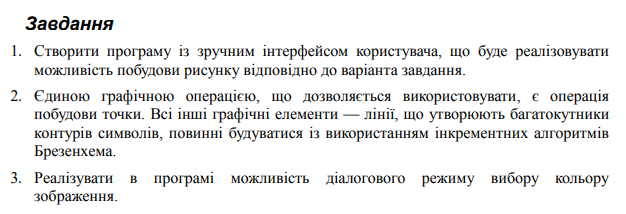
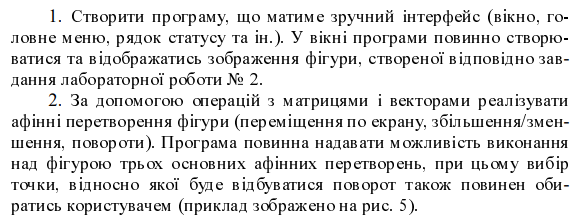
**Лабораторна робота №2**

Інкрементальні алгоритми та афінні перетворення на площині

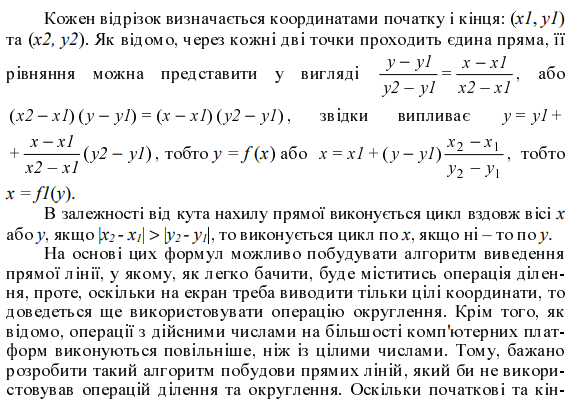
Мета: навчитися будувати рисунки за допомогою інкрементальних алгоритмів та трансформувати їх через основні афінні перетворення.

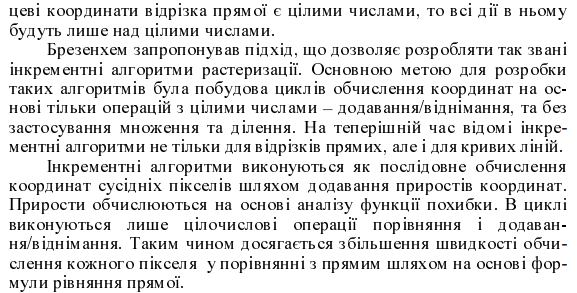






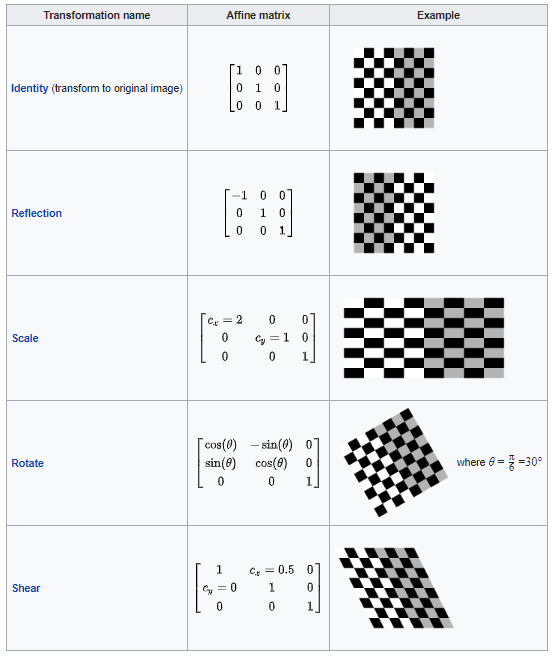
Короткі теоретичні відомості





Афінні перетворення на площині

Афінні перетворення це перетворення взаємно однозначне перетворення під час якого образом будь-якої прямої є пряма. До основних афінних перетворень належать: паралельний перенос, масштабування (а також віддзеркалення), поворот. Усі перетворення можна подати у матричній формі. На практиці саме матрична форма виявляється найбільш зручною.



