МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

Звіт про виконання лабораторних робіт
№1-4
з курсу «Комп'ютерна графіка»
Варіант К7

Лектор:
доцент. каф. пз
Левус Є. В.

Виконали студенти групи:
ПЗ-33
Лесневич Є. Є.
та Курганевич А.-М.А.

Перевірено:
асист. каф. пз
Коцун В. І.

«_____»_____2023 р.
∑=____ .....

Львів – 2023

Зміст

Завдання

Варіант К7.

1) Побудувати фрактальні зображення:

1.1) Різновиди фрактала Коха: «рандомізована» сніжинка, сніжинка «навпаки», острів.

1.2) Множини Мандельброта для 4 та 5 степеня z

2) Колірні моделі: RGB і XYZ. Змінити яскравість зображення. Робота з фрагментом зображення щодо перетворення моделі та зміни атрибуту кольору.

3) Реалізувати поворот трапеції відносно однієї з вершин проти та за годинниковою стрілки та одночасне пропорційне зменшення в N раз. Трапеція вводиться через вершини.

Теоретичні відомості

1. Опис функцій програми

1. Робота з фракталами
   1.1. Побудова фракталу Коха трьох видів: острів, рандомізована сніжинка, Сніжинка “навпаки” із заданою кількістю ітерацій (від 0 до 7) та обраним кольором.
   1.2. Побудова множини Мандельброта для заданого степеня (від 2 до 15) та заданої кількості ітерацій (від 1 до 100).
   1.3. Збереження (завантаження) зображення побудованого фрактала у форматі png, jpg, jpeg або jiff.
2. Робота із кольорами зображення
   2.1. Завантаження власного зображення для подальшого виконання над ним дій.
   2.2. Зміна яскравості (від -100 до 100) фрагменту зображення, що задається координатами верхнього лівого кута (x, y) та розміром (ширина, висота).

2.3. Зміщення значень кольорів по тьром каналам RGB (r, g, b) від -255 до 255 фрагменту зображення, що задається координатами верхнього лівого кута (x, y) та розміром (ширина, висота).

2.4. Відображення на екрані значення кольору пікселя у форматі RGB(r, g, b) та XYZ(x, y, z) при наведенні курсора на піксель.

2.5. Збереження (завантаження) отриманого зображення у форматі png, jpg, jpeg або jiff.

2.6. Скидування раніше внесених у зображення змін.

3. Афінні перетворення

3.1. Побудова на графіку трапеції, заданої через координати (x, y) чотирьох вершин.

3.2. Виконання повороту трапеції відносно заданої вершини проти та за годинниковою стрілкою на N градусів та одночасне пропорційне масштабування в M разів.

3.3. Збереження (завантаження) зображення отриманого графіка у форматі png.

4. Сторінка із навчальними матеріалами.

4.1. Сторінка із посиланнями на навчальні відеоматеріали для самостійного опрацювання та закріплення раніше вивченого матеріалу.

4.2. Сторінки із тестами для перевірки знань: сторінка із тестом на тему “Фрактали”, сторінка із тестом на тему “Кольорні моделі” та сторінка із тестом на тему “Афінні перетворення”.

Текстова специфікація навчальних компонентів програми

Відвідавши сторінку із посиланнями на навчальні відеоматеріали, користувач має змогу ознайомитись із теоретичними відомостями на такі теми: “Фрактали”, “Кольорові моделі” та “Афінні перетворення”. Для закріплення засвоєної інформації користувач може пройти тести для самоперевірки на відповідні теми. В кінці кожного тесту на екран виводиться результат проходження тесту. Користувач має змогу перепроходити тести безліч разів з метою покращення результату.

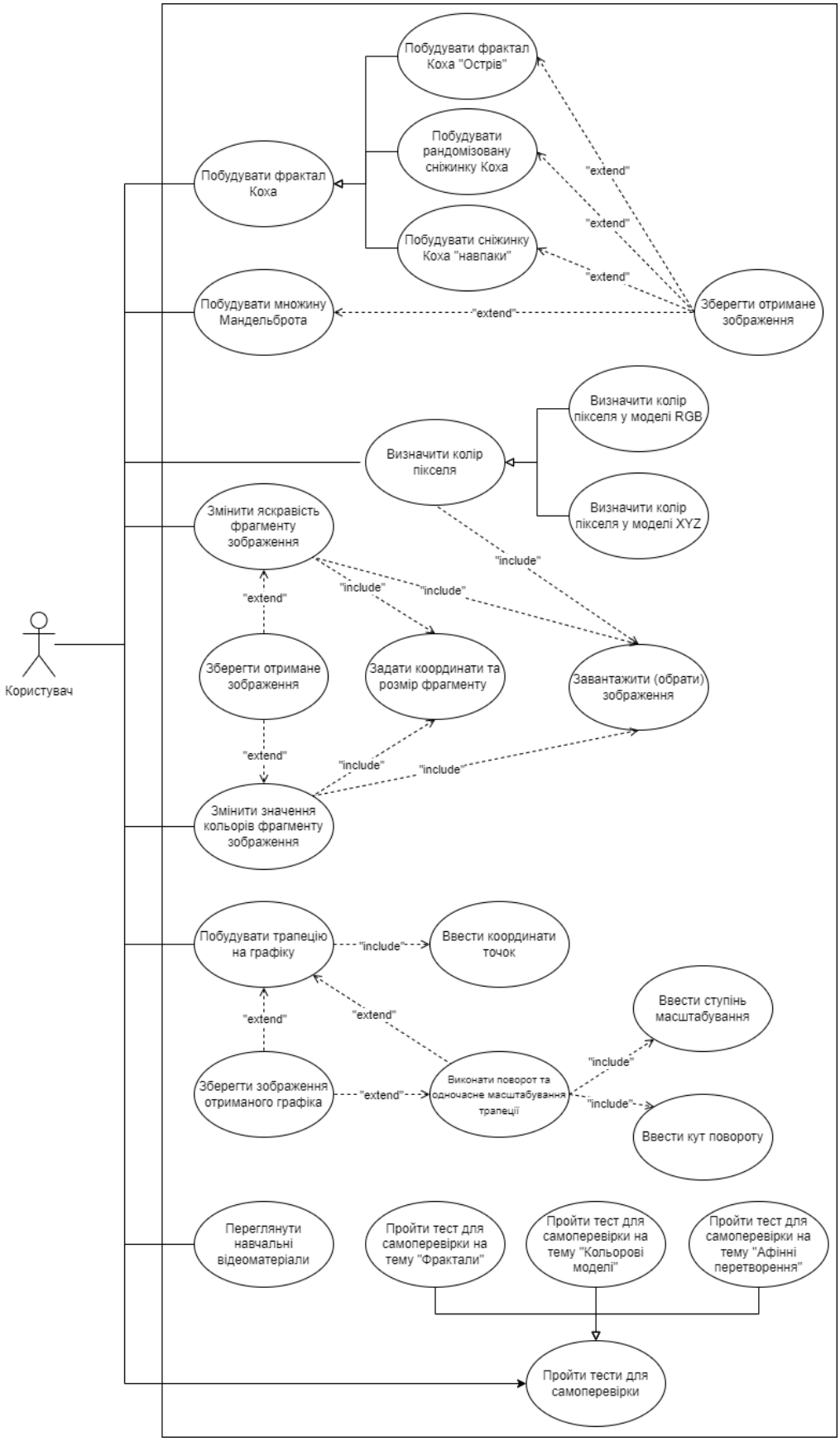

Рис. 1. Діаграма прецедентів

2. Алгоритми фракталів.

2.1. Алгоритм мовою JS для рекурсивної побудови фракталу Коха (Острів, Рандомізована сніжинка та сніжинка навпаки) заданої ітерації.

```
drawKochFractal(ctx, p1, p2, iterations, size, color, selectedKochFractal) {
    if (iterations === 0 || iterations === "0") { // Якщо кількість ітерацій != 0
      // Малювання сегменту (лінії)
      ctx.beginPath();
      ctx.moveTo(p1.x, p1.y);
      ctx.lineTo(p2.x, p2.y);
      ctx.strokeStyle = color;
      ctx.stroke();
      ctx.fillStyle = "rgb(0,0,255)"
    } else {

      let rand1 = 1; // змінна для рандомізації координати x
      let rand2 = 1; // змінна для рандомізації координати y

      let inverted = 1;

      if (selectedKochFractal === "randomized") { // рандомізована сніжинка
        rand1 = Math.random() * (1 - 0.1) + 0.1;
        rand2 = Math.random() * (1 - 0.1) + 0.1;
      } else if (selectedKochFractal === "inverted") { // сніжинка навпаки
        inverted = -1; // для інвертування координат
      }

      // Обчислення координат точки
      const p3 = {
        x: p1.x + rand1 * (p2.x - p1.x) / 3,
        y: p1.y + rand2 * (p2.y - p1.y) / 3,
      };
      const p4 = {
        x: p1.x + rand1 * (p2.x - p1.x) / 2 + inverted * (p2.y - p1.y) * (Math.sqrt(3) /
6),
        y: p1.y + rand2 * (p2.y - p1.y) / 2 - inverted * (p2.x - p1.x) * (Math.sqrt(3) /
6),
      };
```

```
      const p5 = {
        x: p1.x + rand1 * (p2.x - p1.x) * 2 / 3,
        y: p1.y + rand2 * (p2.y - p1.y) * 2 / 3,
      };

      // Рекурсивне малювання інших менших сегментів
      this.drawKochFractal(ctx, p1, p3, iterations - 1, size, color, selectedKochFractal);
      this.drawKochFractal(ctx, p3, p4, iterations - 1, size, color, selectedKochFractal);
      this.drawKochFractal(ctx, p4, p5, iterations - 1, size, color, selectedKochFractal);
      this.drawKochFractal(ctx, p5, p2, iterations - 1, size, color, selectedKochFractal);
    }
  }
```

