МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
Кафедра **ПЗ**

ЗВІТ

До комплексної лабораторної роботи
З дисципліни: *“Комп'ютерна графіка”*

Лектор:
доцент каф. ПЗ
Левус Є. В.

Виконали:
ст. гр. ПЗ-32
Герман А.-С. Р.,
Місяйло О. О.

Прийняла:
доцент каф. ПЗ
Левус Є. В.

« ____ » _____ 2023 р.

∑= ____ _____.

Львів – 2023

Зміст

1. Завдання

Індивідуальний варіант

Варіант К6.

1) Побудувати фрактальні зображення:

1.1) Фрактал універсальна замкнена лінія Коха, для якої вводиться базова фігура (трикутник, квадрат) і відношення, що задаватиме розбиття сторони цієї фігури на ламану (посередині ламаної рівнобедрений трикутник без основи). Забезпечити можливість згенерувати зображення для різної кількості ітерацій.

1.2) Два різновиди алгебраїчних фракталів, що використовують тригонометричну функцію tg z. 11

2) Колірні моделі: CMYK і HSV. Змінити насиченість по пурпуровому кольору та доповнюючому до нього кольору. Робота з фрагментом зображення щодо перетворення моделі та зміни атрибуту кольору.

3) Реалізувати рух рівнобедреного трикутника, введеного за його двома вершинами основи і довжиною висоти, проведеної до основи. Рух генерується на основі дзеркального відображення відносно довільної прямої Ax+By+C=0, коефіцієнти якої вводяться користувачем.

Завдання лабораторних робіт

Л.1. Створення мокапів UI

Базуючись на wireframes та технологіях, які ви обрали, створіть UI-дизайн для шести виглядів системи (початковий екран, вікно для Л2, вікно для Л3, вікно для Л4, два вікна навчальних матеріалів).

Ці мокапи повинні бути імплементовані програмно в наступній частині лабораторного практикуму. Також продемонструйте у своїй розробці 5 евристик Нільсена (на вибір).

Л.2. Модуль «Фрактали»

Необхідно побудувати на екрані фрактальні зображення згідно варіанту та забезпечити їх збереження у файлах. Реалізувати введення користувачем усіх зазначених параметрів, що впливають на вигляд фракталів.

Л.3. Модуль «Колірні схеми»

Необхідно реалізувати:

- перечитування графічного зображення з одного простору кольорів в інший (простори задано у варіанті Завдання);
- відображення координат точок в кожному просторі кольорів;
- опрацювання атрибуту кольору згідно варіанту з обов'язковим використання перцепційних моделей;
- збереження змін у графічному файлі.

Л.4. Модуль «Рухомі зображення»

Необхідно реалізувати:

- відображення координатної площини і всіма відповідними позначеннями та підписами;
- зручне введення координат фігури (чи її конструктивних елементів) користувачем згідно варіанту;
- синтез рухомого зображення на основі афінних перетворень згідно варіанту з обов'язковим використанням матричних перетворень для реалізації комбінації афінних перетворень (рух – це зміна положення фігури в часі);
- динамічну зміну одиничного відрізку координатної площини;
- збереження початкового зображення у файл.

2. Теоретичні відомості

Опис функцій програми

Модуль "Фрактали"

1. Побудова універсальної замкненої лінії Коха:
    - Вибір базової фігури (трикутник, квадрат).
    - Введення відношення для розбиття сторони базової фігури на ламану.
    - Визначення кількості ітерацій для генерації зображення.
2. Побудова алгебраїчних фрактали:
    - Вибір типу алгебраїчного фрактала ($\tan(z^3)\cdot z, \tan(z^2)\cdot z$).
    - Налаштування зближення фракталу відносно центру.
3. Збереження фракталу у графічному файлі.

Модуль "Колірні схеми"

1. Перетворення зображення з RGB в CMYK в HSV в RGB:
    - Відображення координат точки зображення на яку натиснуто у колірних просторах CMYK та HSV .
2. Опрацювання атрибуту кольору:
    - Зміна насиченості пурпурового кольору та доповнюючого до нього кольору.
    - Визначення області зображення для застосування змін.
3. Збереження змін у графічному файлі.

Модуль "Рухомі зображення"

1. Відображення координатної площини:
    - Показ координатної площини з підписами та позначеннями.
    - Динамічна зміна довжини одиничного відрізку
2. Введення параметрів фігури, та прямої:
    - Зручне введення координат основи рівнобедерного трикутника та довжини висоти проведеної до основи.
    - Зручне введення коефінцієнтів прямої

3. Синтез рухомого зображення:
    - Побудова трикутника та прямої, дзеркальне відображення та рух відносно прямої
4. Збереження зображення у файл

Навчальні матеріали: Доступ до сторінки з детальним описом тем компютерної графіки використаних у програмі, описом роботи алгоритмів та функціоналу програмию.

Алгоритми фракталів

Алгоритм малювання фрактала за допомогою тангенсу (drawTanFractal і drawTan2Fractal):

1. Ініціалізувати параметри, такі як ширина та висота канвасу, максимальна кількість ітерацій, межі та кроки для обчислення точок.
2. Ітерувати через кожен піксель канвасу.
3. Для кожного пікселя визначити відповідний комплексний номер zx та zy.
4. Застосувати ітеративну функцію з тангенсами для обчислення нових значень tx та ty.
5. Перевірити, чи комплексне число виходить за межі визначеної області.
6. Визначити яскравість та колір для кожного пікселя в залежності від кількості ітерацій.
7. Створити SVG-елемент rect для кожного пікселя з визначеним коліром та додати його до канвасу.
8. Актуалізувати канвас після кожних updateInterval ітерацій та затримати виконання для асинхронного відображення.

Алгоритм малювання сніжинки Коха (drawSnowflake):

1. Ініціалізувати параметри, такі як кількість ітерацій та довжину сторони сніжинки.

2. Визначити початкові точки для кожного типу сніжинки (трикутник або квадрат).
3. Рекурсивно обчислити вершини кривої Коха для кожного бічного відрізка.
4. Визначити точки для середньої третини кожного відрізка та вершину виїмки.
5. Рекурсивно викликати функцію для обчислення шляху кривої Коха для кожного з чотирьох нових відрізків.
6. Створити SVG-елемент **path** із згенерованим шляхом та додати його до канвасу.

Анотація кольорових схем

Кольорові моделі є важливим інструментом у графіці, друкарстві, веб-дизайні та інших областях, де використовується кольорова інформація. Два поширених типи моделей - CMYK та HSV - пропонують різні підходи до представлення кольору.

1. Модель CMYK (Cyan, Magenta, Yellow, Black):

- *Практичне застосування:* Використовується в друкарстві для створення кольорових зображень. Кожен колір представлений відтінком одного з чотирьох базових кольорів.
- *Переваги:*
    - Ідеальна для друкарських потреб, оскільки дозволяє точно визначити співвідношення кольорів на друкарському матеріалі.
    - Дозволяє відтворити багато кольорів, включаючи темні та насичені.
- *Недоліки:*
    - Неідеальна для відображення кольорів на екрані, оскільки багато моніторів використовують RGB-модель.

- Може бути складно досягти точного подання деяких яскравих кольорів.

2. Модель HSV (Hue, Saturation, Value):

- *Практичне застосування:* Зручно використовується у графіці та дизайні для регулювання кольорів на основі їхньої відтінку, насиченості та яскравості.
- *Переваги:*
  - Легко використовується для користувачів, оскільки дозволяє змінювати кольори за звичайними поняттями, такими як "відтінок" та "яскравість".
  - Ефективна для роботи зі змінами відтінку та насиченості.
- *Недоліки:*
  - Може викликати проблеми у точному відтворенні кольорів для друку через відсутність чіткого визначення кольору.
  - Не завжди ідеальна для роботи з технічними деталями кольорового відтворення.

Кожна з цих моделей має свої унікальні застосування та переваги, і вибір залежить від конкретних потреб проекту чи завдання.

