середовище реалізації додатку, був огляд та порівняння алгоритмів Z-буфера та Художника й була зроблена та детально розглянута реалізація додатку з написанням до нього інструкції.

Третій розділ був присвячений експерименту, спрямованому на оцінку додатку. Метою експерименту була оцінка часу роботи розробленого алгоритму та оцінка зовнішнього вигляду нарисованої фігури. Була знайдена робота експерта для проведення порівняння та були показані результати роботи розробленого алгоритму, був проведений аналіз результатів роботи, був зроблений аналіз вже створених та реалізованих алгоритмів, результатів їх роботи.

У четвертому розділі розглядається майбутня робота.

У результаті проведених експериментів були визначені сильні та слабкі сторони алгоритму та його реалізації. Також був наданий список потенційних вдосконалень для алгоритму та список розширень. Грунтуючись на результатах експериментальної частини можна зазначити, що алгоритм справляється з функцією видалення прихованих ліній та поверхонь, але ще не є досконалим.

Дивлячись на кінцевий результат можна стверджувати, що мета даної дипломної роботи була досягнута, але це не є фінальною стадією для програмного продукту. При подальшому розвитку цього додатку, першим буде вдосконалення алгоритму малювання шляхом додавання різних графічних покращень у відображення фігур. Також обов’язковим покращенням буде виправлення проблеми кінця трикутнику, яка іноді виникає.

ПЕРЕЛІК ПОСИЛАНЬ

1. А. Ю. Дёмин, А. В. Кудинов, Компьютерная графика, Томский политехнический университет, 2005
2. Основы интерактивной машинной графики (1976)
3. Sutherland, Ivan E., Sproull, Robert F., and Schumacher, R.A., "A Characterization of Ten Hidden-Surface Algorithms" Computing surveys 1974
4. Алгоритм плаваючого горизонту. URL: https://studopedia.com.ua/1_275247_algoritm-plavayuchogo-gorizontu.html (дата звернення: 04.11.2020)
5. Никулин, Е. А. Компьютерная геометрия и алгоритмы машинной графики / Е. А. Никулин. СанктПетербург : БХВ-Петербург, 2003. с558
6. Catmull, Edwin, "Computer Display of Curved Surfaces" Proc. IEEE conf. comput. grafics pattern recognition data struct. May 1975 с 11
7. Роджерс Д.Ф. Алгоритмические основы машинной графики. (Procedural Elements for Computer Graphics) Учебное издание. Перевод с английского С.А. Вичеса, Г.В. Олохтоновой, П.А. Монахова под редакцией Ю.М. Банковского, В.А. Галактионова 530
8. Apurva A. Desai Computer Graphics с258
9. Newell, M.E., Newell R.G., and Sancha, T.L., "A New Approach to the Shaded Picture Problem" Proc. ACM natl. conf. 1972 с450
10. Warnock, J.E., "A Hidden Surface Algorithm for Computer Generated Halftone Pictures", Dept. Comp. Sci., U. of Utah, (1969).
11. Hidden surface removal using polygon area sorting by Kevin Weiler and Peter Atherton, Program of Computer Graphics, Cornell University, Ithaca, New York 14853
12. Java. URL: https://www.java.com/en/ (дата звернення: 10.11.2020)
13. Алгоритм Z-буфера. URL: https://studbooks.net/2248058/informatika/otobrazhenie_okne (дата звернення: 06.11.2020)
14. Алгоритм использующий Z-буфер. URL: https://poisk-ru.ru/s19673t4.html (дата звернення: 06.11.2020)
15. BSP-дерево. URL: https://neerc.ifmo.ru/wiki/index.php?title=BSP-дерево (дата звернення: 09.11.2020)
16. Apache NetBeans. URL: https://netbeans.apache.org/ (дата звернення: 09.11.2020)
17. Битва титанов open-source: NetBeans или Eclipse. URL: https://webformyself.com/bitva-titanov-open-source-netbeans-ili-eclipse/ (дата звернення: 10.11.2020)
18. The Eclipse Foundation. URL: https://www.eclipse.org/ (дата звернення: 10.11.2020)

19. Прямая, плоскость и их уравнения. URL: http://www.cleverstudents.ru/line_and_plane/ (дата звернення: 14.11.2020)
20. Первая программа в Eclipse. URL: https://metanit.com/java/tutorial/1.4.php (дата звернення: 14.11.2020)
21. Прямые и плоскости. URL: http://math4school.ru/priamye_i_ploskosti.html (дата звернення: 14.11.2020)
22. Польский, С. В. Компьютерная графика : учебн.-методич. пособие. – М.:ГОУ ВПО МГУЛ, 2008. – 38 с
23. CISC 360 Project. URL: https://github.com/McMerrison/CISC360Project (дата звернення: 30.11.2020)
24. ComputerGraphics. URL: https://github.com/BorZzzenko/ComputerGraphics (дата звернення: 30.11.2020)

25.