2.2. Алгоритм мовою JS для ітеративної побудови множини Мандельброта заданого степеня для заданої ітерації.

```
drawMandelbrot() {
    const width = this.state.width; // Ширина зображення
    const height = this.state.height; // Висота зображення
    const exponent = this.props.exponent; // Степінь множини
    const maxIterations = this.props.iterations; // Кількість ітерацій
    const mainColor = this.props.color; // Колір точки

    if(maxIterations > 100 | maxIterations < 0) return;
    if(exponent > 15 || exponent < 2) return;

    const canvas = this.refs.canvas;
    const ctx = canvas.getContext('2d');

    for (let x = 0; x < width; x++) {
      for (let y = 0; y < height; y++) {
        const zx = (x - width / 2) * 4 / width;
        const zy = (y - height / 2) * 4 / height;

        let cRe = zx; // Дійсна частина комплексного числа
        let cIm = zy; // Уявна частина комплексного числа

        let i = 0;
```

```
        for (; i < maxIterations; i++) {
          const re2 = cRe * cRe; // Дійсна частина комплексного числа
          const im2 = cIm * cIm; // Уявна частина комплексного числа
          if (re2 + im2 > 4) break; // Умова виходу

          const newRe = Math.pow(re2 + im2, exponent / 2) * Math.cos(exponent *
Math.atan2(cIm, cRe)) + zx;
          const newIm = Math.pow(re2 + im2, exponent / 2) * Math.sin(exponent *
Math.atan2(cIm, cRe)) + zy;
          cRe = newRe;
          cIm = newIm;
        }

        // Отримання кольору пікселя залежно від числа ітерацій
        const color = this.getGradientColor(i, maxIterations, mainColor);
        ctx.fillStyle = color;
        ctx.fillRect(x, y, 1, 1);
      }
    }
  }
```

3. Анотація кольорових моделей

4.1. Модель sRGB(Standard RGB(Red, Green, Blue))

RGB - це адитивна кольорова модель, що використовує комбінацію червоного (Red), зеленого (Green) та синього (Blue) для утворення кольорів. Кожен колір представлений числовим значенням від 0 до 255 (в бітових системах) або від 0 до 1 (у вигляді дробу).

Використання: Використовується у комп'ютерних моніторах, телевізорах, фотокамерах та програмах для обробки зображень.

Переваги:

- Простота використання: Ця модель дуже популярна для відображення кольорів на екранах пристроїв через свою простоту в роботі з пікселями.

- Придатність для світлодіодних пристроїв: RGB відмінно підходить для роботи з сучасними світлодіодними технологіями.
- Добре відображає кольори для сенсорних пристроїв: RGB часто використовується для побудови кольорових моделей у сенсорних пристроях, таких як камери та дисплеї.

Недоліки:

- Обмежена гама кольорів: RGB не може точно відобразити всі кольори спектра, а саме насичені зеленосині, оскільки гама кольорів обмежена трьома основними кольорами.
- Залежність від пристрою: Кольори можуть відрізнятися на різних пристроях через різні характеристики дисплеїв та інших факторів.
- Обмеженість у застосуванні, лише на пристроях, які працюють за принципом випромінювання.

4.2. XYZ D65/2° (CIE 1931 Color Space)

XYZ – це еталонна колірна модель, колірний простір, розроблений Міжнародною комісією з освітлення (CIE), що базується на сприйнятті кольорів людиною. Цей колірний простір надає абстрактну трикомпонентну модель, яка описує колір як суму трьох значень X, Y та Z. Зазвичай, він використовується як початковий для перетворення між різними колірними просторами. Відзначимо, що визначення колірних просторів XYZ включає у себе параметри освітлення та сприйняття кольору людиною.

Щодо позначення "XYZ D65/2°", це вказує на параметри стандарту. D65 вказує на стандартне джерело світла (приблизно відповідає денному світлу). Число "2°" стосується кутового розділу кольорового простору, оскільки в спектральних дослідженнях використовується стандартний кут огляду 2 градуси.

Граничні значення для XYZ D65/2°:

X = [0, 94.811], Y = [0, 100], Z = [0, 107.304].

Переваги:

- Стандартизація: XYZ розроблена Міжнародною комісією з освітлення (CIE) і є міжнародним стандартом для опису кольорів. Це дозволяє використовувати модель у наукових та технічних дослідженнях, що стосуються кольорів.
- Незалежність від пристроїв: XYZ не залежить від конкретних характеристик пристроїв (наприклад, дисплеїв), тому вона використовується для аналізу кольорів незалежно від способу їх відображення.
- Точність та об'єктивність: Модель XYZ базується на наукових дослідженнях і математичних принципах, тому вона забезпечує більш об'єктивний опис кольорів з точки зору фізичних вимірювань.

Недоліки:

- Основним недоліком цієї системи є те, що використовуючи її, ми можемо констатувати тільки збіг чи розходження двох кольорів, але відстань між двома точками цього колірного простору не відповідає зоровому сприйняттю відмінності кольорів.
- Не використовується безпосередньо для відображення кольорів: XYZ використовується для аналізу кольорів та перетворень між різними колірними просторами, але не використовується безпосередньо для відображення кольорів на екранах чи в інших пристроях.
- Складність інтерпретації: Значення XYZ не завжди інтуїтивно зрозумілі для звичайного користувача через їх абстрактну природу і вимагають конвертації до інших колірних просторів для практичного використання.

4.3. Псевдокод для перетворення sRGB у XYZ

```
var_R = ( sR / 255 )
var_G = ( sG / 255 )
var_B = ( sB / 255 )

if ( var_R > 0.04045 ) var_R = ( ( var_R + 0.055 ) / 1.055 ) ^ 2.4
else                   var_R = var_R / 12.92
if ( var_G > 0.04045 ) var_G = ( ( var_G + 0.055 ) / 1.055 ) ^ 2.4
else                   var_G = var_G / 12.92
if ( var_B > 0.04045 ) var_B = ( ( var_B + 0.055 ) / 1.055 ) ^ 2.4
else                   var_B = var_B / 12.92

var_R = var_R * 100
var_G = var_G * 100
var_B = var_B * 100

X = var_R * 0.4124 + var_G * 0.3576 + var_B * 0.1805
Y = var_R * 0.2126 + var_G * 0.7152 + var_B * 0.0722
Z = var_R * 0.0193 + var_G * 0.1192 + var_B * 0.9505
```

4.4. Псевдокод для перетворення XYZ у sRGB

```
var_X = X / 100
var_Y = Y / 100
var_Z = Z / 100

var_R = var_X *  3.2406 + var_Y * -1.5372 + var_Z * -0.4986
var_G = var_X * -0.9689 + var_Y *  1.8758 + var_Z *  0.0415
var_B = var_X *  0.0557 + var_Y * -0.2040 + var_Z *  1.0570

if ( var_R > 0.0031308 ) var_R = 1.055 * ( var_R ^ ( 1 / 2.4 ) ) - 0.055
else                     var_R = 12.92 * var_R
if ( var_G > 0.0031308 ) var_G = 1.055 * ( var_G ^ ( 1 / 2.4 ) ) - 0.055
else                     var_G = 12.92 * var_G
if ( var_B > 0.0031308 ) var_B = 1.055 * ( var_B ^ ( 1 / 2.4 ) ) - 0.055
else                     var_B = 12.92 * var_B

sR = var_R * 255
sG = var_G * 255
sB = var_B * 255
```

4. Оптимальний матричний вираз афінних перетворень

На вході маємо матрицю однорідних координат точок трапеції, A(x, y) – вершина трапеції, N – рівень масштабування та ϕ – кут повороту. План дій для афінного перетворення:

1) Переміщення початку системи координат у A(x, y) на вектор (x, y) так, щоб точка повороту стала початком координат;

2) Поворот точки на кут ϕ у додатному напрямку відносно початку координат;

3) Пропорційне масштабування точки в N разів;

4) Переміщення одержаного результату назад так (повернення системи координат), щоб точка повороту співпала з точкою A.

$$X = \begin{matrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{matrix}$$ – початкова матриця однорідних координат точок трапеції

$$O = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{matrix}$$ – матриця переміщення початку системи координат на (x, y)

$$R = \begin{matrix} cos\phi & sin\phi & 0 \\ -\sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{matrix}$$ – матриця повороту на кут ϕ у додатному напрямку

$$S = \begin{matrix} N & 0 & 0 \\ 0 & N & 0 \\ 0 & 0 & 1 \end{matrix}$$ – матриця пропорційного масштабування в N разів

$$\acute{O} = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{matrix}$$ – матриця переміщення початку системи координат назад

$T = O * R * S * O$ – матриця перетворень

$X' = X * T$ – матриця перетворених координат.

5. Реалізація графічного режиму

Для розробки нашої програми ми обрали бібліотеку React, яка надає можливість створювати веб-інтерфейси за допомогою компонентів. Компоненти - це будівельні блоки інтерфейсу користувача. Вони дозволяють інкапсулювати частини інтерфейсу, які можна використовувати повторно, а також керувати логікою та рендерингом. Компоненти у React можуть реагувати на події або зміни даних, і автоматично оновлювати своє відображення. Такий підхід був дуже корисним для нашого проекту, оскільки ми потребували постійного оновлення відображення в залежності від змін в даних чи користувацьких взаємодій.