Рішення

1. Описання

Рішення представляє собою веб додаток у якому користувач може ввести будь-який текст, що відразу буде надруковано на екрані за допомогою алгоритму брезенхама для малювання лінії. Після цього користувач може застосовувати різноманітні афінні трансформації: translate, scale, rotate, shear, reflect. За допомогою цих трансформацій користувач може трансформувати ведений ним текст. Крім цього користувач також може масштабувати та переміщати саму видиму область що відображається на екрані.

Усі трансформації виконуються відносно центру трансформацій (transformation center). Спочатку він знаходиться в точці (0, 0), але користувач може переміщувати його за допомогою панелі керування праворуч, або через drag & drop.

Завдяки афінним перетворенням ми можем дізнаватись нові координаті пікселів, але при цьому можуть з’явитися пропуски між ними. Додаток має опцію «noGaps» яка заповнює їх (через інвертування трансформації).

Під час виконання лабораторної роботи були використані наступні технології:

* + TypeScript, Pug, Scss
  + Canvas 2d
  + Parsel

Рішення підтримує такі можливості:

* + Відображення введеного користувачем тексту (зараз підтримуються лише 3 символи – A, D, 5).
  + Transform, scale, rotate, reflect, shear
  + Додаток працює на будь-якому пристрої, на якому завантажений сучасний браузер.

Посилання на рішення:

* + <https://js-coder.tk/computer-graphics/affine/>
  + <https://js-coder.tk/computer-graphics/bresenham/>

Github:

* + <https://github.com/DimaGashko/computer-graphics/tree/master/affine>
  + <https://github.com/DimaGashko/computer-graphics/tree/master/bresenham>

1. Текст програми

Affine.ts

export function scale(cx: number, cy: number) {

    return [

        [cx, 0, 0],

        [0, cy, 0],

        [0, 0, 1],

    ];

}

export function translate(dx: number, dy: number) {

    return [

        [1, 0, dx],

        [0, 1, dy],

        [0, 0, 1],

    ];

}

export function shear(cx: number, cy: number) {

    return [

        [1, cx, 0],

        [cy, 1, 0],

        [0, 0, 1],

    ];

}

export function reflection(x: boolean, y: boolean) {

    return [

        [x ? -1 : 1, 0, 0],

        [0, y ? -1 : 1, 0],

        [0, 0, 1],

    ];

}

export function rotate(a: number) {

    return [

        [Math.cos(a), -Math.sin(a), 0],

        [Math.sin(a), Math.cos(a), 0],

        [0, 0, 1],

    ];

}

**Math.ts**

export default function matrix3x3MulVec(a: number[][], b: number[]): number[] {

   return [

       a[0][0] \* b[0] + a[0][1] \* b[1] + a[0][2] \* b[2],

       a[1][0] \* b[0] + a[1][1] \* b[1] + a[1][2] \* b[2],

       a[2][0] \* b[0] + a[2][1] \* b[1] + a[2][2] \* b[2]

   ];

}

export default function invert3x3Matrix(matrix: number[][]): number[][] {

   const a = matrix[1][1] \* matrix[2][2] - matrix[1][2] \* matrix[2][1];

   const b = matrix[1][2] \* matrix[2][0] - matrix[1][0] \* matrix[2][2];

   const c = matrix[1][0] \* matrix[2][1] - matrix[1][1] \* matrix[2][0];

   const determinant = matrix[0][0] \* a + matrix[0][1] \* b + matrix[0][2] \* c;

   // Singular; inverse does not exist

   if (determinant === 0) {

       return null;

