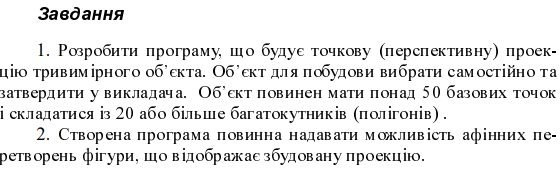
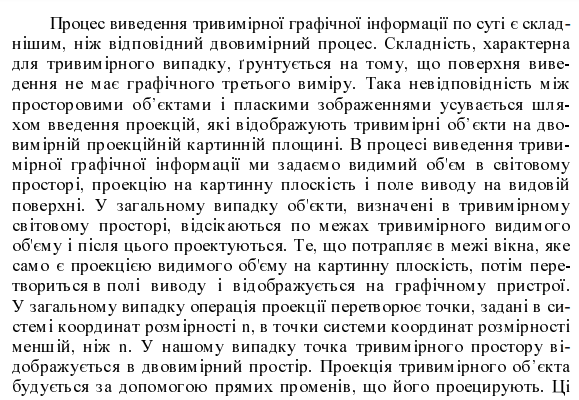
**Лабораторна робота №3**

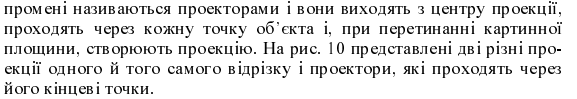
Побудова проекцій тривимірних об’єктів, афінні перетворення у просторі

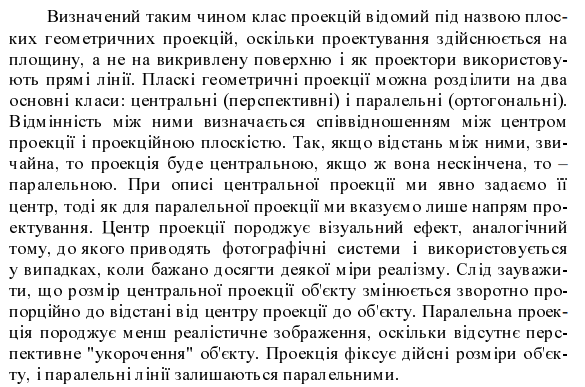
Мета: навчитися будувати тривимірні об’єкти та трансформувати їх за допомогою афінних трансформацій.



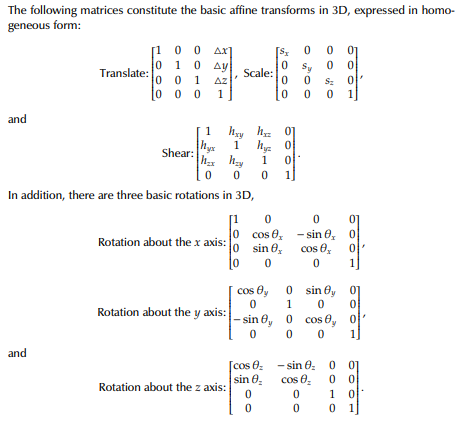
Короткі теоретичні відомості







Афінні перетворення у просторі



Рішення

1. Описання

Під час виконання лабораторної роботи були використані наступні технології:

* TypeScript, Pug, Scss
* WebGL (vanilla WebGL, no other libraries for drawing)
* GLSL language
* Pointer Lock API
* Fullscreen API

Рішення – це веб додаток, що представляє собою сцену, на якій намальований «3d світ» подібних до того, який може бути у якійсь грі.

Користувач може переміщатись по цьому світу за допомогою клавіатури та миші, як у іграх (переміщувати камеру). На сцені намальована досить велика кількість тривимірних об’єктів. Користувач може вибрати будь-який з них та трансформувати його.

Керування:

* W/S/A/D - forward, back, left, right
* Space – up
* Shift – down

Спочатку треба натиснути на сцену, щоб реальний курсор замінився ігровим, після чого за допомогою миші можна повертати камеру, а за допомогою клавіатури переміщати її.