Крім того, ми використали біблотеку MUI (Material UI) для налаштування графіки. А саме нам пригодились такі компоненти як TextField, RadioButtons, MuiColorInput, Select та Button. TextField був використаний для введення текстової інформації, RadioButtons дозволили вибирати один із декількох варіантів, MuiColorInput – для вибору кольору, Select - для вибору зі списку параметрів, а Button використовувався для створення кнопок, що викликали певні дії у програмі. За допомогою цих компонент бібліотеки MUI нам вдалось швидко та легко створити необхідні інструменти користувача для введення інформації.

Також для виведення графічної інформації у розділі програми з фракталами та колірними моделями ми застосували такий елемент як <canvas>. <canvas> - це

елемент HTML5, який надає простий, але потужний інтерфейс для малювання графіки з використанням скриптів JavaScript. Він дозволяє створювати різноманітні малюнки, діаграми, графіки, анімації та інші візуальні ефекти на веб-сторінці. Контекст у використанні канвасу (canvas context) - це об'єкт, який надає доступ до методів та властивостей, що дозволяють малювати на полотні <canvas>.

Використані методи та атрибути компоненти <canvas> під час виконання лабораторних робіт 1-4:

- Отримання контексту малювання: <canvas> має контекст малювання, який можна отримати за допомогою методу getContext()
- Читання пікселів: getImageData() дозволяє отримати дані про колір пікселів у певній області на <canvas>.
- Запис пікселів: putImageData() використовується для запису змін у дані пікселів на <canvas>.
- Малювання ліній: beginPath(), moveTo(), lineTo() для створення ліній.
- Колір та стиль: fillStyle, strokeStyle, lineWidth для керування кольорами та стилями.
- Обробка подій: Додавання події onChange() та інших для створення інтерактивних елементів.

Дуже корисною бібліотекою для побудови координатної площини та відображення на ній результату повороту трапеції стала Plotly.js. Вона надає широкі можливості для побудови різноманітних візуальних елементів, таких як лінійні графіки, кругові діаграми, стовпчасті діаграми та ін. Plotly.js гарантує високий рівень інтерактивності графіків, адже ця бібліотека дозволяє користувачам змінювати масштаб графіка, збільшувати або зменшувати окремі елементи, виокремлювати окремі області графіка для подальшого дослідження, зберігати графічні елементи у різноманітних форматах файлів.