Оптимальний матричний вираз афінних перетворень

Для здійснення дзеркального перетворення трикутника відносно прямої $Ax+By+C=0$, можна скористатися матричним виразом афінних перетворень. Основна ідея - використати матрицю перетворення для визначення нових координат точок трикутника. Ось як це можна зробити:

Нехай $P(x,y)$ - координати точки трикутника, а M - матриця дзеркального перетворення.

Матриця дзеркального перетворення для прямої $Ax+By+C=0$ виглядає наступним чином:

$$M = \frac{1}{A^2+B^2}\begin{bmatrix} B^2 - A^2 & 2AB \\ 2AB & A^2 - B^2 \end{bmatrix}$$

Тоді нові координати точок трикутника $P1(x1,y1)$, $P2(x2,y2)$ і $P3(x3,y3)$ можна знайти за допомогою матричного множення:

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \end{bmatrix} = \frac{1}{A^2+B^2}\begin{bmatrix} B^2 - A^2 & 2AB \\ 2AB & A^2 - B^2 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$$

Реалізація графічного режиму

Технологія реалізації: HTML, SVG, та JavaScript з використанням Canvas

Обґрунтування технології реалізації:

- HTML: Використовується для структуризації та розмітки веб-сторінки.
- SVG (Scalable Vector Graphics): Використовується для побудови фракталів, оскільки це XML-базований формат, який дозволяє легко створювати векторні зображення та масштабувати їх без втрати якості.
- JavaScript: Використовується для динамічного керування елементами сторінки та обробки подій.

Основні інструменти для графіки:

1. Налаштування графіки:
    - HTML та SVG: Використовуються для створення структури сторінки та визначення елементів SVG.
    - JavaScript: Використовується для взаємодії з елементами HTML та SVG, зміни їх параметрів та анімацій.
2. Полотно графічного виведення:
    - SVG: Надає векторне полотно для зображення фракталів. Його можна легко масштабувати та анімувати.
    - Canvas: Використовується для роботи з растровим графічним полотном, що дозволяє ефективно малювати та обробляти

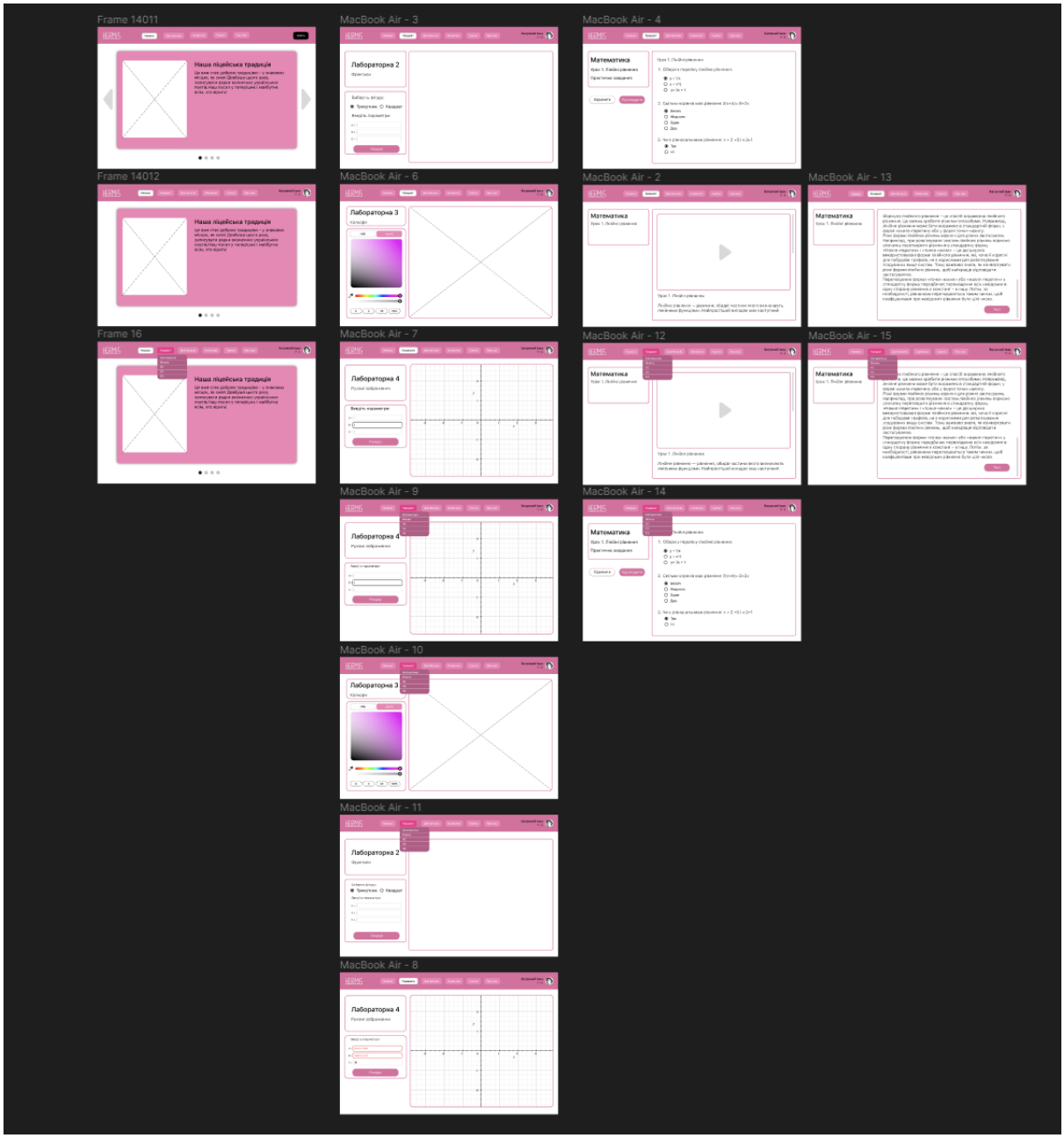
растрову графіку, також може використовуватися для генерації фракталів.

3. Примітиви:
    - SVG: Дозволяє використовувати примітиви, такі як <circle>, <rect>, <line>, і <path>, для конструювання графічних елементів та фракталів.
    - Canvas: Надає методи для малювання примітивів, таких як fillRect, strokeRect, arc, і lineTo, для створення графічних об'єктів.
4. Робота з колірними схемами та афінними перетвореннями:
    - JavaScript та Canvas: Використовуються для зміни кольорів об'єктів та виконання афінних перетворень над графічними елементами.
    - SVG: Дозволяє вказати кольори за допомогою атрибуту fill та використовувати трансформації для афінних перетворень.

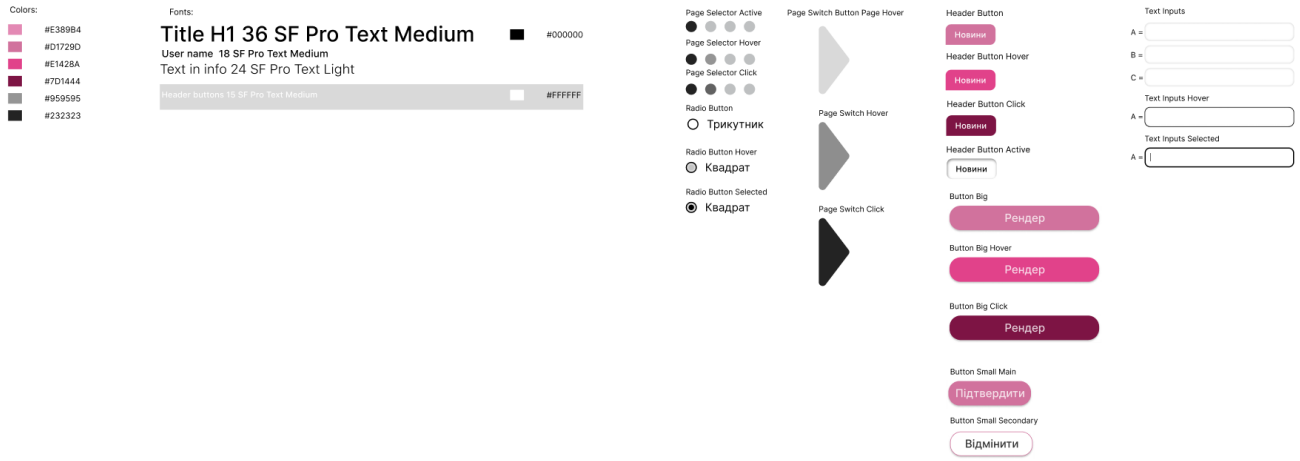
Ця комбінація технологій дозволяє легко створювати динамічні та ефективні графічні програми з використанням векторної та растрової графіки.