ДОДАТОК А

ValueResolver.java

```
public class ValueResolver implements ResolverInterface {
	protected double[] plane;
	public ValueResolver() {
	}
	public ValueResolver(double x1, double x2, double x3, double y1, double
y2, double y3, double z1, double z2, double z3) {
		this(ValueWorker.createPlane(x1, x2, x3, y1, y2, y3, z1, z2, z3));
	}
	public ValueResolver(double[] plane) {
		setPlane(plane);
	}
	public void setPlane(double[] plane) {
		if (plane.length != 4) {
			throw new IllegalArgumentException("");
		}
		this.plane = plane;
	}
	public double resolve(double x, double y) {
		return ValueWorker.resolveZ(x, y, plane);
	}
}
```

ДОДАТОК Б

ValueWorker.java

```
public class ValueWorker {
	public static ResolverInterface createPolygonResolver(final int[] xpoints,
final int[] ypoints, double z1, double z2, double z3) {
		ValueResolver resolver = new ValueResolver(
				xpoints[0], xpoints[1], xpoints[2],
				ypoints[0], ypoints[1], ypoints[2],
				z1, z2, z3);
		return resolver;	}
	public static double[] createPlane(double x1, double x2, double x3,
			double y1, double y2, double y3,
			double z1, double z2, double z3) {
		double[] plane = new double[4];
		plane[0] = y1*(z2 -z3) + y2*(z3 -z1) + y3*(z1 -z2);
		plane[1] = z1*(x2 -x3) + z2*(x3 -x1) + z3*(x1 -x2);
		plane[2] = x1*(y2 -y3) + x2*(y3 -y1) + x3*(y1 -y2);
		plane[3] = -( x1*(y2*z3 -y3*z2) + x2*(y3*z1 -y1*z3) + x3*(y1*z2 -
y2*z1));
		return plane;}
	public static double resolveZ(double x, double y, double[] plane) {
		if (plane[2] == 0) {
			return Double.MAX_VALUE;
		}
		return - (plane[0] * x + plane[1] * y + plane[3]) / plane[2];
```

```
}}
```

ДОДАТОК В

ZBuffer.java

```
import java.awt.Composite;
import java.awt.CompositeContext;
import java.awt.RenderingHints;
import java.awt.image.ColorModel;
import java.util.Arrays;
public class ZBuffer implements Composite {
	protected double[] buffer;
	protected double[] clearBuffer;
	protected int width;
	protected int height;
	protected ResolverInterface  ResolverInterface;
	public ZBuffer(int width, int height) {
		this.width = width +1;
		setSize(width, height);     }
	public void setSize(int newWidth, int newHeight) {
		if (newWidth == width && newHeight == height) {
			return;                  }
		this.width = newWidth;
		this.height = newHeight;
		buffer = new double[height * width];
		clearBuffer = new double[height * width];
		Arrays.fill(clearBuffer, Double.MAX_VALUE);
	}
```

```
	public CompositeContext createContext(ColorModel srcColorModel,
			ColorModel dstColorModel, RenderingHints hints) {
		return new ZBufferSort(this);
	}
	public void clearBufferBit() {
		System.arraycopy(clearBuffer, 0, buffer, 0, buffer.length);
	}
	public void setZOf(int x, int y, double value) {
		if (x >= width || x < 0 ||
			y >= height || y < 0) {
			throw new IllegalArgumentException("Point [" + x + ", " + y +
"] is outside of the Z Buffer array ["+width+", "+height+"]");
		}
		buffer[y*width + x] = value;
	}
	public double[] getBuffer() {
		return buffer;
	}
	public void setResolverInterface(ResolverInterface  ResolverInterface) {
		this.ResolverInterface = ResolverInterface;
	}
	public ResolverInterface getResolverInterface() {
		return ResolverInterface;
	}
	public double getZOf(int x, int y) {
		if (x >= width || x < 0 ||
				y >= height || y < 0) {
				throw new IllegalArgumentException("Point [" + x + ", "
+ y + "] is outside of the Z Buffer array ["+width+", "+height+"]");
		}
```

```
return buffer[y*width + x];        }
```