Результати

1. Wireflow

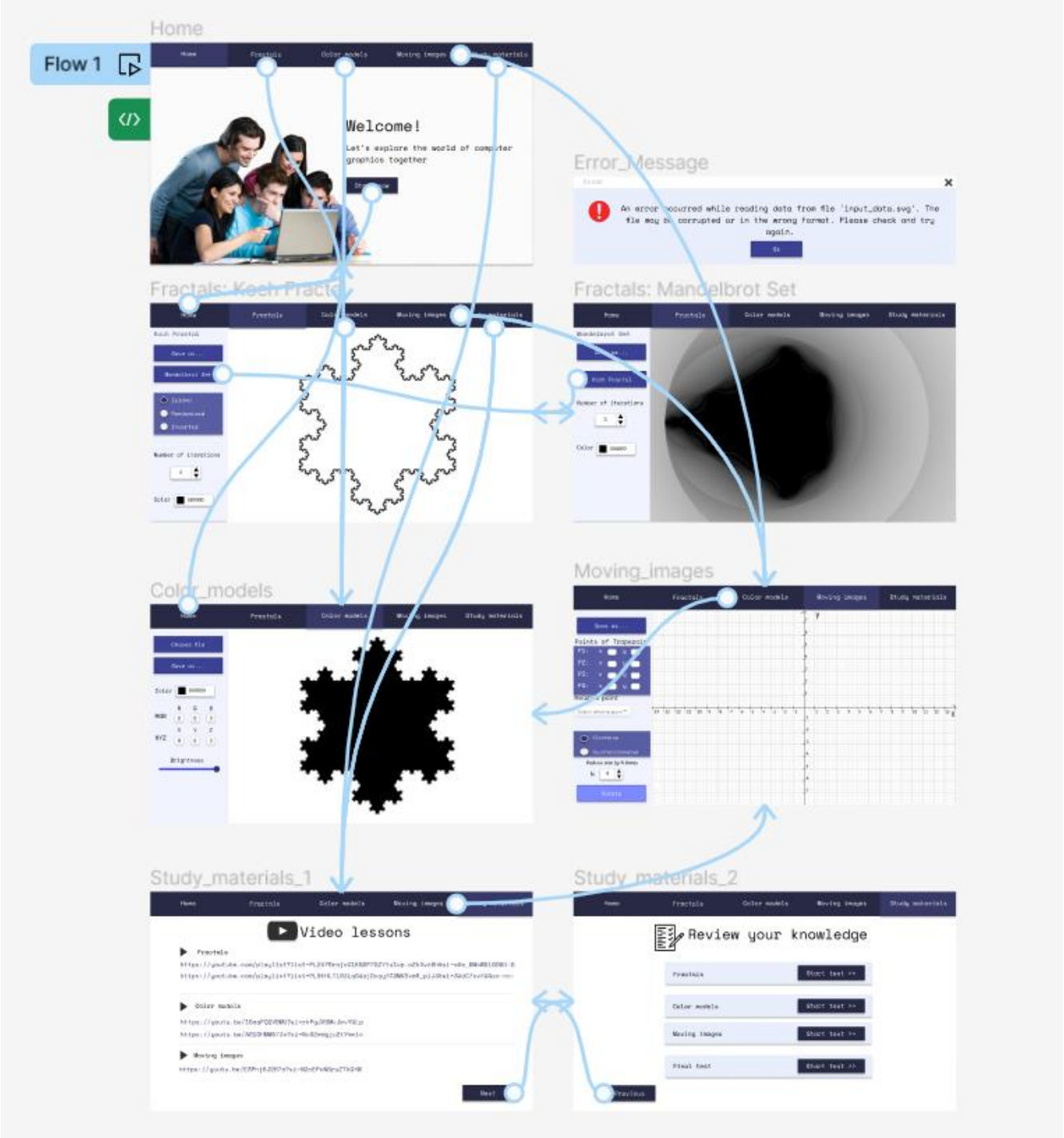

Рис. 2. Wireflow

2. UI kit

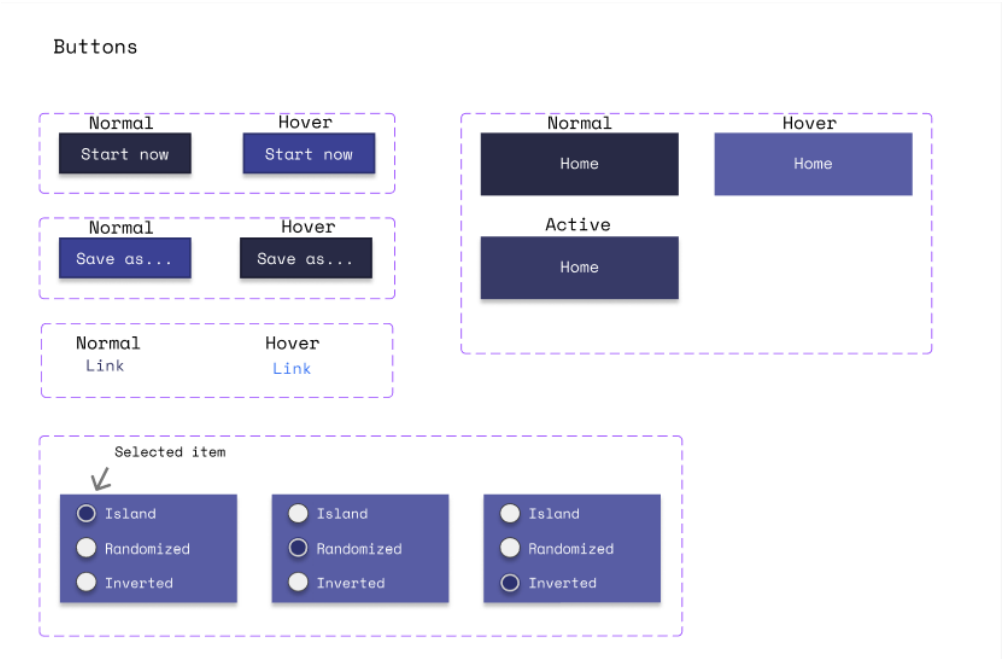

Рис. 3. UI kit для кнопок

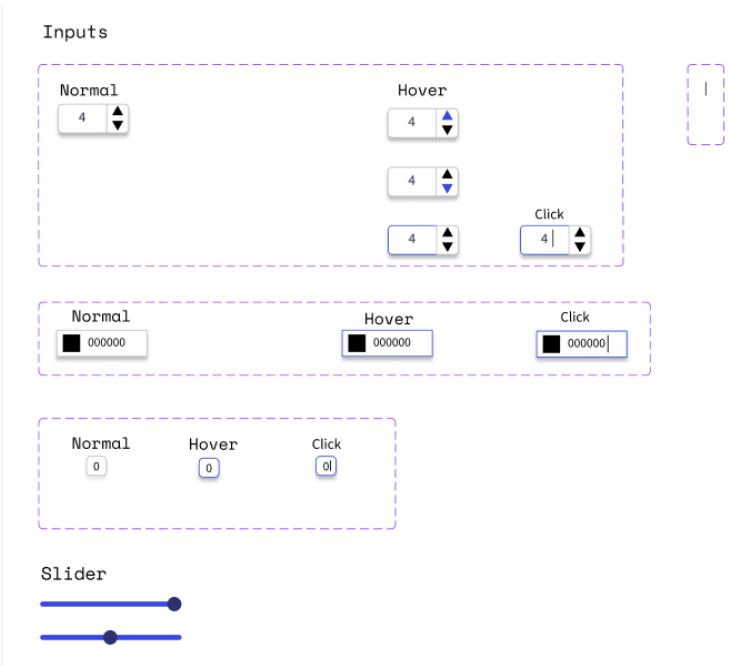

Рис. 4. UI kit для полів введення та повзунка

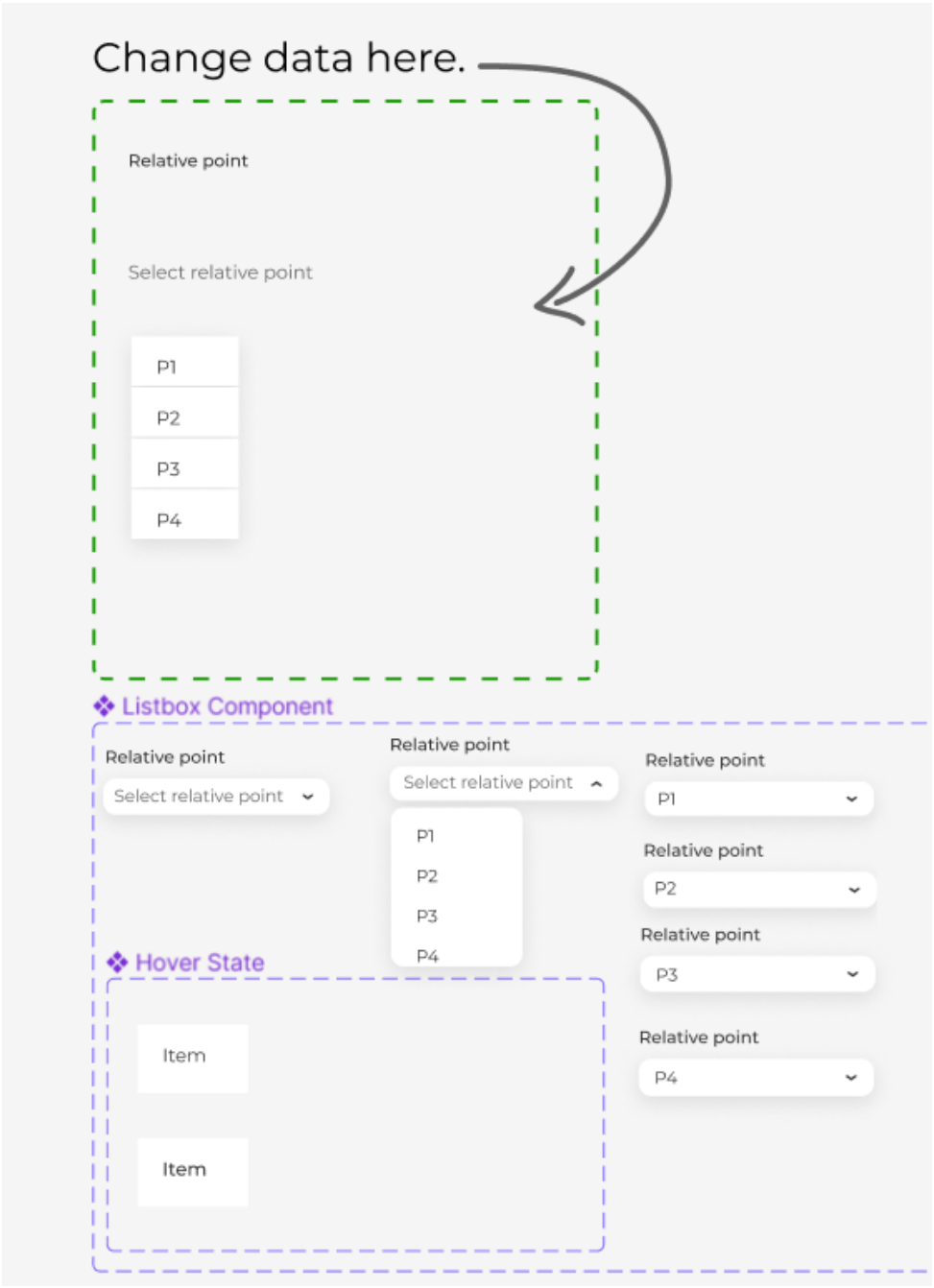

Рис. 5. UI kit для випадного списку

3. Отримані фрактали

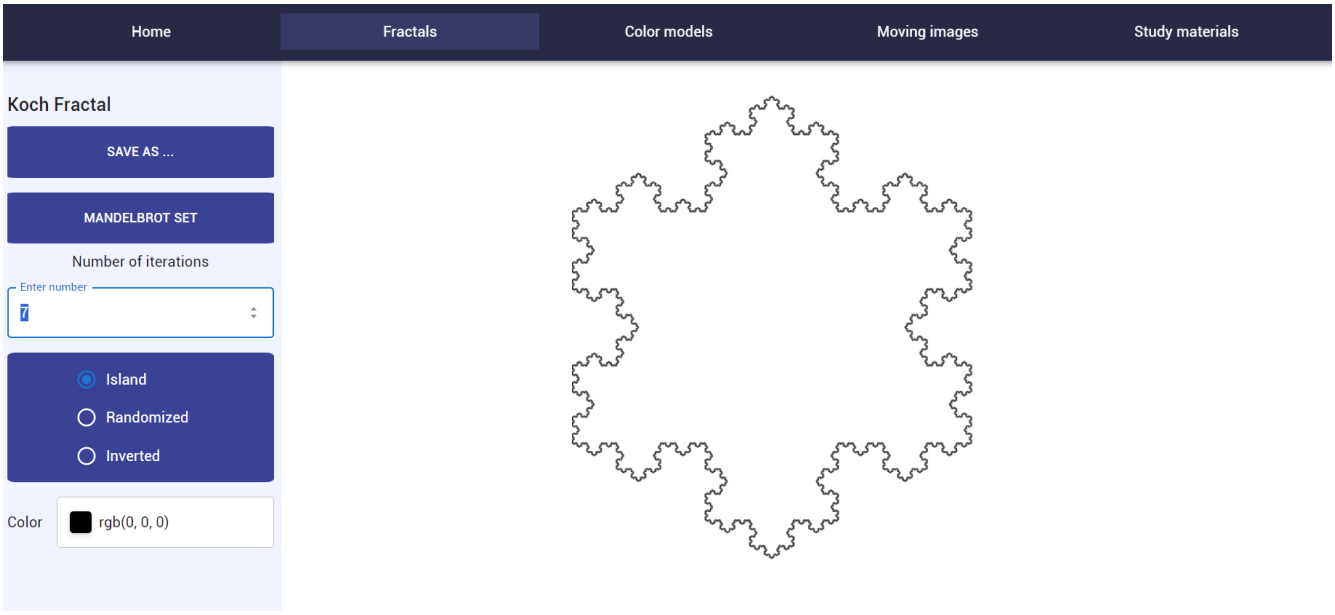

Рис. 6. Фрактал Коха (острів)

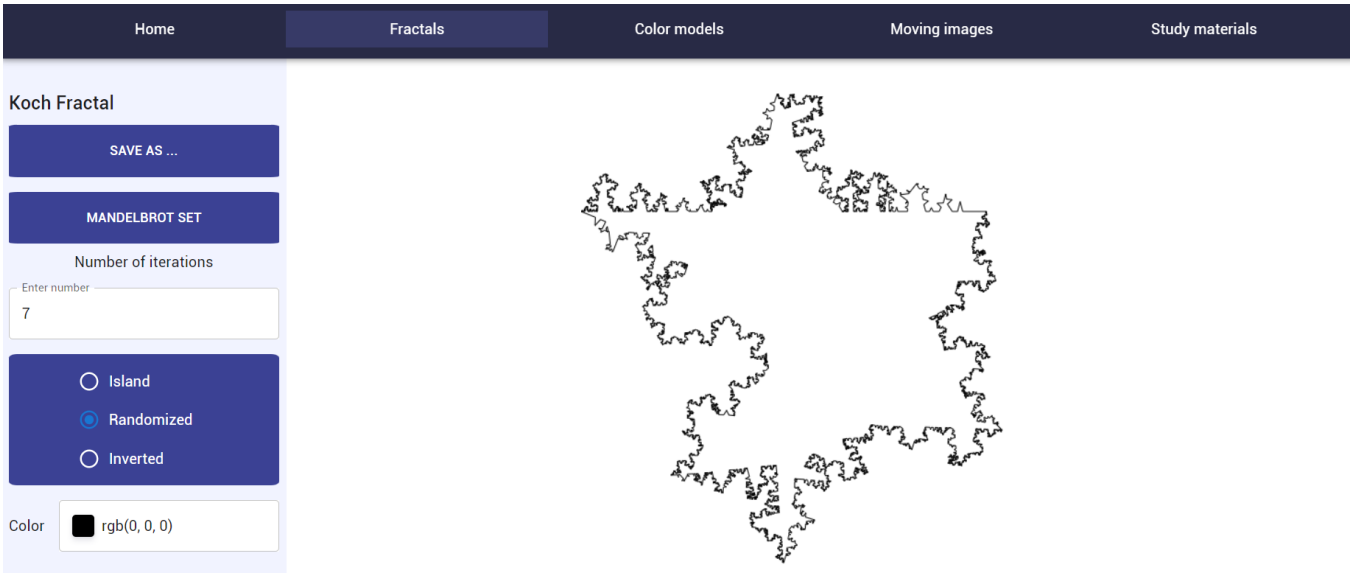

Рис. 7. Фрактал Коха («рандомізована» сніжинка)