4. Результати

Wireflow:

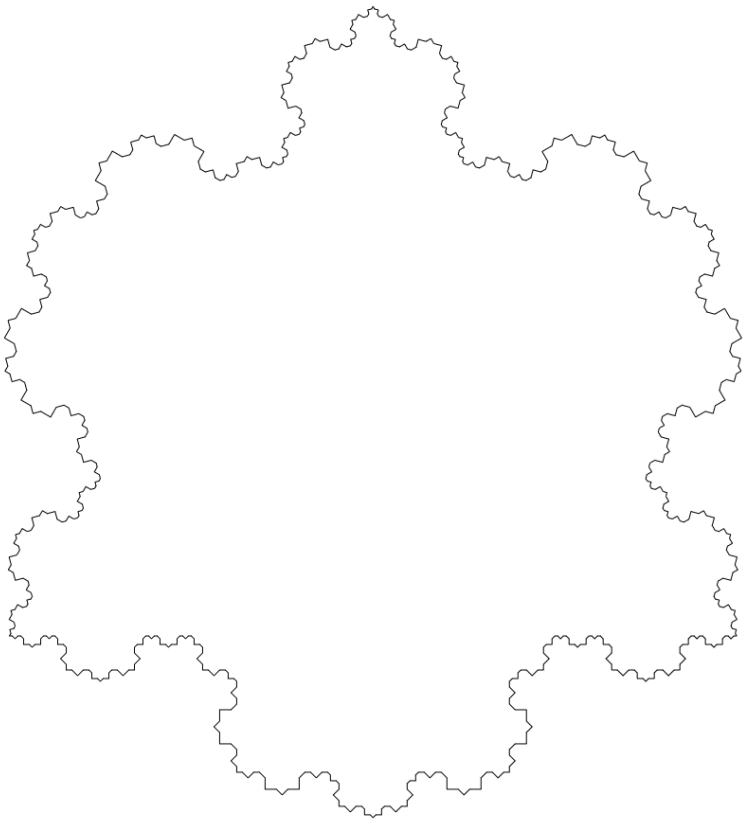


UI Kit:

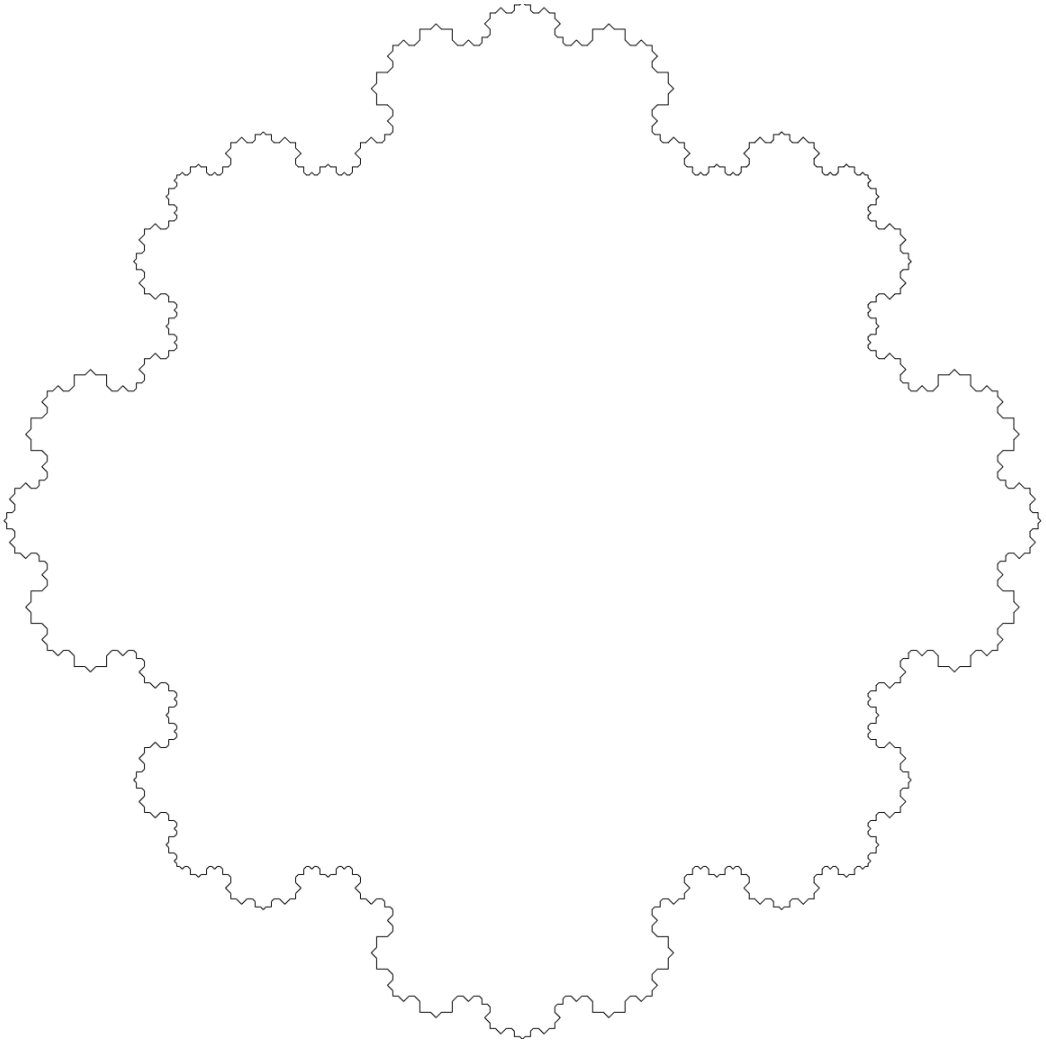


Побудовані фрактали:

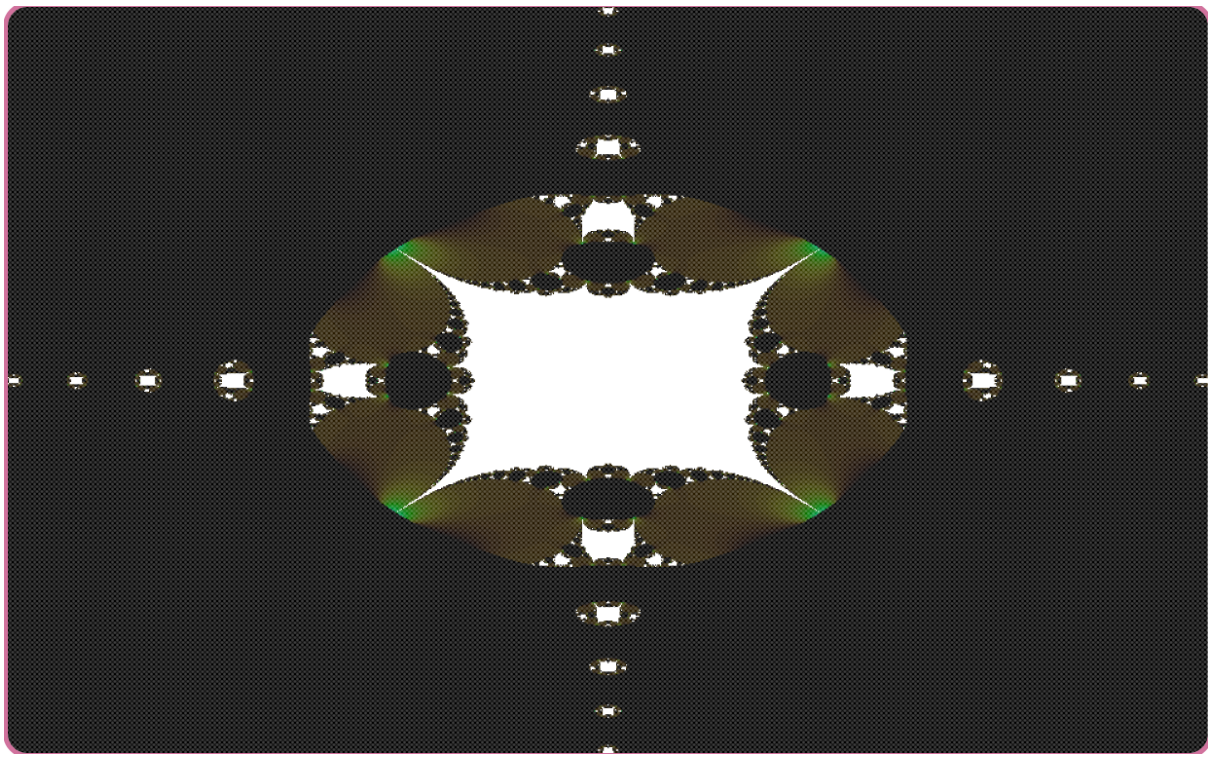
- Сніжинка коха на основі трикутника:



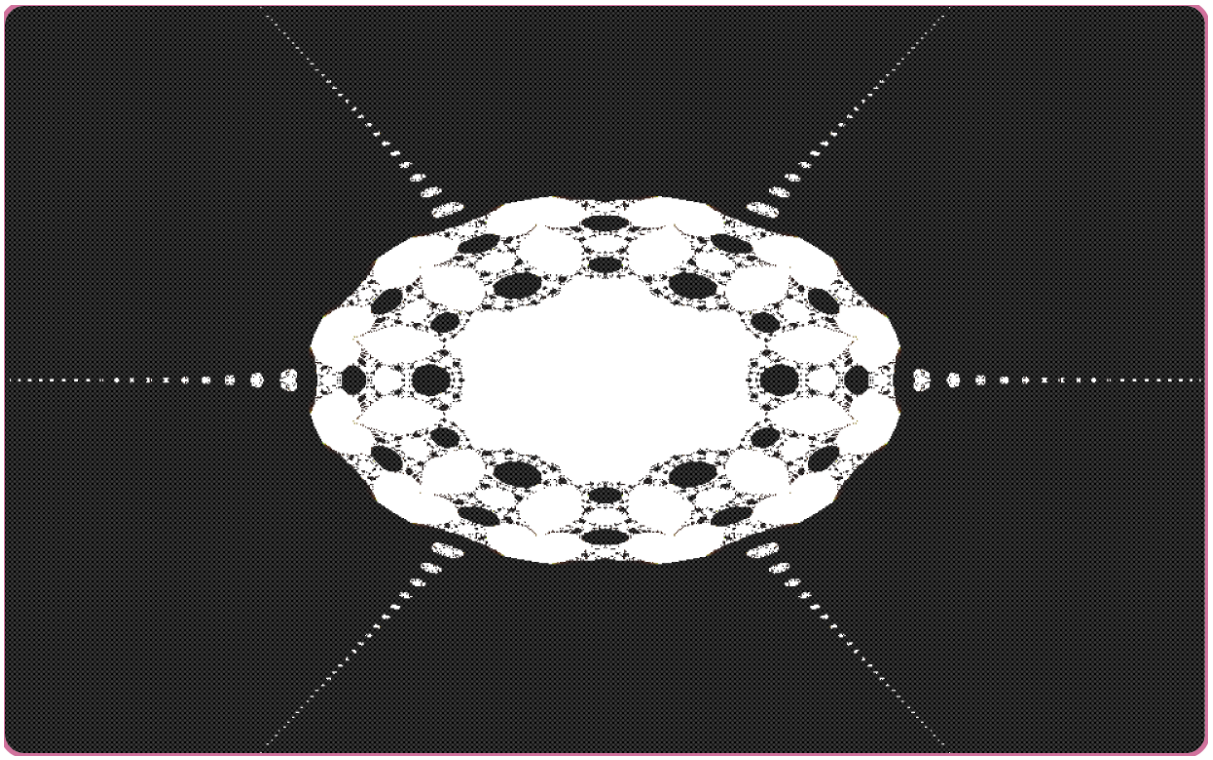
- Сніжинка коха на основі квадрату



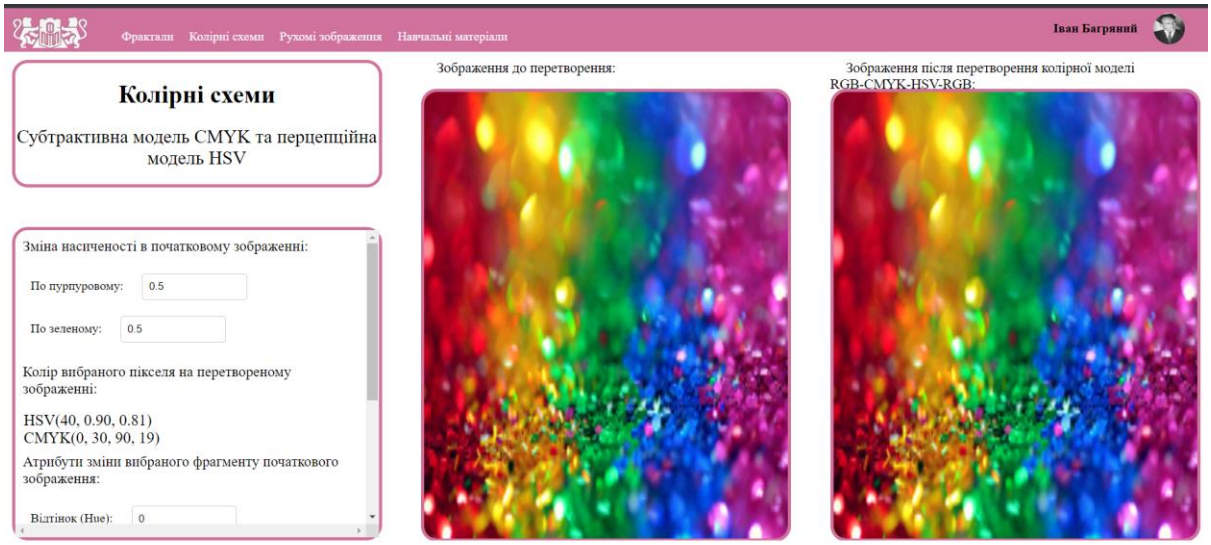
- Алгебраїчний фрактал на основі $\tan(z^2) \cdot z$



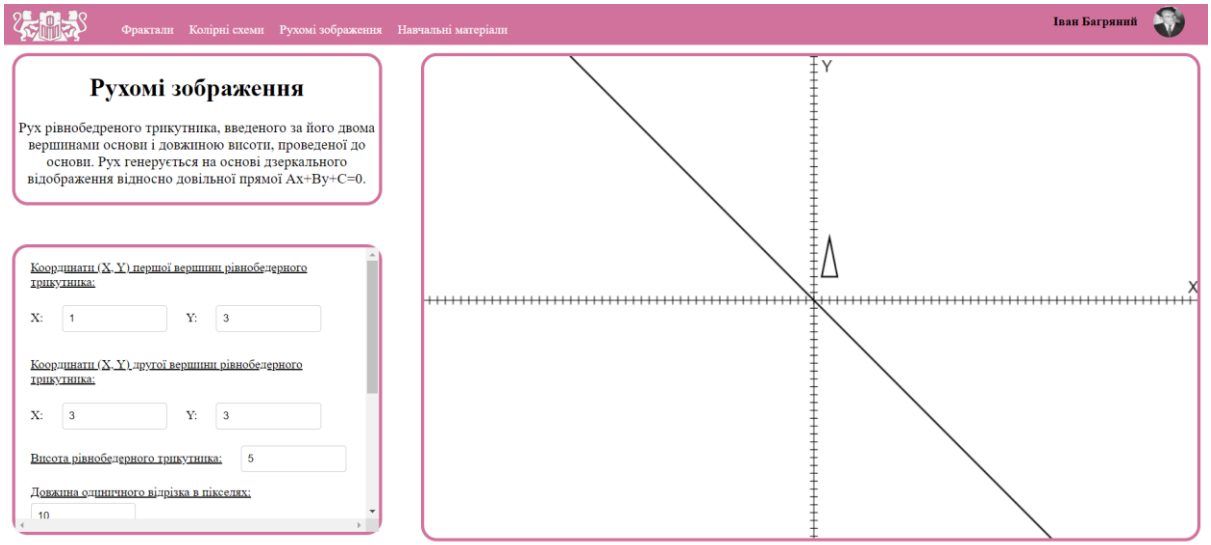
- Алгебраїчний фрактал на основі $\tan(z^3) \cdot z$