   }

   return [[

       a,

       (matrix[0][2] \* matrix[2][1] - matrix[0][1] \* matrix[2][2]),

       (matrix[0][1] \* matrix[1][2] - matrix[0][2] \* matrix[1][1]),

   ], [

       b,

       (matrix[0][0] \* matrix[2][2] - matrix[0][2] \* matrix[2][0]),

       (matrix[0][2] \* matrix[1][0] - matrix[0][0] \* matrix[1][2]),

   ], [

       c,

       (matrix[0][1] \* matrix[2][0] - matrix[0][0] \* matrix[2][1]),

       (matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0])

   ]].map(row => row.map(col => col / determinant));

};

**Index.ts**

import { throttle } from 'throttle-debounce';

import Hammer from 'hammerjs';

import \* as dat from 'dat.gui';

import \_font from './font';

import Vector from './scripts/Vector';

import matrixMulMatrix from './scripts/matrix/matrixMulMatrix';

import matrix3x3MulVec from './scripts/matrix/matrix3x3MulVec';

import VirtualCanvas from './scripts/VirtualCanvas';

import { scale, translate, reflection, rotate, shear } from './affine';

import invert3x3Matrix from './scripts/matrix/invert3x3Matrix';

$.root = document.querySelector('.app');

$.canvas = $.root.querySelector('.app\_\_canvas');

$.tCenter = $.root.querySelector('.app\_\_t-center');

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

const gui = new dat.GUI();

const pressedKeys: { [keycode: string]: boolean } = {}

const font: Font = \_font;

const charMap = font2CharMap(font);

const KEYS = {

    left: 37,

    top: 38,

    right: 39,

    bottom: 40,

}

const options = {

    color: '#0f0',

    text: 'A5',

    letterSpacing: 0.8,

    worldZoom: 3,

    translateX: 0,

    translateY: 0,

    zoomX: 1,

    zoomY: 1,

    shearX: 0,

    shearY: 0,

    deg: 0,

    reflectX: false,

    reflectY: false,

    tCenterX: 0,

    tCenterY: 0,

    noGaps: false,

    resetWorldCoords: () => {

        coords = initialCoords.copy();

    }

}

const screenSize = new Vector(0, 0);

const initialCoords = new Vector(50, 50);

let coords = initialCoords.copy();

let tMatrix = [

    [1, 0, 0],

    [0, 1, 0],

    [0, 0, 1],

]

let invertTMatrix: number[][];

const virtualCanvas = new VirtualCanvas();

let charStart = 0;

resize();

updateTMatrix();

start();

initGui();

initEvents();

function start() {

    requestAnimationFrame(function tik() {

        clearCanvas();

        drawFrame();

        requestAnimationFrame(tik);

    });

}

function drawFrame() {

    ctx.save();

    if (options.noGaps) {

        options.worldZoom = Math.max(options.worldZoom, 1);

    }

    initStyles();

    useKeyboard();

    clear();

    drawAllGrids();

    draw();

    moveTCenter();

    ctx.restore();

}

function updateTMatrix() {

    const rawTMatrixes = [

        translate(options.translateX, options.translateY),

        reflection(options.reflectX, options.reflectY),

        shear(options.shearX, options.shearY),

        scale(options.zoomX, options.zoomY),

        rotate(options.deg),

    ];

    tMatrix = rawTMatrixes.reduceRight((prev, cur) => {

        return matrixMulMatrix(prev, cur);

    });

    (<any>window).tm = tMatrix;

}

function draw() {

    drawText(options.text.slice(0, maxLen).toUpperCase());

    if (!options.noGaps) return;

    const z = options.worldZoom;

    const start = new Vector(0, 0);

    const end = screenSize.copy();

    const step = Math.min(z ^ 0, 1);

    const steps = end.sub(start).div(new Vector(step, step));

    for (let i = 0; i < steps.x; i++) {

        for (let j = 0; j < steps.y; j++) {

            const viewX = start.x + i \* step;

            const viewY = start.y + j \* step;

            let { x, y } = toWorld(new Vector(viewX, viewY));

            x -= options.tCenterX;

            y -= options.tCenterY;