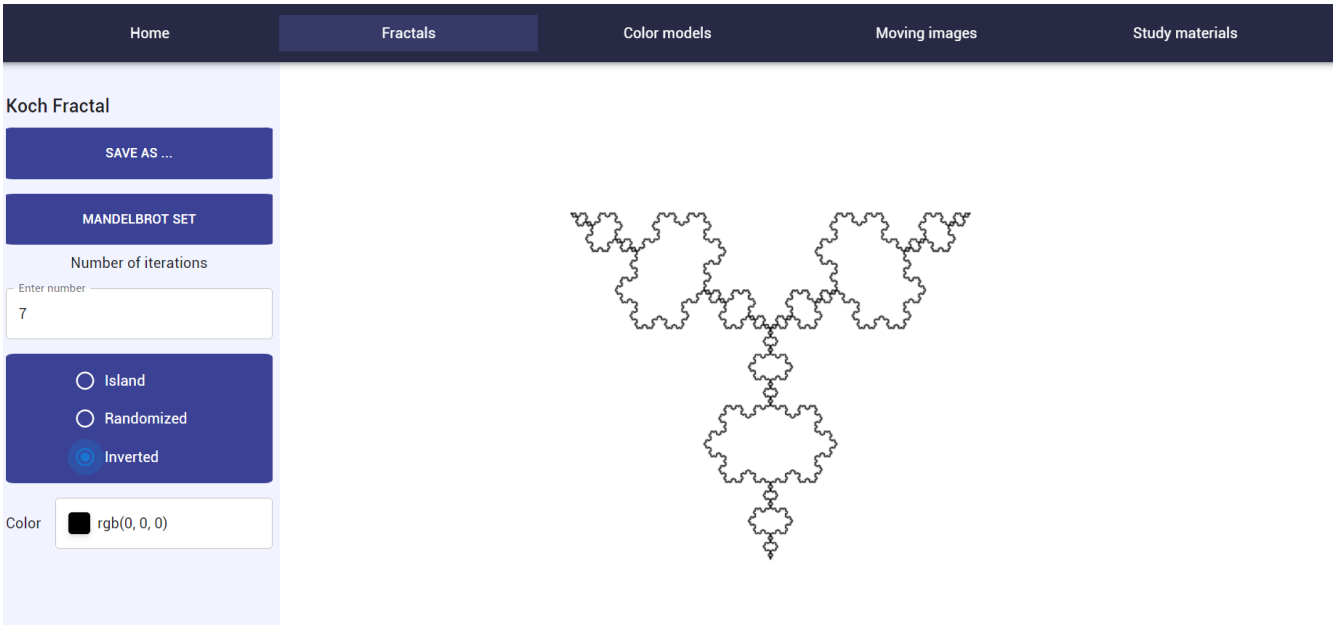

Рис. 8. Фрактал Коха (сніжинка «навпаки»)

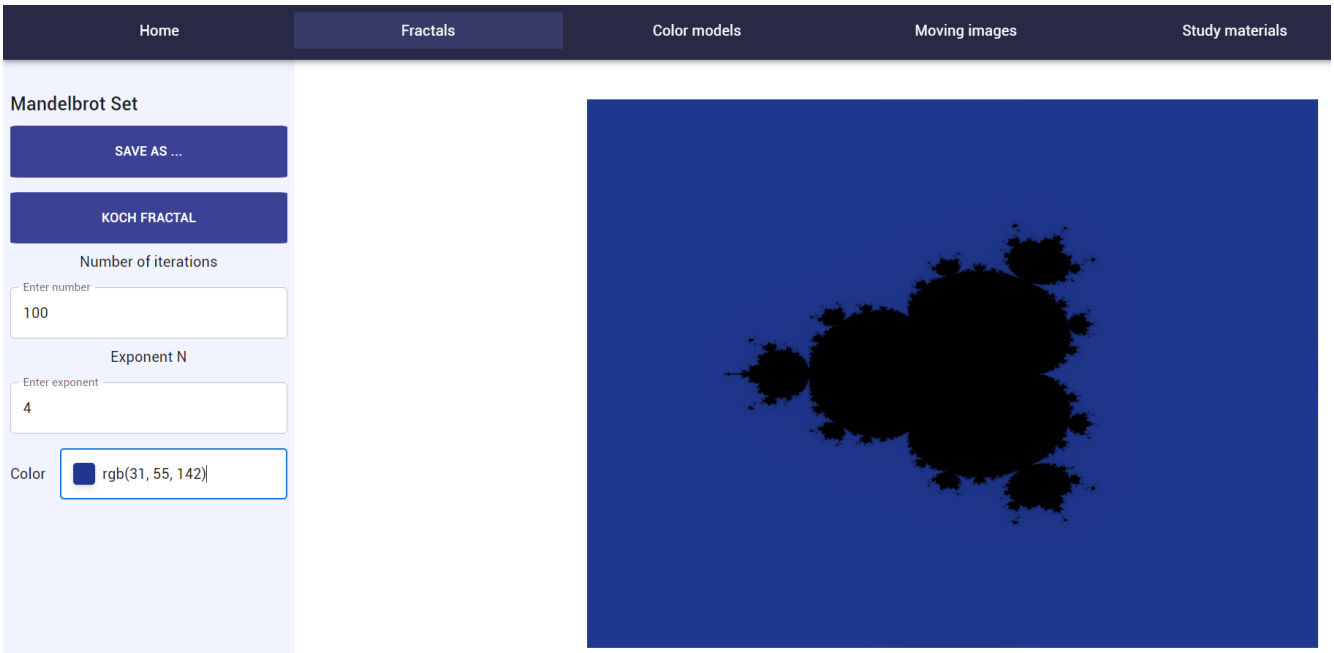

Рис. 9. Множина Мандельброта для 4 степеня z

Рис. 10. Множина Мандельброта для 5 степеня z

4. Вигляд програми для лабораторної роботи №3

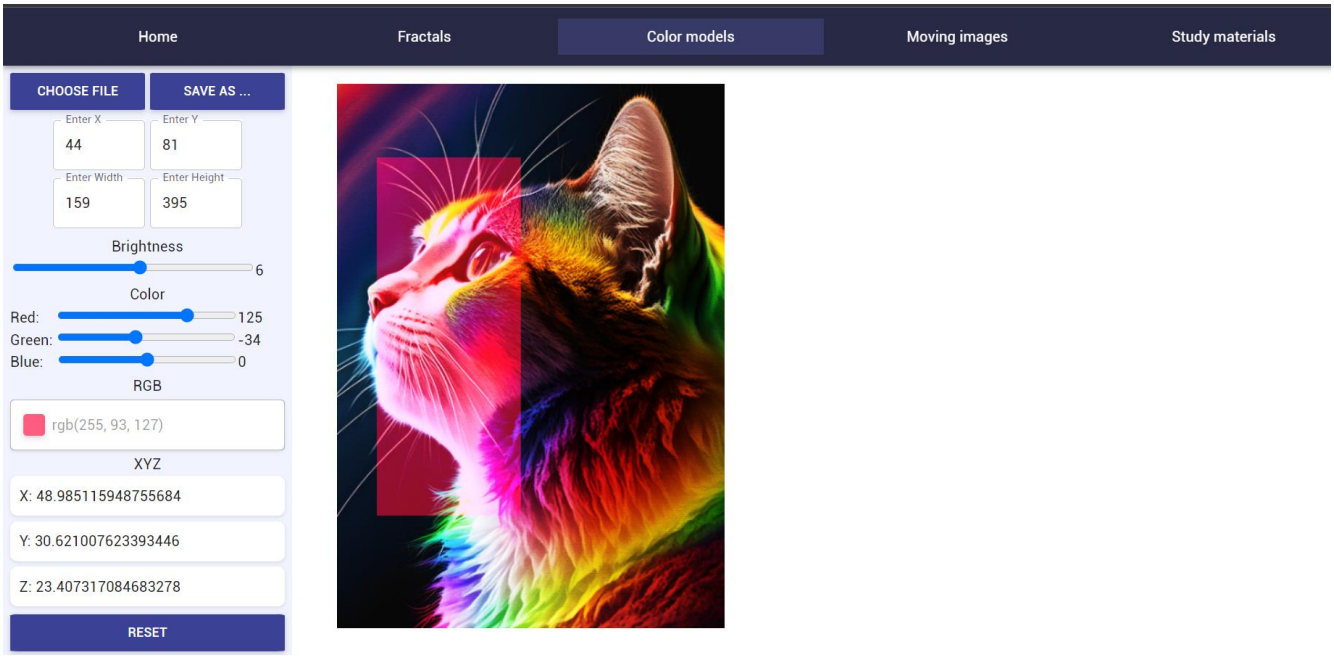

Рис. 11. Колірні моделі: RGB і XYZ

5. Вигляд програми для лабораторної роботи №4

Рис. 12. Афінні перетворення

6. Текст програми з коментарями.