Вигляд програми для модуля «Кольори»:



Вигляд програми для модуля «Рухомі зображення»:



Код програми:

script.js:

```
document.addEventListener("DOMContentLoaded", function () {

    const canvas = document.getElementById("fractal");
    const renderBut = document.getElementById("renderButton");
    var scale = 1;

//Navigation
    document.getElementById('fractals-header').addEventListener('click', () =>
{
        document.getElementById('fractals-page').style.display = 'grid';
        document.getElementById('colors-page').style.display = 'none';
        document.getElementById('movement-page').style.display = 'none';
        document.getElementById('learning-materials').style.display = 'none';
    });
    document.getElementById('colors-header').addEventListener('click', () => {
        document.getElementById('fractals-page').style.display = 'none';
        document.getElementById('colors-page').style.display = 'grid';
        document.getElementById('movement-page').style.display = 'none';
        document.getElementById('learning-materials').style.display = 'none';
    });
    document.getElementById('movement-header').addEventListener('click', () =>
{
        document.getElementById('fractals-page').style.display = 'none';
        document.getElementById('colors-page').style.display = 'none';
        document.getElementById('movement-page').style.display = 'grid';
        document.getElementById('learning-materials').style.display = 'none';
    });
    document.getElementById('learning-materials-
header').addEventListener('click', () => {
        document.getElementById('fractals-page').style.display = 'none';
        document.getElementById('colors-page').style.display = 'none';
        document.getElementById('movement-page').style.display = 'none';
        document.getElementById('learning-materials').style.display = 'grid';
    });



//Complex calculations
    function complexMultiply(a, b) {
        const real = a[0] * b[0] - a[1] * b[1];
        const imag = a[0] * b[1] + a[1] * b[0];
        return [real, imag];
    }
```

```
    function complexTan(z) {
        const realPart = (Math.sin(2 * z[0]) / (Math.cos(2 * z[0]) +
Math.cosh(2 * z[1])));
        const imagPart = (Math.sinh(2 * z[1]) / (Math.cos(2 * z[0]) +
Math.cosh(2 * z[1])));
        return [realPart, imagPart];
    }

//tan(z^2)*z Fractal
    async function drawTanFractal() {
        while (canvas.firstChild) {
            canvas.removeChild(canvas.firstChild);
        }
        const updateInterval = 5000;
        let iterationCount = 0;
        const width = canvas.clientWidth;
        const height = canvas.clientHeight;
        const maxIterations = 500;
        const xmin = -4/scale;
        const xmax = 4/scale;
        const ymin = -4/scale;
        const ymax = 4/scale;
        const dx = (xmax - xmin) / width;
        const dy = (ymax - ymin) / height;
        for (let x = 0; x < width; x++) {
            for (let y = 0; y < height; y++) {
                const zx = x * dx + xmin;
                const zy = y * dy + ymin;
                let n = 0;
                let tx = zx;
                let ty = zy;
                while (n < maxIterations){
                    txcur = tx;
                    tx = complexMultiply(complexTan(complexMultiply([tx, ty],
[tx, ty])), [tx, ty])[0];
                    ty = complexMultiply(complexTan(complexMultiply([txcur,
ty], [txcur, ty])), [txcur, ty])[1];
                    if(tx * tx + ty * ty > 4) {
                        break;
                    }
                    n++;
                }
                const brightness = Math.floor((n / maxIterations) * 100);
                const color = `hsl(${n % 360}, 100%, ${brightness}%)`;

                const pixel =
document.createElementNS('http://www.w3.org/2000/svg', 'rect');
                pixel.setAttribute('x', x);
```

```
                pixel.setAttribute('y', y);
                pixel.setAttribute('width', 1);
                pixel.setAttribute('height', 1);
                pixel.setAttribute('fill', color);
                canvas.appendChild(pixel);
                iterationCount++;
                if (iterationCount >= updateInterval) {
                    iterationCount = 0;
                    await new Promise(resolve => setTimeout(resolve, 0));
                }
            }
        }
    }
//tan(z^3)*z Fractal
    async function drawTan2Fractal() {
        while (canvas.firstChild) {
            canvas.removeChild(canvas.firstChild);
        }
        const updateInterval = 5000;
        let iterationCount = 0;
        const width = canvas.clientWidth;
        const height = canvas.clientHeight;
        const maxIterations = 200;
        const xmin = -4/scale;
        const xmax = 4/scale;
        const ymin = -4/scale;
        const ymax = 4/scale;
        const dx = (xmax - xmin) / width;
        const dy = (ymax - ymin) / height;
        for (let x = 0; x < width; x++) {
            for (let y = 0; y < height; y++) {
                const zx = x * dx + xmin;
                const zy = y * dy + ymin;
                let n = 0;
                let tx = zx;
                let ty = zy;

                while (n < maxIterations){
                    txcur = tx;
                    tx = complexMultiply(complexTan(complexMultiply([tx, ty],
complexMultiply([tx, ty], [tx, ty]))), [tx, ty])[0];
                    ty = complexMultiply(complexTan(complexMultiply([txcur,
ty], complexMultiply([txcur, ty], [txcur, ty]))), [txcur, ty])[1];
                    if(tx * tx + ty * ty > 4) {
                        break;
                    }
                    n++;
                }
```

```
                const brightness = Math.floor((n / maxIterations) * 100);
                const color = `hsl(${n % 360}, 100%, ${brightness}%)`;

                const pixel =
document.createElementNS('http://www.w3.org/2000/svg', 'rect');
                pixel.setAttribute('x', x);
                pixel.setAttribute('y', y);
                pixel.setAttribute('width', 1);
                pixel.setAttribute('height', 1);
                pixel.setAttribute('fill', color);
                canvas.appendChild(pixel);
                iterationCount++;
                if (iterationCount >= updateInterval) {
                    iterationCount = 0;
                    await new Promise(resolve => setTimeout(resolve, 0));
                }
            }
        }
    }
//Saving as file
    function saveSvgAsImage() {
        if(document.getElementById('colors-page').style.display === 'grid'){
            var canvas = document.getElementById('canvasColors');
            var dataUrl = canvas.toDataURL();
            var link = document.createElement('a');
            link.download = 'colors_image.png';
            link.href = dataUrl;
            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);
        }
        else if (document.getElementById('fractals-page').style.display ===
'grid') {
            const svg = document.getElementById('fractal');
            const svgData = new XMLSerializer().serializeToString(svg);
            const blob = new Blob([svgData], { type: "image/svg+xml" });
            const element = document.createElement("a");
            element.download = "w3c.svg";
            element.href = window.URL.createObjectURL(blob);
            element.click();
            element.remove();
        }else {
            var canvas = document.getElementById('movementCanvas');
            var dataUrl = canvas.toDataURL();
            var link = document.createElement('a');
            link.download = 'movement_image.png';
            link.href = dataUrl;
            document.body.appendChild(link);
```

```
            link.click();
            document.body.removeChild(link);
        }

    }

    document.getElementById('saveSVGButton1').addEventListener('click',
saveSvgAsImage);
    document.getElementById('saveSVGButton2').addEventListener('click',
saveSvgAsImage);
    document.getElementById('saveSVGButton3').addEventListener('click',
saveSvgAsImage);

    const iterationsValue = document.querySelector("#iterationsValue");
    const iterationsInput = document.querySelector("#iterations");
    const scaleValue = document.querySelector("#scaleValue");
    scaleValue.textContent = 1;
    const scaleInput = document.querySelector("#scale");
    iterationsValue.textContent = iterationsInput.value;
    const divisionRatioInput = document.querySelector("#divisionRatio");
    const divisionRatioValue = document.querySelector("#divisionRatioValue");
    divisionRatioValue.textContent = divisionRatioInput.value;

    divisionRatioInput.addEventListener("input", (event) => {
        divisionRatioValue.textContent = event.target.value;
        drawSnowflake();
    });

    iterationsInput.addEventListener("input", (event) => {
        iterationsValue.textContent = event.target.value;
    });

    scaleInput.addEventListener("input", (event) => {
        scaleValue.textContent = event.target.value;
        scale = event.target.value;
    });


    const fractalTypeSelect = document.querySelector("#fractalTypeSelect");
    const kochInputs = document.querySelector("#kochInputs");
    const tgInputs = document.querySelector("#tgInputs");
    const radioTriangle = document.querySelector('#radioTriangle');
    const radioCube = document.querySelector('#radioCube');


//Koch on parameters change
    fractalTypeSelect.addEventListener("change", (event) => {
        var fractalType = event.target.value;
```

```
        switch(fractalType){
            case 'koch-snowflake':
                kochInputs.hidden = false;
                tgInputs.hidden = true;
                break;
            default:
                if(event.target.value === 'tan-frac') {
                    document.getElementById('ffEq').hidden = false;
                    document.getElementById('sfEq').hidden = true;
                } else {
                    document.getElementById('ffEq').hidden = true;
                    document.getElementById('sfEq').hidden = false;
                }
                kochInputs.hidden = true;
                tgInputs.hidden = false;
                break;
        }
    })

    iterationsInput.addEventListener("input", drawSnowflake);

//Koch snowflake algorythm on parameters change

    function drawKochCurve(x1, y1, x2, y2, iterations) {
        if (iterations === 0) {
            return `L ${x2} ${y2} `;
        }
        const dx = x2 - x1;
        const dy = y2 - y1;
        const length = Math.sqrt(dx * dx + dy * dy);
        const spikeLength = length * divisionRatioInput.value;
        const sideLength = (length - spikeLength) / 2;

        const xA = x1 + dx * sideLength / length;
        const yA = y1 + dy * sideLength / length;

        const xB = x2 - dx * sideLength / length;
        const yB = y2 - dy * sideLength / length;

        const height = spikeLength / 2;

        const direction = radioCube.checked ? -1 : 1;

        const xC = (xA + xB) / 2 - direction * height * dy / length;
        const yC = (yA + yB) / 2 + direction * height * dx / length;

        return drawKochCurve(x1, y1, xA, yA, iterations - 1) +
               drawKochCurve(xA, yA, xC, yC, iterations - 1) +
```

```
                drawKochCurve(xC, yC, xB, yB, iterations - 1) +
                drawKochCurve(xB, yB, x2, y2, iterations - 1);
    }


    function drawSnowflake() {
        while (canvas.firstChild) {
            canvas.removeChild(canvas.firstChild);
        }
        const iterations =
parseInt(document.getElementById("iterations").value);
        const sideLength = Math.min(canvas.width.baseVal.value,
canvas.height.baseVal.value) * 2 / 3;
        let pathData = "";

        if (radioTriangle.checked) {
            const height = sideLength * Math.sqrt(3) / 2;
            const yOffset = (canvas.height.baseVal.value - height) / 2;
            const xOffset = (canvas.width.baseVal.value - sideLength) / 2;
            pathData += `M ${xOffset} ${yOffset + height} `;
            pathData += drawKochCurve(xOffset, yOffset + height, xOffset +
sideLength, yOffset + height, iterations);
            pathData += drawKochCurve(xOffset + sideLength, yOffset + height,
xOffset + sideLength / 2, yOffset, iterations);
            pathData += drawKochCurve(xOffset + sideLength / 2, yOffset,
xOffset, yOffset + height, iterations);
        } else if (radioCube.checked) {
            const xOffset = (canvas.width.baseVal.value - sideLength) / 2;
            const yOffset = (canvas.height.baseVal.value - sideLength) / 2;
            pathData += `M ${xOffset} ${yOffset} `;
            pathData += drawKochCurve(xOffset, yOffset, xOffset + sideLength,
yOffset, iterations);
            pathData += drawKochCurve(xOffset + sideLength, yOffset, xOffset +
sideLength, yOffset + sideLength, iterations);
            pathData += drawKochCurve(xOffset + sideLength, yOffset +
sideLength, xOffset, yOffset + sideLength, iterations);
            pathData += drawKochCurve(xOffset, yOffset + sideLength, xOffset,
yOffset, iterations);
        }

        const pathElement =
document.createElementNS("http://www.w3.org/2000/svg", "path");
        pathElement.setAttribute("d", pathData);
        pathElement.setAttribute("fill", "none");
        pathElement.setAttribute("stroke", "black");
        canvas.appendChild(pathElement);
    }
```

```
    document.getElementById("radioTriangle").addEventListener("change",
drawSnowflake);
    document.getElementById("radioCube").addEventListener("change",
drawSnowflake);
    drawSnowflake();


    renderBut.addEventListener("click", () => {
        var fractalType = document.getElementById('fractalTypeSelect').value;
        switch(fractalType){
            case 'koch-snowflake':
                drawSnowflake();
                break;
            case 'tan-frac':
                drawTanFractal();
                break;
            case 'tricorn-frac':
                drawTan2Fractal();
                break;
            case 'mandelbrot-frac':
                drawMandelbrotFractal();
                break;
            default:
                break;
        }
    });
});
```