            [x, y] = matrix3x3MulVec(invertTMatrix, [x, y, 1]);

            x += options.tCenterX;

            y += options.tCenterY;

            if (!virtualCanvas.check(x, y)) continue;

            ctx.fillRect(viewX, viewY, z, z);

        }

    }

}

function drawText(text: string) {

    text.split('').forEach((realChar) => {

        const char = (charMap.has(realChar))

            ? charMap.get(realChar) : font.unknown;

        drawCharacter(char);

        drawLetterSpace();

    });

}

function drawLetterSpace() {

    charStart += font.size \* options.letterSpacing;

}

function drawCharacter(char: Char) {

    char.forEach((l) => {

        drawLine(l[0], l[1], l[2], l[3]);

    });

}

function drawLine(x1: number, y1: number, x2: number, y2: number) {

    const incX = Math.sign(x2 - x1);

    const incY = Math.sign(y2 - y1);

    const dx = Math.abs(x2 - x1);

    const dy = Math.abs(y2 - y1);

    const d = (dx > dy) ? dx : dy;

    let x = x1;

    let y = y1;

    let xErr = 0;

    let yErr = 0;

    drawPixel(x, y);

    for (let i = 0; i < d; i++) {

        xErr += dx;

        yErr += dy;

        if (xErr > d) {

            xErr -= d;

            x += incX;

        }

        if (yErr > d) {

            yErr -= d;

            y += incY;

        }

        drawPixel(x, y);

    }

}

function drawPixel(x: number, y: number) {

    if (options.noGaps) {

        virtualCanvas.setPixel(x + charStart ^ 0, y, true);

    } else {

        x = x + charStart - options.tCenterX;

        y = y - options.tCenterY;

        [x, y] = matrix3x3MulVec(tMatrix, [x, y, 1]);

        x += options.tCenterX;

        y += options.tCenterY;

        ({ x, y } = toView(new Vector(x, y)));

        const z = options.worldZoom;

        ctx.fillRect(x ^ 0, y ^ 0, z, z);

    }

}

function drawAllGrids() {

    ctx.save();

    ctx.strokeStyle = 'rgba(255,255,255,.2)';

    if (options.worldZoom >= 9) {

        ctx.lineWidth = 1;

        drawGrid(new Vector(1, 1));

    }

    if (options.worldZoom >= 3) {

        ctx.lineWidth = 1;

        drawGrid(new Vector(10, 10));

    }

    if (options.worldZoom >= 0.4) {

        ctx.lineWidth = 2;

        drawGrid(new Vector(50, 50));

    } else if (options.worldZoom >= 0.08) {

        ctx.lineWidth = 1;

        drawGrid(new Vector(250, 250));

    }

    ctx.lineWidth = (options.worldZoom >= 0.4) ? 3 : 1;

    drawGrid(new Vector(1000, 1000));

    ctx.restore();

}

function drawGrid(interval: Vector) {

    const z = new Vector(options.worldZoom, options.worldZoom)

    const step = interval.copy().mul(z);

    const numberOfSteps = screenSize.copy().div(step);

    const g0 = toWorld(new Vector(-1, -1));

    const worldStart = g0.copy().sub(g0.mod(interval));

    const start = toView(worldStart);

    for (let i = 0; i < numberOfSteps.x; i++) {

        const x = Math.round(start.x + step.x \* i);

        ctx.beginPath();

        ctx.moveTo(x, 0);

        ctx.lineTo(x, screenSize.y);

        ctx.stroke();

    }

    for (let i = 0; i < numberOfSteps.y; i++) {

        const y = Math.round(start.y + step.y \* i);

        ctx.beginPath();

        ctx.moveTo(0, y);

        ctx.lineTo(screenSize.x, y);

        ctx.stroke();