Код для побудови різновидів фрактала Коха:

Назва файлу: KochSnowflake.js

```
import React, { Component } from 'react';


class KochSnowflake extends Component {
  constructor(props) {
    super(props);
    this.state = {
      MAX_NUMBER_OF_ITERATIONS: 7,// Максимальна кількість ітерацій
      iterations: props.iterations || 0, // Кількість ітерацій
      color: props.color || '#000000',// Колір ліній
      size: props.size || 1100,  // Розмір полотна
```

```
    selectedKochFractal: props.selectedKochFractal,// Вид фрактала Коха
  };
}
componentDidMount() {
   // Отримання контексту для малювання на canvas
  const canvas = this.refs.canvas;
  const ctx = canvas.getContext('2d');
  ctx.fillStyle = "rgb(0,0,255)"
  const { iterations, color, size } = this.state;
  // Визначення точок для створення початкового рівностороннього трикутника
  const p1 = { x: size * 0.3 - size * 0.3, y: 150 };
  const p2 = { x: size * 0.7 - size * 0.3, y: 150 };
  const p3 = {
    x: (p1.x + p2.x) / 2 + (p1.y - p2.y) * (Math.sqrt(3) / 2),
    y: (p1.y + p2.y) / 2 + (p2.x - p1.x) * (Math.sqrt(3) / 2),
  };
   // Фон канвасу
  ctx.globalCompositeOperation = 'destination-under'
  ctx.fillStyle = "white";
  ctx.fillRect(0, 0, canvas.width, canvas.height);
 // Виклик функції для малювання сніжинки Коха
  this.drawKochSnowflake(ctx, p1, p2, p3, iterations, size, color);
}
// Функція для малювання трикутника
drawTriangle(){
  const canvas = this.refs.canvas;
  const ctx = canvas.getContext('2d');
```

```
    ctx.fillStyle = "rgb(0,0,255)"
    const { iterations, color, size } = this.state;
     // Визначення точок для рівностороннього трикутника
    const p1 = { x: size * 0.3 - size * 0.3, y: 150 };
    const p2 = { x: size * 0.7 - size * 0.3, y: 150 };
    const p3 = {
      x: (p1.x + p2.x) / 2 + (p1.y - p2.y) * (Math.sqrt(3) / 2),
      y: (p1.y + p2.y) / 2 + (p2.x - p1.x) * (Math.sqrt(3) / 2),
    };
    // Фон канвасу
    ctx.globalCompositeOperation = 'destination-under'
    ctx.fillStyle = "white";
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    // Виклик функції для малювання сніжинки Коха
    this.drawKochSnowflake(ctx, p1, p2, p3, iterations, size, color);
  }
   // Функція для малювання сніжинки Коха
  drawKochSnowflake(ctx, p1, p2, p3, iterations, size, color, selectedKochFractal) {
    if('' === iterations) return;
    if(0 > iterations || this.state.MAX_NUMBER_OF_ITERATIONS < iterations) return;
    this.drawKochFractal(ctx, p1, p2, iterations, size, color, selectedKochFractal);
    this.drawKochFractal(ctx, p2, p3, iterations, size, color, selectedKochFractal);
    this.drawKochFractal(ctx, p3, p1, iterations, size, color, selectedKochFractal);
  }
  drawKochFractal(ctx, p1, p2, iterations, size, color, selectedKochFractal) {
    if (iterations === 0 || iterations === "0") {
       // Малювання лінії
```

```
      ctx.beginPath();// Починаємо новий шлях для малювання
      ctx.moveTo(p1.x, p1.y); // Встановлюємо початкову точку лінії
      ctx.lineTo(p2.x, p2.y);// Малюємо лінію до певної точки
      ctx.strokeStyle = color;// Встановлюємо колір лінії зі змінної color
      ctx.stroke(); // Виконуємо малювання лінії
      ctx.fillStyle = "rgb(0,0,255)"// Встановлюємо колір заливки
    } else {
      let rand1 = 1;
      let rand2 = 1;
      let inverted = 1;
      // Перевірка типу вибраного фрактала.
      if (selectedKochFractal === "randomized") {
        // Якщо вибраний тип "randomized", генеруються випадкові значення для rand1 та
rand2 в межах від 0.1 до 1.
        rand1 = Math.random() * (1 - 0.1) + 0.1;
        rand2 = Math.random() * (1 - 0.1) + 0.1;
      } else if (selectedKochFractal === "inverted") {
         // Якщо вибраний тип "inverted", змінна inverted встановлюється на -1 для зміни
орієнтації побудови точок фрактала.
        inverted = -1;
      }
       // Обчислення нових точок
      const p3 = {
        x: p1.x + rand1 * (p2.x - p1.x) / 3,
        y: p1.y + rand2 * (p2.y - p1.y) / 3,
      };
      const p4 = {
```

```
        x: p1.x + rand1 * (p2.x - p1.x) / 2 + inverted * (p2.y - p1.y) * (Math.sqrt(3) /
6),
        y: p1.y + rand2 * (p2.y - p1.y) / 2 - inverted * (p2.x - p1.x) * (Math.sqrt(3) /
6),
      };
      const p5 = {
        x: p1.x + rand1 * (p2.x - p1.x) * 2 / 3,
        y: p1.y + rand2 * (p2.y - p1.y) * 2 / 3,
      };
       // Рекурсивне малювання менших відрізків
      this.drawKochFractal(ctx, p1, p3, iterations - 1, size, color, selectedKochFractal);
      this.drawKochFractal(ctx, p3, p4, iterations - 1, size, color, selectedKochFractal);
      this.drawKochFractal(ctx, p4, p5, iterations - 1, size, color, selectedKochFractal);
      this.drawKochFractal(ctx, p5, p2, iterations - 1, size, color, selectedKochFractal);
    }
  }
  componentDidUpdate(prevProps) {
    if (
      prevProps.iterations !== this.props.iterations ||
      prevProps.color !== this.props.color ||
      prevProps.selectedKochFractal !== this.props.selectedKochFractal
      ) {
      const canvas = this.refs.canvas;
      const ctx = canvas.getContext('2d');
      ctx.clearRect(0, 0, canvas.width, canvas.height);
      // Фон канвасу
      ctx.globalCompositeOperation = 'destination-under'
```

```
      ctx.fillStyle = "white";
      ctx.fillRect(0, 0, canvas.width, canvas.height);
      const size = this.state.size;
      const iterations = this.props.iterations;
      const color = this.props.color;
      const selectedKochFractal = this.props.selectedKochFractal;
      this.setState({ iterations, color, size, selectedKochFractal });
       // Визначення точок для рівностороннього трикутника
      const p1 = { x: size * 0.3 - size * 0.3, y: 150 };
      const p2 = { x: size * 0.7 - size * 0.3, y: 150 };
      const p3 = {
        x: (p1.x + p2.x) / 2 + (p1.y - p2.y) * (Math.sqrt(3) / 2),
        y: (p1.y + p2.y) / 2 + (p2.x - p1.x) * (Math.sqrt(3) / 2),
      };
       // Виклик функції для малювання сніжинки Коха
      this.drawKochSnowflake(ctx, p1, p2, p3, iterations, size, color,
selectedKochFractal);
      this.render();
    }
  }
  render() {
    return <canvas id="canva" ref="canvas" width={700} height={700}></canvas>;
  }
}
export default KochSnowflake;
```

Код для побудови множини Мандельброта:

Назва файлу: MandelbrotSetFractal.js

```
import React, { Component } from 'react';


class MandelbrotSet extends Component {
  constructor(props) {
    super(props);
    this.state = {
      width: 800,
      height: 600,
      exponent: props.exponent || 2,
      maxIterations: props.iterations || 0,
      color: props.color || "rgb(255, 255, 255)",
    };
  }
    componentDidMount() {
      // Малювання фрактала Мандельброта при завантаженні компоненти
    this.drawMandelbrot();
  }
  componentDidUpdate(prevProps) {
     // Оновлення фрактала при зміні параметрів
    if (
        prevProps.iterations !== this.props.iterations ||
        prevProps.color !== this.props.color ||
        prevProps.exponent !== this.props.exponent
    ) {
       // Очистка канвасу
```

```
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext('2d');
        ctx.clearRect(0, 0, canvas.width, canvas.height);
         // Отримання нових значень параметрів
        const width = this.state.width;
        const height = this.state.height;
        const iterations = this.props.iterations;
        const color = this.props.color;
        const exponent = this.props.exponent;
        this.setState({width, height, exponent, iterations, color});
        // Перемалювання фрактала Мандельброта
        this.drawMandelbrot();
        this.render();
    }
}
  drawMandelbrot() {
    const width = this.state.width;// Ширина canvas
    const height = this.state.height;// Висота canvas
    const exponent = this.props.exponent;// Степінь z фракталу Мандельброта
    const maxIterations = this.props.iterations;// Максимальна кількість ітерацій
    const mainColor = this.props.color;// Основний колір фракталу
     // Перевірка обмежень для кількості ітерацій та показника
    if(maxIterations > 100 | maxIterations < 0) return;
    if(exponent > 15 || exponent < 2) return;
    const canvas = this.refs.canvas;// Отримання доступу до canvas
    const ctx = canvas.getContext('2d');// Отримання контексту малювання 2D
    for (let x = 0; x < width; x++) {
```

```
      for (let y = 0; y < height; y++) {
        const zx = (x - width / 2) * 4 / width;// Реальна частина комплексного числа
        const zy = (y - height / 2) * 4 / height;// Уявна частина комплексного числа
        let cRe = zx;
        let cIm = zy;
        let i = 0;
        for (; i < maxIterations; i++) {
          const re2 = cRe * cRe;
          const im2 = cIm * cIm;
          if (re2 + im2 > 4) break;
          const newRe = Math.pow(re2 + im2, exponent / 2) * Math.cos(exponent *
Math.atan2(cIm, cRe)) + zx;
          const newIm = Math.pow(re2 + im2, exponent / 2) * Math.sin(exponent *
Math.atan2(cIm, cRe)) + zy;
          cRe = newRe;
          cIm = newIm;
        }
        // Колірування пікселів на основі кількості ітерацій
        const color = this.getGradientColor(i, maxIterations, mainColor);
        ctx.fillStyle = color;
        ctx.fillRect(x, y, 1, 1);// Малювання пікселя
      }
    }
  }
getGradientColor(iterations, maxIterations, baseColor) {
    let hue = (iterations / maxIterations) * 360;
    if(0 === maxIterations) {
```

```
      return `rgb(${0},${0},${0})`

    }

    let match = baseColor.match(/rgb\((\d+), (\d+), (\d+)\)/);

    let r, g, b, a = 1;

    if(match === null)

    {

      match = baseColor.match(/rgba\((\d+), (\d+), (\d+), ([0-9]*[.,]?[0-9]+)\)/);

      a = parseFloat(match[4]);

    }

      r = parseInt(match[1], 10);

      g = parseInt(match[2], 10);

      b = parseInt(match[3], 10);

    const rgbColor = this.hslToRgb(hue / 360, 1, 0.5);

    const red = Math.floor(r * (1 - iterations / maxIterations) + rgbColor.r * iterations
/ maxIterations);

    const green = Math.floor(g * (1 - iterations / maxIterations) + rgbColor.g *
iterations / maxIterations);

    const blue = Math.floor(b * (1 - iterations / maxIterations) + rgbColor.b * iterations
/ maxIterations);

    return `rgba(${red},${green},${blue}, ${a})`;

  }

  hexToRgb(hex) {

    hex = hex.replace(/^#/, '');

    const bigint = parseInt(hex, 16);

    const r = (bigint >> 16) & 255;

    const g = (bigint >> 8) & 255;

    const b = bigint & 255;

    return { r, g, b };
```

```
  }
  hslToRgb(h, s, l) {
    let r, g, b;
    if (s === 0) {
      r = g = b = l;
    } else {
      const hue2rgb = (p, q, t) => {
        if (t < 0) t += 1;
        if (t > 1) t -= 1;
        if (t < 1 / 6) return p + (q - p) * 6 * t;
        if (t < 1 / 2) return q;
        if (t < 2 / 3) return p + (q - p) * (2 / 3 - t) * 6;
        return p;
      };
      const q = l < 0.5 ? l * (1 + s) : l + s - l * s;
      const p = 2 * l - q;
      r = hue2rgb(p, q, h + 1 / 3);
      g = hue2rgb(p, q, h);
      b = hue2rgb(p, q, h - 1 / 3);
    }
    return { r, g, b };
  }
  render() {
    return <canvas id="canva" ref="canvas" width={this.state.width}
height={this.state.height}></canvas>;
  }
}
```

```
export default MandelbrotSet;
```