colorScript.js:

```
document.addEventListener("DOMContentLoaded", function () {


//RGB to HSV transission
function RGBToHSV(r, g, b) {
    r /= 255, g /= 255, b /= 255;

    var max = Math.max(r, g, b), min = Math.min(r, g, b);
    var h, s, v = max;

    var d = max - min;
    s = max == 0 ? 0 : d / max;

    if (max == min) {
      h = 0;
    } else {
```

```
      if (max === r) {
        h = (g - b) / d + (g < b ? 6 : 0);
      } else if (max === g) {
        h = (b - r) / d + 2;
      } else {
        h = (r - g) / d + 4;
      }

      h /= 6;
    }

    return { h, s, v };
}

//HSV to RGB transission
function    HSVToRGB(h, s, v) {
    var r, g, b;

    var i = Math.floor(h * 6);
    var f = h * 6 - i;
    var p = v * (1 - s);
    var q = v * (1 - f * s);
    var t = v * (1 - (1 - f) * s);

    switch (i % 6) {
     case 0: r = v, g = t, b = p; break;
     case 1: r = q, g = v, b = p; break;
     case 2: r = p, g = v, b = t; break;
     case 3: r = p, g = q, b = v; break;
     case 4: r = t, g = p, b = v; break;
     case 5: r = v, g = p, b = q; break;
    }

    return [ r * 255, g * 255, b * 255 ];
}

//RGB to CMYK transission
function rgbToCmyk(r, g, b) {
    const red = r / 255;
    const green = g / 255;
    const blue = b / 255;

    const black = 1 - Math.max(red, green, blue);

    const cyan = (1 - red - black) / (1 - black);
    const magenta = (1 - green - black) / (1 - black);
    const yellow = (1 - blue - black) / (1 - black);
```

```
    const c = Math.round(cyan * 100);
    const m = Math.round(magenta * 100);
    const y = Math.round(yellow * 100);
    const k = Math.round(black * 100);

    return { c, m, y, k };
}

//CMYK to RGB transission
function cmykToRgb(c, m, y, k) {
    const cyan = c / 100;
    const magenta = m / 100;
    const yellow = y / 100;
    const black = k / 100;

    const red = 255 * (1 - cyan) * (1 - black);
    const green = 255 * (1 - magenta) * (1 - black);
    const blue = 255 * (1 - yellow) * (1 - black);

    return { red, green, blue };
}

const canvasColors = document.getElementById("canvasColors");
const canvasColors2 = document.getElementById("canvasColorsResult");
const ctxColors = canvasColors.getContext("2d");
const imgColors = new Image();
imgColors.src = "pexels-alexander-grey-1191710.jpg";
imgColors.onload = function() {
    ctxColors.drawImage(imgColors, 0, 0, canvasColors.width,
canvasColors.height);
};


document.getElementById("changeColorButton").addEventListener("click", () => {
    const imageData = ctxColors.getImageData(0, 0, canvasColors.width,
canvasColors.height);
    const data = imageData.data;

    for (let i = 0; i < data.length; i += 4) {
       const r = data[i];
       const g = data[i + 1];
       const b = data[i + 2];
       const cmyk = rgbToCmyk(r, g, b);
       const rgb1 = cmykToRgb(cmyk.c, cmyk.m, cmyk.y, cmyk.k);
       const hsv = RGBToHSV(rgb1.red, rgb1.green, rgb1.blue);
       const rgb = HSVToRGB(hsv.h, hsv.s, hsv.v);
       data[i] = rgb[0];
       data[i + 1] = rgb[1];
```

```
        data[i + 2] = rgb[2];
    }
    canvasColors2.getContext("2d").putImageData(imageData, 0, 0);
});



//Displaying color coordinates on click
const hsvText = document.getElementById("color-hsv");
const cmykText = document.getElementById("color-cmyk");
function pick(event){
    const bounding = canvasColors2.getBoundingClientRect();

    const xScale = canvasColors2.width / bounding.width;
    const yScale = canvasColors2.height / bounding.height;


    const x = (event.clientX - bounding.left) * xScale;
    const y = (event.clientY - bounding.top) * yScale;

    const pixel = canvasColors2.getContext("2d").getImageData(x, y, 1, 1);
    const data = pixel.data;

    const r = data[0];
    const g = data[1];
    const b = data[2];
    const cmyk = rgbToCmyk(r, g, b);

    const hsv = RGBToHSV(r, g, b);

    const cmykVal = `CMYK(${cmyk.c}, ${cmyk.m}, ${cmyk.y}, ${cmyk.k })`;
    const hsvVal = `HSV(${(hsv.h*360).toFixed()}, ${hsv.s.toFixed(2)},
${hsv.v.toFixed(2)})`;

    cmykText.textContent = cmykVal;
    hsvText.textContent = hsvVal;
}
canvasColors2.addEventListener("click", (event) => pick(event));

document.getElementById("resetToDefault").addEventListener("click", () => {
    imgColors.src = "pexels-alexander-grey-1191710.jpg";
    imgColors.onload = function() {
        ctxColors.drawImage(imgColors, 0, 0, canvasColors.width,
canvasColors.height);
    };
});

const greenSaturInput = document.querySelector("#green-saturation");
var greenSatur = 1;
```

```
var magentaSatur = 1;

greenSaturInput.addEventListener("input", (event) => {
    greenSatur = event.target.value;
    adjustSaturationOnCanvas(greenSatur, magentaSatur);
});

const purpleSaturInput = document.querySelector("#magenta-saturation");
console.log(purpleSaturInput.value)
purpleSaturInput.addEventListener("input", (event) => {
    magentaSatur = event.target.value;
    adjustSaturationOnCanvas(greenSatur, magentaSatur);

});

//Check if pixel magenta
function isMagenta(hsv) {
    const magentaHueMin = 301;
    const magentaHueMax = 360;
    return hsv.h >= magentaHueMin / 360 && hsv.h <= magentaHueMax / 360;
}

//Check if pixel green
function isGreen(hsv) {
    const greenHueMin = 121;
    const greenHueMax = 180;
    return hsv.h >= greenHueMin / 360 && hsv.h <= greenHueMax / 360;
}

//Ajusting saturation on input
function adjustSaturationOnCanvas(greenSaturation, magentaSaturation) {
    const imageData = ctxColors.getImageData(0, 0, canvasColors.width,
canvasColors.height);
    const data = imageData.data;

    for (let i = 0; i < data.length; i += 4) {
        let r = data[i];
        let g = data[i + 1];
        let b = data[i + 2];

        let hsv = RGBToHSV(r, g, b);

        if (isGreen(hsv)) {
            hsv.s = greenSaturation;
        } else if (isMagenta(hsv)) {
            hsv.s = magentaSaturation;
        }
```

```
        let rgb = HSVToRGB(hsv.h, hsv.s, hsv.v);

        data[i] = rgb[0];
        data[i + 1] = rgb[1];
        data[i + 2] = rgb[2];
    }

    canvasColors.getContext("2d").putImageData(imageData, 0, 0);
}

});
```