    }

}

/\*\*

 \* Convert viewport coords to world coords

 \* @param coords viport coordinates

 \*/

function toWorld(targetCoords: Vector) {

    return targetCoords.copy()

        .sub(screenSize.copy().div(new Vector(2, 2)))

        .div(new Vector(options.worldZoom, options.worldZoom))

        .add(coords);

}

/\*\*

 \* Convert world coords to viewport coords

 \* @param targetCoords world coordinates

 \*/

function toView(targetCoords: Vector) {

    return targetCoords.copy()

        .sub(coords)

        .mul(new Vector(options.worldZoom, options.worldZoom))

        .add(screenSize.copy().div(new Vector(2, 2)));

}

function useKeyboard() {

    const step = 8;

    const type = (document.activeElement === $.tCenter)

        ? 'tCenter' : 'coords';

    const targ = (type === 'coords')

        ? coords : new Vector(options.tCenterX, options.tCenterY);

    if (pressedKeys[KEYS['left']]) targ.x -= step;

    if (pressedKeys[KEYS['top']]) targ.y -= step;

    if (pressedKeys[KEYS['right']]) targ.x += step;

    if (pressedKeys[KEYS['bottom']]) targ.y += step;

    if (type === 'coords') {

        coords = targ;

    } else {

        options.tCenterX = targ.x;

        options.tCenterY = targ.y;

    }

}

function initGui() {

    const setup = gui.addFolder('Setup');

    setup.addColor(options, 'color');

    setup.add(options, 'text');

    setup.add(options, 'letterSpacing', 0.5, 2.5, 0.1);

    setup.add(options, 'noGaps');

    setup.add(options, 'resetWorldCoords');

    const base = gui.addFolder('Main Transformations');

    base.add(options, 'worldZoom', 0.05, 50, 0.05)

    base.add(options, 'zoomX', 0.1, 3, 0.1).onChange(updateTMatrix);

    base.add(options, 'zoomY', 0.1, 3, 0.1).onChange(updateTMatrix);

    base.add(options, 'deg', 0, Math.PI \* 2, 0.01).onChange(updateTMatrix);

    const other = gui.addFolder('Additional Transformations');

    other.add(options, 'translateX', -500, 500, 1).onChange(updateTMatrix);

    other.add(options, 'translateY', -500, 500, 1).onChange(updateTMatrix);

    other.add(options, 'shearX', -3, 3, 0.1).onChange(updateTMatrix);

    other.add(options, 'shearY', -3, 3, 0.1).onChange(updateTMatrix);

    other.add(options, 'reflectX').onChange(updateTMatrix);

    other.add(options, 'reflectY').onChange(updateTMatrix);

    const tCenter = gui.addFolder('Transformation Center');

    tCenter.add(options, 'tCenterX', -500, 500, 0.1);

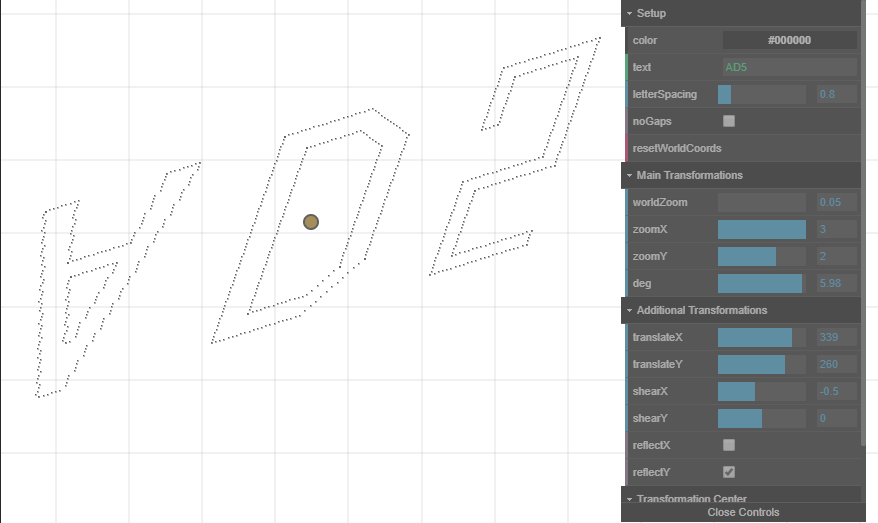
    tCenter.add(options, 'tCenterY', -500, 500, 0.1);

    setup.open();

    base.open();