Так як у смартфона немає клавіатури та миші, єдиний спосіб для керування камерою на мобільному телефоні – розділ «camera» на панелі керування праворуч.

Можливості додатку:

Додаток підтримує велику кількість опцій. Через панель керування праворуч можна змінити майже все. Можна змінити «колір» сцени, швидкість обертання об’єктів, швидкість камери, тип керування камерою (з клавіатури або автоматичне переміщення), змінити поле зору, переміщувати камеру та повертати її у 3 вимірах, вибирати кожен з об’єктів та трансформувати його (translate, scale, rotate, reflect, shear по всім координатним осям) тощо.

У Github репозиторії міститься більш детальна інформація про цей та інші додатки. Деякі відео та зображення, що показують основні приклади його роботи.

Посилання на рішення: <https://js-coder.tk/computer-graphics/3d/>

Github: <https://github.com/DimaGashko/computer-graphics/tree/master/3d>

1. Текст програми

Affine.ts

export function makeIdentity() {

    return [

        1, 0, 0, 0,

        0, 1, 0, 0,

        0, 0, 1, 0,

        0, 0, 0, 1,

    ];

}

export function makeTranslation(dx: number, dy: number, dz: number) {

    return [

        1, 0, 0, 0,

        0, 1, 0, 0,

        0, 0, 1, 0,

        dx, dy, dz, 1,

    ];

}

export function makeScale(cx: number, cy: number, cz: number) {

    return [

        cx, 0, 0, 0,

        0, cy, 0, 0,

        0, 0, cz, 0,

        0, 0, 0, 1,

    ];

}

export function makeShear(xy: number, yx: number, xz: number, zx: number, yz: number, zy: number) {

    return [

        1, xy, xz, 0,

        yx, 1, yz, 0,

        zx, zy, 1, 0,

        0, 0, 0, 1,

    ];

}

export function makeRotateX(deg: number) {

    return [

        1, 0, 0, 0,

        0, Math.cos(deg), -Math.sin(deg), 0,

        0, Math.sin(deg), Math.cos(deg), 0,

        0, 0, 0, 1,

    ];

}

export function makeRotateY(deg: number) {

    return [

        Math.cos(deg), 0, Math.sin(deg), 0,

        0, 1, 0, 0,

        -Math.sin(deg), 0, Math.cos(deg), 0,

        0, 0, 0, 1,

    ];

}

export function makeRotateZ(deg: number) {

    return [

        Math.cos(deg), -Math.sin(deg), 0, 0,

        Math.sin(deg), Math.cos(deg), 0, 0,

        0, 0, 1, 0,

        0, 0, 0, 1,

    ];

}

export function makeReflect(cx: boolean, cy: boolean, cz: boolean) {

    return [

        (cx ? -1 : 1), 0, 0, 0,

        0, (cy ? -1 : 1), 0, 0,

        0, 0, (cz ? -1 : 1), 0,

        0, 0, 0, 1,

    ];

}

export function makePerspective(fieldOfView: number, aspect: number, near: number, far: number) {

    var f = Math.tan(Math.PI \* 0.5 - 0.5 \* fieldOfView);

    var rangeInv = 1 / (near - far);

    return [

        f / aspect, 0, 0, 0,

        0, -f, 0, 0,

        0, 0, (near + far) \* rangeInv, -1,

        0, 0, near \* far \* rangeInv \* 2, 0

    ];