movmentScript.js:

```
document.addEventListener("DOMContentLoaded", function () {

    var moving = false;
    const canvas = document.getElementById("movementCanvas");
    const ctx = canvas.getContext('2d');
    ctx.lineWidth = 2;
    ctx.strokeStyle = 'black';
    canvas.width = document.getElementById("movement-container").offsetWidth;
    canvas.height = document.getElementById("movement-
container").offsetHeight;
    const halfWidth = canvas.width / 2;
    const halfHeight = canvas.height / 2;
    ctx.translate(halfWidth, halfHeight);
    ctx.scale(1, -1);
    var intervalInPixel = 10;
    var A;
    var B;
    var C;
    var currentA;
    var currentB;
    var currentC;
    var currentBaseApex1X;
    var currentBaseApex1Y;
    var currentBaseApex2X;
    var currentBaseApex2Y;
    var currentTopApexX;
    var currentTopApexY;
    var currentHeight;
    var firstVertexX;
    var firstVertexY;
    var secondVertexX;
    var secondVertexY;
```

```
    var heightInput;
    var isDrawn = false;
    drawAxis();


//Drawing Triangle
    function drawTriangleByParameters()
    {
        ctx.clearRect(-canvas.width /2 , -canvas.height /2 , canvas.width,
canvas.height);
        drawAxis();
        drawTriangle();
        drawLine();
        isDrawn = true;
    }
//Moving and reflecting the Triangle
    function reflectTriangleByParameters()
    {
        ctx.clearRect(-canvas.width /2 , -canvas.height /2 , canvas.width,
canvas.height);
        drawAxis();
        drawLine();
        reflectTriangle(currentBaseApex1X, currentBaseApex1Y,
currentBaseApex2X, currentBaseApex2Y, currentTopApexX, currentTopApexY,
currentA, currentB, currentC);
        isDrawn = true;
    }

    document.querySelector("#intervalInPixel").addEventListener("input",
(event) => {
        intervalInPixel = parseInt(event.target.value, 10);
        if(intervalInPixel < 3){
            alert("Довжина одиничного відрізка мусить бути більшою 2px")
            intervalInPixel = 3;
            event.target.value = 3;
        }
        console.log(intervalInPixel);
        if(isDrawn){
            ctx.clearRect(-canvas.width /2 , -canvas.height /2 , canvas.width,
canvas.height);
            drawAxis();
            drawIsoscelesTriangle(firstVertexX * intervalInPixel, firstVertexY
* intervalInPixel, secondVertexX * intervalInPixel, secondVertexY *
intervalInPixel, heightInput * intervalInPixel);
            drawLinearEquation(A, B, C * intervalInPixel);
        } else{
            ctx.clearRect(-canvas.width /2 , -canvas.height /2 , canvas.width,
canvas.height);
            drawAxis();
```

```
        }

    });

    document.getElementById('drawButton').addEventListener('click', function()
{
        drawTriangleByParameters()
    });

    function reflectPeriodically() {
        if(moving)
            reflectTriangleByParameters();
        setTimeout(reflectPeriodically, 500);
    }
    reflectPeriodically();
    document.getElementById('moveButton').addEventListener('click', function()
{
        moving = !moving;
    });


    function drawTriangle() {

        firstVertexX =
parseFloat(document.getElementById('firstVertexX').value, 10);
        firstVertexY =
parseFloat(document.getElementById('firstVertexY').value, 10);
        secondVertexX =
parseFloat(document.getElementById('secondVertexX').value, 10);
        secondVertexY =
parseFloat(document.getElementById('secondVertexY').value, 10);
        heightInput =
parseFloat(document.getElementById('trianglesHeight').value, 10);
        drawIsoscelesTriangle(firstVertexX * intervalInPixel, firstVertexY *
intervalInPixel, secondVertexX * intervalInPixel, secondVertexY *
intervalInPixel, heightInput * intervalInPixel);

    }



    function drawIsoscelesTriangle(baseApex1X, baseApex1Y, baseApex2X,
baseApex2Y, height) {
        if (canvas.getContext) {
            // Calculate the midpoint of the base
            var midX = (baseApex1X + baseApex2X) / 2;
            var midY = (baseApex1Y + baseApex2Y) / 2;

            // Calculate the top apex coordinates
```

```
            var dx = baseApex2X - baseApex1X;
            var dy = baseApex2Y - baseApex1Y;
            var angle = Math.atan2(dy, dx);

            var topApexX = midX - height * Math.sin(angle);
            var topApexY = midY + height * Math.cos(angle);

            // Draw the triangle
            ctx.beginPath();
            ctx.moveTo(baseApex1X, baseApex1Y);
            ctx.lineTo(baseApex2X, baseApex2Y);
            ctx.lineTo(topApexX, topApexY);
            ctx.closePath();

            // Style the triangle
            ctx.lineWidth = 2;
            ctx.strokeStyle = '#000000';
            ctx.stroke();

            currentBaseApex1X = baseApex1X;
            currentBaseApex1Y = baseApex1Y;
            currentBaseApex2X = baseApex2X;
            currentBaseApex2Y = baseApex2Y;
            currentTopApexX = topApexX;
            currentTopApexY = topApexY;
            currentHeight = height;
        }
    }


    //Drawing XY axis
    function drawAxis() {
        const width = ctx.canvas.width;
        const height = ctx.canvas.height;

        // Draw X-axis
        ctx.beginPath();
        ctx.moveTo(-width / 2, 0);
        ctx.lineTo(width / 2, 0);

        // Draw Y-axis
        ctx.moveTo(0, -height / 2);
        ctx.lineTo(0, height / 2);

        // Drawing marks on the axes, starting from the center
        for (let x = 0; x <= width / 2; x += intervalInPixel) {
            ctx.moveTo(x, -5);
            ctx.lineTo(x, 5);
            if (x !== 0) { // Ensure we don't draw at the center twice
```

```
                ctx.moveTo(-x, -5);
                ctx.lineTo(-x, 5);
            }
        }

        // Drawing marks on the Y axis, starting from the center
        for (let y = 0; y <= height / 2; y += intervalInPixel) {
            ctx.moveTo(-5, y);
            ctx.lineTo(5, y);
            if (y !== 0) { // Ensure we don't draw at the center twice
                ctx.moveTo(-5, -y);
                ctx.lineTo(5, -y);
            }
        }

        ctx.strokeStyle = 'black';
        ctx.stroke();
        ctx.closePath();

        // Drawing 'X' and 'Y' at the ends of the axes
        // Adjust positions if needed
        ctx.font = "20px Arial";
        ctx.scale(1, -1);  // Temporary flip to draw text upright
        ctx.fillText("X", width / 2 - 20, -10);
        ctx.fillText("Y", 10, -height / 2 + 20);
        ctx.scale(1, -1);  // Flip back
    }

//Drawing the linear equation
    function drawLine() {
        A = parseFloat(document.getElementById('lineA').value);
        B = parseFloat(document.getElementById('lineB').value);
        C = parseFloat(document.getElementById('lineC').value);
        currentA = A;
        currentB = B;
        currentC = C * intervalInPixel;
        drawLinearEquation(A, B, C * intervalInPixel);
    }



    function drawLinearEquation(A, B, C) {
        ctx.beginPath();

            let x1 = -halfWidth;
            let y1 = (-C - A * x1) / B;
            let x2 = halfWidth;
            let y2 = (-C - A * x2) / B;
```

```
        ctx.beginPath();
        ctx.moveTo(x1, y1);
        ctx.lineTo(x2, y2);
        ctx.stroke();
    }

    function reflectTriangle(x1, y1, x2, y2, x3, y3, A, B, C) {
        // Normalize the line coefficients
        const norm = Math.sqrt(A * A + B * B);
        A /= norm;
        B /= norm;
        C /= norm;

        // Construct the reflection matrix
        const reflectionMatrix = [
            [1 - 2 * A * A, -2 * A * B, -2 * A * C],
            [-2 * A * B, 1 - 2 * B * B, -2 * B * C],
            [0, 0, 1]
        ];

        const moveMatrix = [
            [1, 0, A],
            [0, 1, B],
            [0, 0, 1]
        ];

        // Function to apply the reflection matrix to a point
        function reflectPoint(x, y) {
            const reflected = [
                moveMatrix[0][-0]*reflectionMatrix[0][0] * x +
reflectionMatrix[0][1] * y + reflectionMatrix[0][2],
                reflectionMatrix[1][0] * x + reflectionMatrix[1][1] * y +
reflectionMatrix[1][2],
                1
            ];
            return { x: reflected[0], y: reflected[1]};
        }

        // Reflect each point of the triangle
        const p1 = reflectPoint(x1 - 2* A * intervalInPixel, y1);
        const p2 = reflectPoint(x2 - 2* A * intervalInPixel, y2);
        const p3 = reflectPoint(x3 - 2* A * intervalInPixel, y3);

        // Draw the triangle
        ctx.beginPath();
        ctx.moveTo(p1.x, p1.y);
```

```
        ctx.lineTo(p2.x, p2.y);
        ctx.lineTo(p3.x, p3.y);
        ctx.closePath();
        currentBaseApex1X = p1.x;
        currentBaseApex1Y = p1.y;
        currentBaseApex2X = p2.x;
        currentBaseApex2Y = p2.y;
        currentTopApexX = p3.x;
        currentTopApexY = p3.y;

        firstVertexX = currentBaseApex1X/intervalInPixel;
        firstVertexY = currentBaseApex1Y/intervalInPixel;
        secondVertexX = currentBaseApex2X/intervalInPixel;
        secondVertexY = currentBaseApex2Y/intervalInPixel;
        heightInput = -heightInput;
        // Style the triangle
        ctx.lineWidth = 2;
        ctx.strokeStyle = '#000000';
        ctx.stroke();


    }



});
```