}

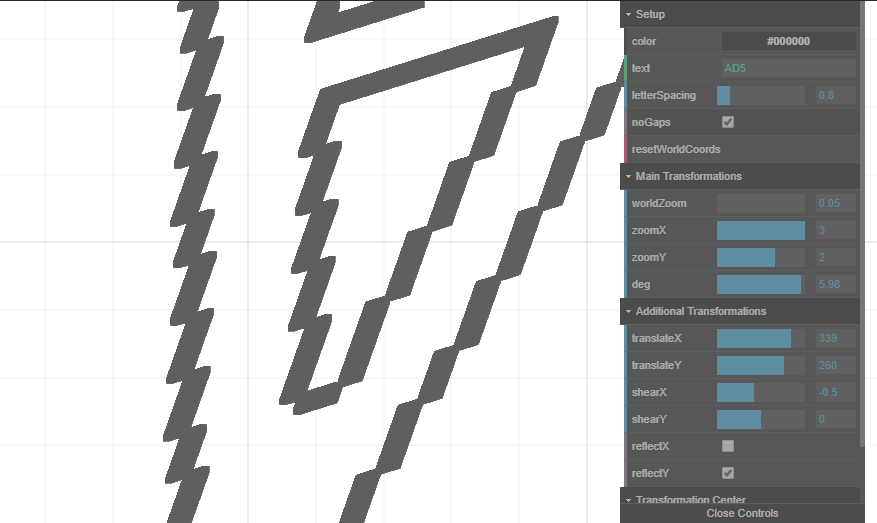
1. Результат виконання програми

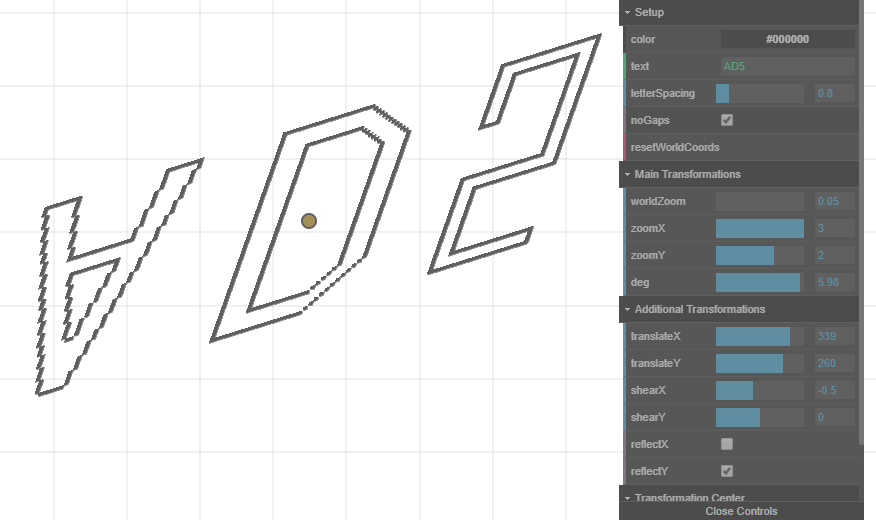


При збільшенні:



З опціїєю «noGaps»:





**Висновок:** на цій лабораторній роботі оволодів навичками побудови ліній за допомогою інкрементальних алгоритмів та перетворення їх через афінні перетворення.