}

**Index.ts**

import vShaderSource from './shaders/v.glsl';

import fShaderSource from './shaders/f.glsl';

const gl = $.canvas.getContext('webgl');

const vShader = createShader(gl, gl.VERTEX\_SHADER, vShaderSource);

const fShader = createShader(gl, gl.FRAGMENT\_SHADER, fShaderSource);

const program = createProgram(gl, vShader, fShader);

const loc = {

    aPosition: gl.getAttribLocation(program, 'a\_position'),

    aColor: gl.getAttribLocation(program, 'a\_color'),

    tMatrix: gl.getUniformLocation(program, 'tMatrix'),

    baseColor: gl.getUniformLocation(program, 'baseColor'),

    time: gl.getUniformLocation(program, 'time'),

}

const gui = new dat.GUI();

const fpsCorrection = new FpsCorrection().start();

const worldRadius = 2500;

const options = {

    baseColor: '#f9ff00',

    rotateSpeed: 0.01,

    fieldOfView: Math.PI / 3,

    far: worldRadius \* 2 \* 3,

    near: 1,

    activeGeometry: 'Geometry 1',

    baseGlColor: null,

    DEPTH\_TEST: true,

    toggleFullscreen: () => toggleFullscreen(),

}

const camera = {

    translateX: 0,

    translateY: -300,

    translateZ: 3500,

    rotateX: Math.PI / 9,

    rotateY: 0,

    rotateZ: 0,

    minX: -Math.PI / 2.2,

    maxX: Math.PI / 2.2,

    speed: 10,

    moveMode: 'keyboard',

}

const KEYS = {

    forward: 87,

    back: 83,

    right: 68,

    left: 65,

    up: 32,

    down: 16,

};

const pressedKeys: Map<string, boolean> = new Map();

let screenW = 0;

let screenH = 0;

let canvasW = 0;

let canvasH = 0;

let ratio = 0;

let curMovementX = 0;

let curMovementY = 0;

let time = 0;

let viewMatrix: TMatrix;

const geometries: GlGeometry[] = [];

geometries.push(...[

    [worldRadius, -500, worldRadius],

    [worldRadius, -500, -worldRadius],

    [-worldRadius, -500, worldRadius],

    [-worldRadius, -500, -worldRadius],

]

    .map(coords => coords.map(c => c \* 0.8))

    .map(([x, y, z]) => {

        const glGeometry = new GlGeometry(gl, new FGeometry());

        glGeometry.options.translateX = x;

        glGeometry.options.translateY = y;

        glGeometry.options.translateZ = z;

        return glGeometry;

    })

);

geometries.push(...[

    [worldRadius, -500, worldRadius],

    [worldRadius, -500, -worldRadius],

    [-worldRadius, -500, worldRadius],

    [-worldRadius, -500, -worldRadius],

]

    .map(coords => coords.map(c => c \* 0.8))

    .map(([x, y, z]) => {

        const glGeometry = new GlGeometry(gl, new FGeometry());

        glGeometry.options.translateX = x;

        glGeometry.options.translateY = y;

        glGeometry.options.translateZ = z;

        return glGeometry;

    })

);

geometries.push(...new Array(512).fill(0)

    .map((\_, i, { length }) => {

        const geometry = new FGeometry();

        const glGeometry = new GlGeometry(gl, geometry);

        const options = glGeometry.options;

        const side = Math.cbrt(length) ^ 0;

        const x = i % side;

        const y = (i / (side \*\* 2)) ^ 0;

        const z = ((i / side) ^ 0) % side;

        const size = 300;

        const offset = -(size \* side) / 2;

        options.translateX = offset + x \* size;

        options.translateY = -350 - y \* size;

        options.translateZ = -offset - z \* size;

        return glGeometry;

    }));

geometries.push(...new Array(125).fill(0)

    .map((\_, i, { length }) => {

        const geometry = new CubeGeometry();

        const glGeometry = new GlGeometry(gl, geometry);

        const options = glGeometry.options;

        const side = Math.cbrt(length) ^ 0;

        const x = i % side;

        const y = (i / (side \*\* 2)) ^ 0;

        const z = ((i / side) ^ 0) % side;

        const size = 150;

        options.translateX = 3500 + x \* size;

        options.translateY = -350 - y \* size;

        options.translateZ = -3500 - z \* size;

        return glGeometry;