Висновки

Під час виконання лабораторних робіт, ми навчилися основним принципам побудови геометричних та алгебраїчних фракталів, навчилися імплементувати алгоритми для побудови. Також ми ознайомилися з існуючими колірними моделями, детально розібрали алгоритми переходів в моделі CMYK та HSV. Навчились складати та застосовувати афінні перетворення зза допомогою множення матриць.

Відсоткова участь участі у виконанні кожного завдання:

Лаб. 1 UI/UX: Герман А. – 50% Місяйло О. - 50%

Лаб. 2 Фрактали: Герман А. - 45% Місяйло О. - 55%

Лаб. 3 Кольори: Герман А. – 60% Місяйло О. – 40%

Лаб. 4 Рух: Герман А. – 70% Місяйло О. – 30%

Звіт, презентація : Герман А. – 50% Місяйло О. – 50%

Список використаних інформаційних джерел

Пояснення та алгоритм побудови фракталу коха

https://www.mathros.net.ua/kryva-koha-ta-algorytm-ii-pobudovy.html

Побудова фракталів в JS

https://slicker.me/fractals/fractals.htm

Алгоритм переходу з колірної моделі RGB в CMYK

https://www.rapidtables.com/convert/color/rgb-to-cmyk.html

Алгоритм переходу з колірної моделі RGB в HSV

https://math.stackexchange.com/questions/556341/rgb-to-hsv-color-conversion-algorithm

Афінні перетворення віддзеркалення

https://medium.com/@benjamin.botto/mirroring-drawings-symmetry-with-affine-transformations-591d573667ec