Код для роботи з колірними моделями:

Назва файлу: ImageCanvas.jsx

```
import React, { useEffect } from "react";
import { canvasStyles } from "./ImageCanvas.styles";
import { Box } from "@mui/material";


const ImageCanvas = ({
  img,
  posX,
  posY,
  sizeWidth,
  sizeHeight,
  brightness,
  red,
  green,
  blue,
  onColorChange,
  canvasRef,
}) => {
  useEffect(() => {
    // Завантаження зображення під час монтування компоненти та налаштування контексту
малювання
    const image = new Image();
```

```
image.src = img;

const MAX_IMAGE_HEIGHT = 600;

const canvas = canvasRef.current;

const ctx = canvas.getContext("2d");

// Отримання даних про кольори при наведенні миші

const handleMouseMove = (e) => {

  const rect = canvas.getBoundingClientRect();

  const x = e.clientX - rect.left;

  const y = e.clientY - rect.top;

  const pixel = ctx.getImageData(x, y, 1, 1).data;

  onColorChange(pixel[0], pixel[1], pixel[2]);

};

// Застосування яскравості до зображення

const applyBrightness = (data, brightness) => {

  for (var i = 0; i < data.length; i+= 4) {

    data[i] += 255 * (brightness / 100);

    data[i+1] += 255 * (brightness / 100);

    data[i+2] += 255 * (brightness / 100);

  }

};


image.onload = () => {

  let k = (image.height / MAX_IMAGE_HEIGHT);

  // Встановлення розмірів канвасу та відображення зображення

  canvas.height = image.height / k;

  canvas.width = image.width / k;

  ctx.drawImage(image, 0, 0, canvas.width, canvas.height);
```

```
      canvas.addEventListener("mousemove", handleMouseMove);
      updateImageColor();
      let imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);


      ctx.putImageData(imageData, 0, 0);
    };
    const updateImageColor = () => {
      // Оновлення кольорів зображення на канвасі
      if(parseInt(sizeHeight) === 0 || sizeHeight === "0" || sizeHeight === "00") return;
      if(parseInt(sizeWidth) === 0 || sizeWidth === "0" || sizeWidth === "00") return;


      const imageData = ctx.getImageData(posX, posY, sizeWidth, sizeHeight);
      const data = imageData.data;
      for (let i = 0; i < data.length; i += 4) {
        data[i] += red;
        data[i + 1] += green;
        data[i + 2] += blue;
      }
      applyBrightness(imageData.data, brightness);
      ctx.putImageData(imageData, posX, posY);
    };
    return () => {
      canvas.removeEventListener("mousemove", handleMouseMove);
    };
  }, [img, posX, posY, sizeHeight, sizeWidth, brightness, red, green, blue, onColorChange,
canvasRef]);
  return (
```

```
    // Повернення канвасу для відображення зображення
    <Box sx={canvasStyles.imageWrapper}>
      <canvas ref={canvasRef} style={canvasStyles.canvas}></canvas>
    </Box>
  );
};
export default ImageCanvas;
```

Назва файлу: ImageColorChanger.jsx

```
import React, { useState, useRef } from "react";
import Box from "@mui/material/Box";
import ColorMenu from "../ColorMenu/ColorMenu";
import ImageCanvas from "../ImageCanvas/ImageCanvas";
import catRainbowImg from "../cat_rainbow.jpg";
import { styles } from "./ImageColorChanger.styles";

const ImageColorChanger = () => {
  const [img, setImage] = useState(catRainbowImg);// Зображення, використовуване за
замовчуванням
  const canvasRef = useRef(null);// Референс на canvas елемент
  const [posX, setPosX] = useState(0);// Позиція X зображення
  const [posY, setPosY] = useState(0); // Позиція Y зображення
  const [sizeWidth, setSizeWidth] = useState(600);// Ширина зображення
  const [sizeHeight, setSizeHeight] = useState(600);// Висота зображення
  const [red, setRed] = useState(0);// Компонент червоного кольору
  const [green, setGreen] = useState(0);// Компонент зеленого кольору
```

```
const [blue, setBlue] = useState(0);// Компонент синього кольору
const [brightness, setBrightness] = useState(0);// Яскравість зображення
const [r, setR] = useState(0);// Компонент червоного кольору (RGB)
const [g, setG] = useState(0);// Компонент зеленого кольору (RGB)
const [b, setB] = useState(0);// Компонент синього кольору (RGB)
const [x, setX] = useState(0);// Компонент X (XYZ)
const [y, setY] = useState(0);// Компонент Y (XYZ)
const [z, setZ] = useState(0);// Компонент Z (XYZ)
// Функції-обробники для зміни розмірів зображення
const handleSizeWidthChange = (value) => {
  setSizeWidth(value);
};
const handleSizeHeightChange = (value) => {
  setSizeHeight(value);
};
// Функції-обробники для зміни кольорів та яскравості зображення
const handlePosXChange = (value) => {
  setPosX(value);
};
const handlePosYChange = (value) => {
  setPosY(value);
};
const handleBrightnessChange = (value) => {
  setBrightness(value);
};
const handleRedChange = (value) => {
  setRed(value);
```

```
};
const handleGreenChange = (value) => {
  setGreen(value);
};
const handleBlueChange = (value) => {
  setBlue(value);
};
 // Функція для визначення кольорів та значень XYZ
const handleColorChange = (r, g, b) => {
  setR(r);
  setG(g);
  setB(b);
  let xyz = rgbToXyz(r, g, b);
  setX(xyz.x);
  setY(xyz.y);
  setZ(xyz.z);
};
  // Функція для зміни вибраного зображення
const handleFileChange = (image) => {
  setImage(image);
};
// Функція для скидання всіх параметрів зображення
const handleReset = () => {
  setPosX(0);
  setPosY(0);
  setSizeWidth(0);
  setSizeHeight(0);
```

```
  setRed(0);
  setGreen(0);
  setBlue(0);
  setBrightness(0);
  setR(0);
  setB(0);
  setG(0);
  setX(0);
  setY(0);
  setZ(0);
};
const rgbToXyz = (r, g, b) => {
  // Переведення значень кольорів з діапазону 0-255 у діапазон 0-1
  let var_R = r / 255;
  let var_G = g / 255;
  let var_B = b / 255;
  // Корекція значень RGB у відповідності зі стандартом sRGB
  if (var_R > 0.04045) {
    var_R = Math.pow((var_R + 0.055) / 1.055, 2.4);
  } else {
    var_R = var_R / 12.92;
  }
  if (var_G > 0.04045) {
    var_G = Math.pow((var_G + 0.055) / 1.055, 2.4);
  } else {
    var_G = var_G / 12.92;
  }
```

```
    if (var_B > 0.04045) {
      var_B = Math.pow((var_B + 0.055) / 1.055, 2.4);
    } else {
      var_B = var_B / 12.92;
    }
    // Масштабування значень до діапазону від 0 до 100
    var_R = var_R * 100;
    var_G = var_G * 100;
    var_B = var_B * 100;
    // Обчислення компонентів X, Y та Z у моделі кольорів XYZ
    let X = var_R * 0.4124 + var_G * 0.3576 + var_B * 0.1805;
    let Y = var_R * 0.2126 + var_G * 0.7152 + var_B * 0.0722;
    let Z = var_R * 0.0193 + var_G * 0.1192 + var_B * 0.9505;
    // Повернення об'єкта з компонентами кольорів XYZ
    return { x: X, y: Y, z: Z };
  };
  return (
    <Box sx={styles.wrapper}>
      <ColorMenu
        posX={posX}
        posY={posY}
        sizeWidth={sizeWidth}
        sizeHeight={sizeHeight}
        handlePosXChange={handlePosXChange}
        handlePosYChange={handlePosYChange}
        handleSizeWidthChange={handleSizeWidthChange}
        handleSizeHeightChange={handleSizeHeightChange}
```

```
        brightness={brightness}
        red={red}
        green={green}
        blue={blue}
        onBrightnessChange={handleBrightnessChange}
        onRedChange={handleRedChange}
        onGreenChange={handleGreenChange}
        onBlueChange={handleBlueChange}
        rgbValue={`${r}, ${g}, ${b}`}
        xyzValues={`${x}, ${y}, ${z}`}
        onFileChange={handleFileChange}
        onReset={handleReset}
        img={img}
        canvasRef={canvasRef}
      />
      <ImageCanvas
        img={img}
        posX={posX}
        posY={posY}
        sizeWidth={sizeWidth}
        sizeHeight={sizeHeight}
        brightness={brightness}
        red={red}
        green={green}
        blue={blue}
        onColorChange={handleColorChange}
        canvasRef={canvasRef}
```

```
      />
    </Box>
  );
};
export default ImageColorChanger;
```