    }));

(function () {

    const ground = new CubeGeometry();

    const glGeometry = new GlGeometry(gl, ground);

    const { options } = glGeometry;

    const side = worldRadius \* 2 \* 5;

    const size = 50;

    options.translateX = -side;

    options.translateY = 0;

    options.translateZ = -side;

    options.scaleY = 0.25;

    options.scaleX = side / size;

    options.scaleZ = side / size;

    options.autoRotate = false;

    geometries.push(glGeometry);

}());

options.baseGlColor = hexToGlColor(options.baseColor);

initEvents();

resize();

initGui();

start();

function drawFrame() {

    const { DEPTH\_TEST, rotateSpeed, baseGlColor } = options;

    gl.clearColor(0, 0, 0, 0);

      gl.clear(gl.COLOR\_BUFFER\_BIT || gl.DEPTH\_BUFFER\_BIT);

      gl.enable(gl.DEPTH\_TEST);

    gl.useProgram(program);

    update();

geometries.forEach((glGeometry) => {

        const { verticesBuffer, colorsBuffer, geometry, options: gOptions } = glGeometry;

        if (gOptions.autoRotate) {

            gOptions.rotateY += rotateSpeed \* fpsCorrection.val;

        }

        updateTMatrix(glGeometry);

        gl.enableVertexAttribArray(loc.aPosition);

        gl.bindBuffer(gl.ARRAY\_BUFFER, verticesBuffer);

        gl.vertexAttribPointer(loc.aPosition, 3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(loc.aColor);

        gl.bindBuffer(gl.ARRAY\_BUFFER, colorsBuffer);

        gl.vertexAttribPointer(loc.aColor, 3, gl.UNSIGNED\_BYTE, true, 0, 0);

        gl.uniformMatrix4fv(loc.tMatrix, false, geometry.tMatrix.matrix);

        gl.uniform1f(loc.time, time);

        let { r, g, b } = baseGlColor;

        gl.uniform4f(loc.baseColor, r, g, b, 1);

        gl.drawArrays(gl.TRIANGLES, 0, geometry.primitiveCount);

    });

}

function update() {

    time++;

    fpsCorrection.update();

    updateCamera();

    updateViewMatrix();

}

function updateCamera() {

    updateCameraAngle();

    if (camera.moveMode === 'auto') {

        updateCameraCoordsAuto();

    } else {

        updateCameraCoordsByKeyboard();

    }

}

function updateCameraAngle() {

    const { minX, maxX } = camera;

    camera.rotateY += curMovementX / 500;

    camera.rotateX -= curMovementY / 500;

    camera.rotateX = Math.min(Math.max(camera.rotateX, minX), maxX);

    curMovementX = 0;

    curMovementY = 0;

}

function updateCameraCoordsAuto() {

    const { speed, rotateX, rotateY } = camera;

    const d = speed \* fpsCorrection.val;

    camera.translateZ -= d \* Math.cos(rotateY);

    camera.translateY -= d \* Math.sin(rotateX);

    camera.translateX += d \* Math.sin(rotateY);

}

function updateCameraCoordsByKeyboard() {

    const { speed } = camera;

    const normalAngle = Math.PI / 2;

    const d = speed \* fpsCorrection.val;

    let rotateX = 0;

    if (pressedKeys[KEYS.up]) rotateX += normalAngle;

    if (pressedKeys[KEYS.down]) rotateX -= normalAngle;

    camera.translateY -= d \* Math.sin(rotateX);

    let y = 0;

    let z = 0;

    if (pressedKeys[KEYS.forward]) z++;

    if (pressedKeys[KEYS.back]) z--;

    if (pressedKeys[KEYS.left]) y--;

    if (pressedKeys[KEYS.right]) y++;

    if (y === 0 && z === 0) return;

    const rotateY = Math.atan2(y, z);

    const dx = d \* Math.sin(rotateY);

    const dz = d \* Math.cos(rotateY);

    camera.translateX += dx \* Math.cos(-camera.rotateY) - dz \* Math.sin(-camera.rotateY);

    camera.translateZ -= dx \* Math.sin(-camera.rotateY) + dz \* Math.cos(-camera.rotateY);