ДОДАТОК Г

ZBufferSort.java

```
import java.awt.CompositeContext;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;
public class ZBufferSort implements CompositeContext {
	protected final static byte R_BAND = 0;
	protected final static byte G_BAND = 1;
	protected final static byte B_BAND = 2;
	protected ZBuffer ZBuffer;
	ZBufferSort(ZBuffer ZBuffer) {
		this.ZBuffer = ZBuffer;
	}
	public void compose(Raster src, Raster dstIn, WritableRaster dstOut) {
		ResolverInterface  ResolverInterface  = ZBuffer.getResolverInterface
();
		if (ResolverInterface  == null) {
			throw new IllegalArgumentException("You must set a
ResolverInterface before draw any polygon with this composite");
		}
		int maxX = dstOut.getMinX() + dstOut.getWidth();
		int maxY = dstOut.getMinY() + dstOut.getHeight();
		for (int y = dstOut.getMinY(); y < maxY; y++) {
			for (int x = dstOut.getMinX(); x < maxX; x++) {
				int dstInX = -dstIn.getSampleModelTranslateX() + x;
```

```
                        int dstInY = -dstIn.getSampleModelTranslateY() + y;
                        double dstZ = ZBuffer.getZOf(dstInX, dstInY);
                        double srcZ = ResolverInterface.resolve(dstInX, dstInY);
                        if (srcZ < dstZ) {
                                ZBuffer.setZOf(dstInX, dstInY, srcZ);
                                dstOut.setSample(x, y, RED, src.getSample(x, y,
RED));
                                dstOut.setSample(x, y, GREEN, src.getSample(x,
y, GREEN));
                                dstOut.setSample(x, y, BLUE, src.getSample(x, y,
BLUE));
                        } else if (srcZ == dstZ) {
                                dstOut.setSample(x, y, RED, src.getSample(x, y,
RED));
                                dstOut.setSample(x, y, GREEN, src.getSample(x,
y, GREEN));
                                dstOut.setSample(x, y, BLUE, src.getSample(x, y,
BLUE));
                        } else {
                                dstOut.setSample(x, y, RED, dstIn.getSample(x, y,
RED));
                                dstOut.setSample(x, y, GREEN,
dstIn.getSample(x, y, GREEN));
                                dstOut.setSample(x, y, BLUE, dstIn.getSample(x,
y, BLUE));
                        }
                }
        }
    }
    public void dispose() {
```

```
}}
```

ДОДАТОК Д

ZBufferMain.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class ZBufferMain {
	public static void main(String[] args) {
		final JFrame frame = new JFrame("ZBuffer frame");
		JPanel bufferdPanel = new JPanel() {
			private static final long serialVersionUID = 1234567899;
			protected void paintComponent(Graphics g) {
				int[] xpoints;
				int[] ypoints;
				Graphics2D g2d = (Graphics2D) g;
				ZBuffer composite = new ZBuffer(640, 480);
				composite.clearBufferBit();
				g2d.setComposite(composite);
				xpoints = new int[] {
					0, 100, 100, 0 };
				ypoints = new int[] {
```

```
                0, 50, 100, 50                  };

        composite.setResolverInterface(ValueWorker.createPolygonResolver(xpoint
s, ypoints, 100, 0, 0));
                        g2d.setColor(Color.green);
                        g2d.fillPolygon(xpoints, ypoints, xpoints.length);
                        g2d.setColor(Color.black);
                        g2d.drawPolygon(xpoints, ypoints, xpoints.length);
                        xpoints = new int[] {
                                50, 200, 200, 100               };

                        ypoints = new int[] {
                                40, 110, 50, 50                 };
composite.setResolverInterface(ValueWorker.createPolygonResolver(xpoints,
ypoints, 5, 100, 100));
                        g2d.setColor(Color.blue);
                        g2d.fillPolygon(xpoints, ypoints, xpoints.length);
                        g2d.setColor(Color.black);
                        g2d.drawPolygon(xpoints, ypoints, xpoints.length);
                }
        };
        frame.getContentPane().add(bufferdPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(640, 480);
        frame.setLocationRelativeTo(null);
        frame.getContentPane().addMouseMotionListener(new
MouseAdapter() {
                public void mouseMoved(MouseEvent e) {
                        frame.setTitle(e.getX()+", " +e.getY());
                }
```

```
});
frame.setVisible(true);     }}
```