Код для роботи з афінними перетвореннями:

Назва файлу: MovingTrapezoid.jsx

```
import React, { useState } from "react";
import TrapezoidMenu from "../TrapezoidMenu/TrapezoidMenu.jsx";
import TrapezoidArea from "../TrapezoidArea/TrapezoidArea.jsx";
import { styles } from "./MovingTrapezoid.style.js";
import { Box } from "@mui/material";


const numeric = require("numeric");
const MovingTrapezoid = () => {
  // Визначення координат трапеції
  const [points, setPoints] = useState({
    x: [1, 2, 3, 6, 1],
    y: [1, 3, 3, 1, 1],
    name: ["Point 1", "Point 2", "Point 3", "Point 4", "Point 1"],
  });
  // Коефіцієнт збільшення
  const [scale, setScale] = useState(1.0);
  // Кут повороту
```

```
const [rotateDegree, setRotateDegree] = useState(0);

// Точка для повороту навколо неї

const [rotationPoint, setRotationPoint] = useState(0);

const handlePointChange = (index, axis, value) => {

  points[axis][index] = value;

  const updatedPoints = {

    x: [...points.x],

    y: [...points.y],

    name: [...points.name],

  };

  setPoints(updatedPoints);

};

// Функція для створення матриці трансформації для масштабування та повороту

function getTransformationMatrix(centerX, centerY, scale, angleInDegrees) {

  if('0' === scale || 0 === scale) {

    return [

      [1, 0, 0],

      [0, 1, 0],

      [0, 0, 1],

    ];

  }

  // Перетворення кута в радіани

  const angleInRadians = angleInDegrees * (Math.PI / 180);

  let cos = Math.round(Math.cos(angleInRadians) * 1000) / 1000;

  let sin = Math.round(Math.sin(angleInRadians) * 1000) / 1000;

  // Матриця трансформації для здійснення повороту відносно точки

  const rotationMatrix = [
```

```
      [cos, sin, 0],
      [-sin, cos, 0],
      [0, 0, 1],
    ];
    const scalingMatrix = [
      [scale, 0, 0],
      [0, scale, 0],
      [0, 0, 1],
    ];
    // Матриця трансформації для зміщення точки в центрі обертання
    const translationMatrixToCenter = [
      [1, 0, 0],
      [0, 1, 0],
      [centerX, centerY, 1],
    ];
    // Матриця трансформації для повернення точки назад на місце
    const translationMatrixFromCenter = [
      [1, 0, 0],
      [0, 1, 0],
      [-centerX, -centerY, 1],
    ];
    // Порядок застосування матриць: зміщення в центр, масштабування, поворот, зміщення
назад
    const transformationMatrix = multiplyMatrices(
      translationMatrixToCenter,
      scalingMatrix,
      rotationMatrix,
```

```
    translationMatrixFromCenter
  );
  return transformationMatrix;
}
function getTransformedPoints(matrix, points) {
  // Додайте третю координату (однорідні координати)
  const homogeneousPoints = [
    [points[0].x, points[0].y, 1],
    [points[1].x, points[1].y, 1],
    [points[2].x, points[2].y, 1],
    [points[3].x, points[3].y, 1],
    [points[4].x, points[4].y, 1],
  ];

  // Помножте вектор на обернену матрицю
  const transformedPoint = numeric.dot(homogeneousPoints, matrix);
  // Поверніть координати точки
  return [
    { x: transformedPoint[0][0], y: transformedPoint[0][1] },
    { x: transformedPoint[1][0], y: transformedPoint[1][1] },
    { x: transformedPoint[2][0], y: transformedPoint[2][1] },
    { x: transformedPoint[3][0], y: transformedPoint[3][1] },
    { x: transformedPoint[4][0], y: transformedPoint[4][1] },
];
}
// Функція для множення матриць
function multiplyMatrices(...matrices) {
```

```
  return matrices.reduce(
    (result, matrix) =>
      result.map((row, i) =>
        row.map(
          (_, j) =>
            matrix[i][0] * result[0][j] +
            matrix[i][1] * result[1][j] +
            matrix[i][2] * result[2][j]
        )
      ),
    [
      [1, 0, 0],
      [0, 1, 0],
      [0, 0, 1],
    ]
  );
}


const handleRotateDegreeChange = (value) => {
  setRotateDegree(value);
};


const handleScaleChange = (value) => {
  setScale(value);
  if (0 === value || "" === value || !value) return;
};
```

```
const handleTransformOnClick = () => {
 // Точка, відносно якої робить поворот
 const x0 = points.x[rotationPoint];
 const y0 = points.y[rotationPoint];

 let transformedPoints = getTransformedPoints(
   getTransformationMatrix(x0, y0, scale, rotateDegree),
   [
     { x: points.x[0], y: points.y[0] },
     { x: points.x[1], y: points.y[1] },
     { x: points.x[2], y: points.y[2] },
     { x: points.x[3], y: points.y[3] },
     { x: points.x[4], y: points.y[4] },
 ]
 );

 for(let i = 0; i < transformedPoints.length; ++i){
   points.x[i] = transformedPoints[i].x;
   points.y[i] = transformedPoints[i].y;
 }
 const updatedPoints = {
   x: [...points.x],
   y: [...points.y],
   name: [...points.name],
 };

 setPoints(updatedPoints);
```

```
  };

  const handleRotatePointChange = (val) => {
    setRotationPoint(val);
  };

  return (
    <Box sx={styles.wrapper}>
      <TrapezoidMenu
        points={points}
        handlePointChange={handlePointChange}
        scale={scale}
        handleScaleChange={handleScaleChange}
        rotateDegree={rotateDegree}
        handleRotateDegreeChange={handleRotateDegreeChange}
        handleTransformOnClick={handleTransformOnClick}
        rotationPoint={rotationPoint}
        handleRotatePointChange={handleRotatePointChange}
      />
      <TrapezoidArea points={points} />
    </Box>
  );
};

export default MovingTrapezoid;
```

Висновки

Отже, під час виконання даних лабораторних робіт **ми навчились**:

- Програмувати алгоритми для побудови фракталів, працювати з колірними моделями та афінними перетвореннями за допомогою таких бібліотек як React, MUI, Plotly.js. Цей досвід буде корисним у подальшій роботі та відкриє нові можливості для створення цікавих та складних графічних застосунків.
- Злагоджено працювати в команді, краще розумітись в Git, покладатись один на одного. Це є одним з найважливіших результатів, адже сучасному програмісту не обійтись без навичок командної роботи.

Та все ж у нас були складнощі з:

- Освоєнням нових технологій. Завжди є складно з нуля навчатись працювати з новими для себе бібліотеками з нуля. Потрібно було часто читати документацію по React та вишукувати помилки в коді. Але ці знання стануть нам у пригоді під час програмування веб-застосунків у майбутньому.
- Розподілом часу на завдання. Особливо на початку ми відкладали весь час виконання завдання на потім замість того, щоб зробити його за кілька днів наперед. Але згодом ви покращили свій тайм-менеджмент.

Відсотки участі кожного студента у виконанні кожного завдання:

Лаб 1-4:

Єгор – 60%

Марічка – 40%

Звіт:

Єгор – 50%

Марічка – 50%

Список використаних інформаційних джерел

1. https://en.wikipedia.org/wiki/Koch_snowflake - Koch snowflake: [Електронний ресурс] // Вікіпедія – вільна енциклопедія.
2. https://plus.maths.org/content/what-mandelbrot-set - Robert Devaney - “What is the Mandelbrot set?”: [Електронний ресурс] // Plus Magazine
3. https://www.easyrgb.com/en/math.php - Color math and programming code examples: [Електронний ресурс] // EasyRGB
4. https://www.mathworks.com/discovery/affine-transformation.html#:~:text=Affine%20transformation%20is%20a%20linear,with%20non%2Dideal%20camera%20angles – “What Is an Affine Transformation?”: [Електронний ресурс] // MathWorks
5. https://plotly.com/javascript/react/ - React Plotly.js in JavaScript [Електронний ресурс] // Plotly
6. https://surveyjs.io/form-library/documentation/get-started-react#configure-styles - Add a Survey to a React Application [Електронний ресурс] // SurveyJS