}

function updateViewMatrix() {

    const { fieldOfView, near, far } = options;

    const { translateX, translateY, translateZ, rotateX, rotateY, rotateZ } = camera;

    viewMatrix = new TMatrix()

        .translate(translateX, translateY, translateZ)

        .rotateY(rotateY)

        .rotateX(rotateX)

        .rotateZ(rotateZ)

        .inverse();

    viewMatrix.setTMatrix(matMulMat4(

        viewMatrix.matrix,

        makePerspective(fieldOfView, ratio, near, far)

    ));

}

function updateTMatrix(glGeometry: GlGeometry) {

    const { geometry, options } = glGeometry;

    const tMatrix = geometry.tMatrix;

    const {

        reflectX, reflectY, reflectZ,

        rotateX, rotateY, rotateZ,

        scaleX, scaleY, scaleZ,

        shearXY, shearXZ, shearYX, shearYZ, shearZX, shearZY,

        translateX, translateY, translateZ,

    } = options;

    const { centerX, centerY, centerZ } = geometry;

    tMatrix.reset(viewMatrix.matrix)

        .translate(translateX, translateY, translateZ)

        .translate(centerX \* scaleX, centerY \* scaleY, centerZ \* scaleZ)

        .rotateY(rotateY)

        .rotateX(rotateX)

        .rotateZ(rotateZ)

        .reflect(reflectX, reflectY, reflectZ)

        .scale(scaleX, scaleY, scaleZ)

        .translate(-centerX, -centerY, -centerZ)

        .shear(shearXY, shearYX, shearXZ, shearZX, shearYZ, shearZY)

}

function updateMetrics() {

    screenW = $.canvas.clientWidth;

    screenH = $.canvas.clientHeight;

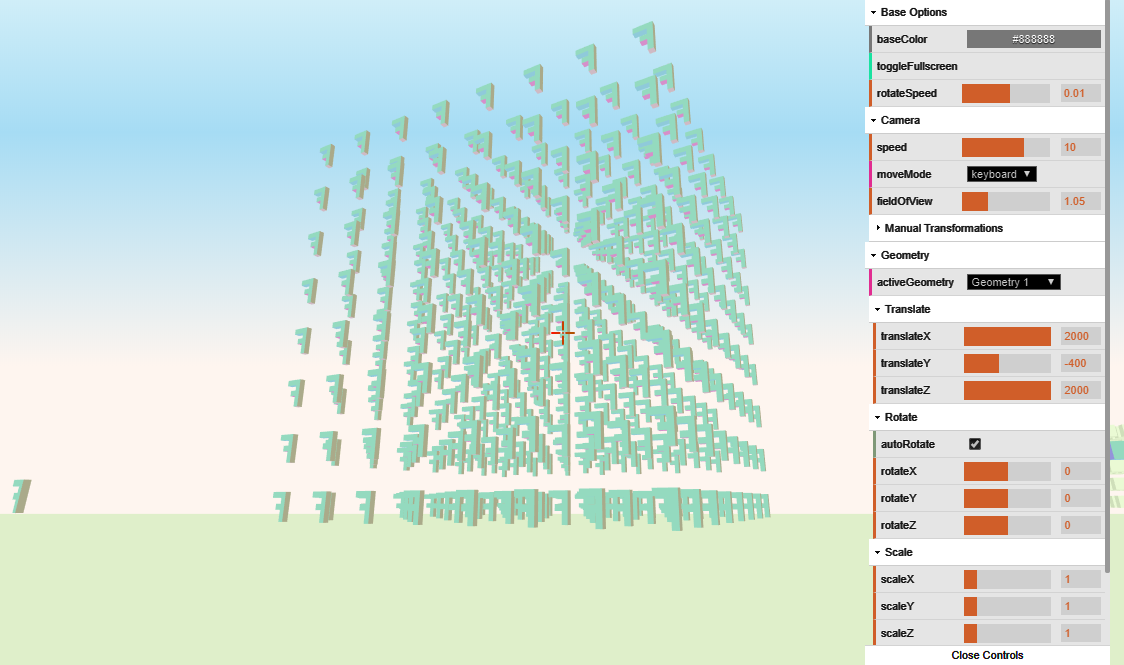
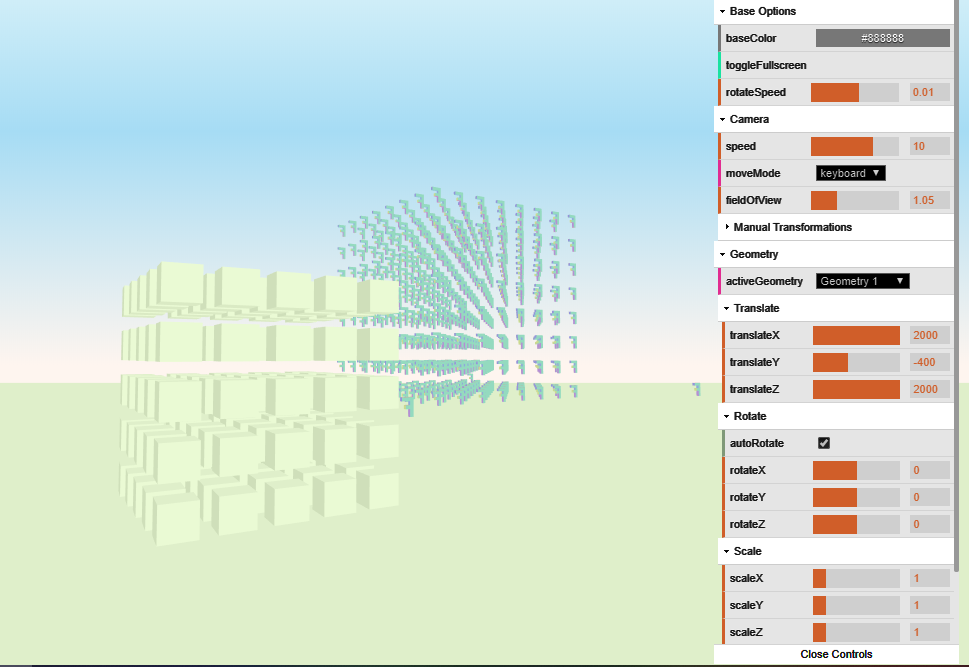
    canvasW = Math.floor(screenW \* window.devicePixelRatio);

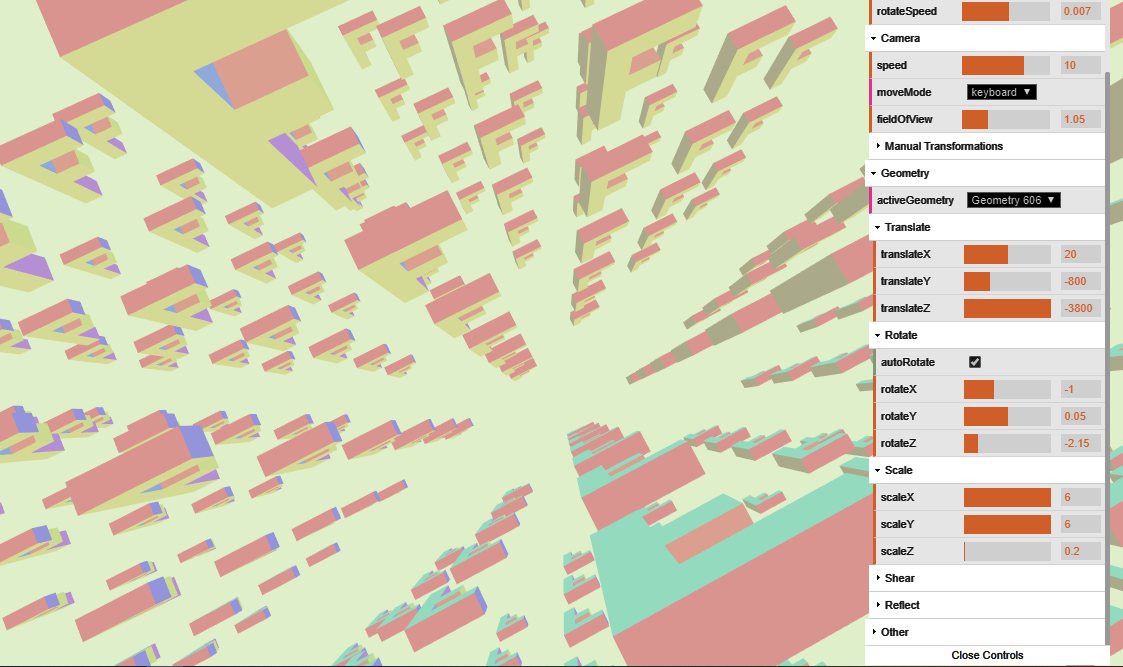
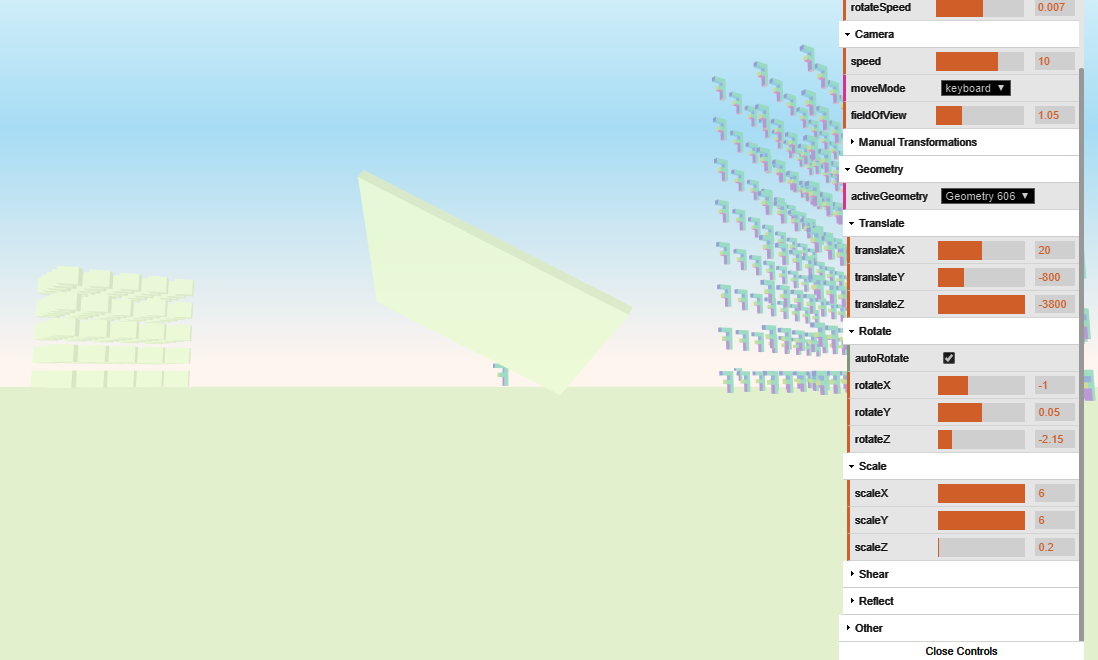
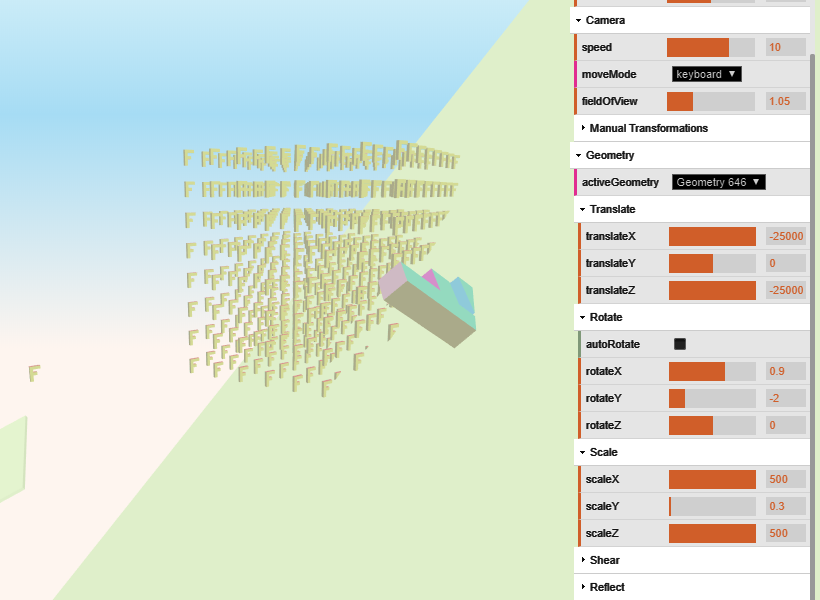
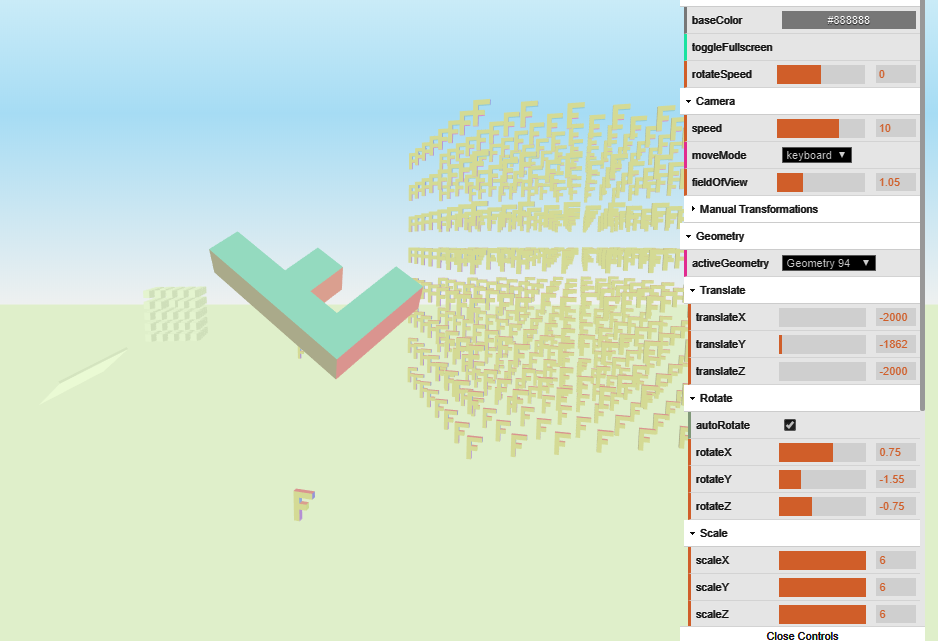
    canvasH = Math.floor(screenH \* window.devicePixelRatio);

    ratio = canvasW / canvasH;

}

1. Результат виконання програми

**Висновок:** на цій лабораторній роботі оволодів навичками тривимірних об’єктів та трансформувати їх через афінні перетворення.