Министерство образования и науки Российской Федерации

Курский государственный университет

Кафедра ПОиАИС

**Аффинные преобразования в пространстве**

Отчёт о выполнении лабораторной работы № 4

По дисциплине «Компьютерная графика»

Выполнил студент гр. 313.1 М.С. Козявин

Проверил асс. Н.Д. Писарева

Курск – 2023

**Цель работы:** Изучение математических методов аффинных преобразований в пространстве и практическое освоение приёмов программной реализации аффинных преобразований.

**Задание:**

Создать программу, реализующую следующие преобразования каркасной модели трёхмерного тела (многогранника) в пространстве в зависимости от индивидуального варианта:

**Вариант 4**

1. Перенос перпендикулярно конкретной грани многогранника.
2. Масштабирование перпендикулярно конкретной плоскости.
3. Отражение относительно заданной грани.
4. Позволять изменять масштаб вывода графика.
5. Поворот вокруг произвольной прямой в пространстве.

**Описание основных алгоритмов:**

Графики строятся с помощью функций drawDec и drawPol c передачей параметров графиков.

**Параметры графика**

gridFreq – количество делений графика от центра до края;

graphVal – максимальное число, отображаемое на графике;

accurancy – количество знаков после запятой;

step – шаг при построении графиков;

dots, lines – массивы хранящие точки и линии фигуры;

projectionAngle – проекционный вектор

xzAngle – угол между осями OX и OY

MODES – множество хранящее режимы работы в соответствиями с проделываемыми преобразованиями

transformMatrix – текущая матрица преобразования

moveVector – вектор на который необходимо передвинуть фигуру

scaleVector - вектор вдоль которого необходимо масштабировать фигуру

rotateVector – вектор вокруг которого необходимо повернуть фигуру

**Функции**

drawAxis – общая функция для построения горизонтальной и вертикальной осей;

drawLabels – общая функция для вывода подписей к осям и чисел на осях;

drawDec – функция, изображающая прямоугольную сетку с шагом gridSize = size/(n\*2) и заданную функцию f с шагом step от –maxVal до maxVal;

drawPol – функция, изображающая в цикле сетку из окружностей с шагом gridSize и заданную функцию f с шагом step от -2pi до 2pi;

doTransform – функция изменяющая матрицу преобразования в зависимости от введённых данных и режима работы.

**Код программы:**

var graphicsSize = 450;

var gridFreq = 5;

var graphVal = 5;

var graphicThickness = 1;

var accurancy = 0;

var textSize = 18;

var color = "red";

var dotSize = 5;

var vectorColor = "DodgerBlue";

var offset = 1;

var gridSize = graphicsSize/(gridFreq\*2);

var dots = [];

var lines = [];

var projectionAngle = 0;

var projectionAngleDeg = 0;

var projectionAngleTransitionTo = 0;

var projectionVector = [0, 0, 1, 1];

var distance = 0;

var distanceTransitionTo = 0;

var xzAngle;

var xzAngleDeg = 90;

var xzAngleTransitionTo = 90;

var animationTimer = null;

var transformMatrix = parallelMatrix();

const AXIS = ["x", "y", "z"];

const MODES = {REGULAR: 1, MOVE: 2, SCALE: 3, REFLECT: 4, ROTATE: 5};

var mode = MODES.REGULAR;

var moveVector = [];

var moveCoef = 1;

var scaleVector = [];

var reflectVector = [];

var rotateVector = [];

var rotateAngle = 0;

var rotateAngleDeg = 0;

var rotateAngleTransitionTo = 0;

function drawAxis(ctx, x, y, size) {

ctx.stroke();

ctx.lineWidth = 3;

ctx.beginPath();

ctx.moveTo(x, y + size/2);

ctx.lineTo(x + size, y + size/2);

ctx.moveTo(x + size, y + size/2);

ctx.lineTo(x + size-10, y + size/2+5);

ctx.moveTo(x + size, y + size/2);

ctx.lineTo(x + size-10, y + size/2-5);

ctx.moveTo(x+size/2, y);

ctx.lineTo(x+size/2, y+size);

ctx.moveTo(x+size/2, y);

ctx.lineTo(x+size/2-5, y+10);

ctx.moveTo(x+size/2, y);

ctx.lineTo(x+size/2+5, y+10);

ctx.stroke();

}

function drawLabels(ctx, x, y, size, maxVal, gridStep, gridSize, n, xid, yid) {

for (let i = 1; i < 2\*n; i++) {

ctx.textAlign = "center";

ctx.fillText((-maxVal+(i\*gridStep)).toFixed(accurancy), x+i\*gridSize-10, y+size/2+15);

ctx.textAlign = "right";

if (maxVal-i\*gridStep != 0) ctx.fillText((maxVal-i\*gridStep).toFixed(accurancy), x+size/2-5, y+i\*gridSize+15);

}

ctx.strokeRect(x, y, size, size);

ctx.textAlign = "center";

ctx.fillText(AXIS[xid], x+size-10, y+size/2+15);

ctx.textAlign = "right";

ctx.fillText(AXIS[yid], x+size/2-5, y+15);

}

function drawSide(ctx, x, y, size, n, maxVal, xid=0, yid=1, matrix=frontalMatrix()) {

ctx.strokeStyle = "black";

ctx.beginPath();

ctx.lineWidth = 1;

let gridStep = maxVal/n;

for (let i = -n+1; i < n; i++) {

ctx.moveTo(x, y + size/2 + gridSize\*i);

ctx.lineTo(x + size, y + size/2 + gridSize\*i);

ctx.moveTo(x + size/2 + gridSize\*i, y);

ctx.lineTo(x + size/2 + gridSize\*i, y + size);

}

ctx.stroke();

drawAxis(ctx, x, y, size)

drawLabels(ctx, x, y, size, maxVal, gridStep, gridSize, n, xid, yid)

ctx.beginPath();

let screenCoord1;

let screenCoord2;

ctx.strokeStyle = color;

switch (mode) {

case MODES.REGULAR:

break;

case MODES.MOVE:

matrix = matrixMul(moveMatrix(moveVector[0]\*moveCoef, moveVector[1]\*moveCoef, moveVector[2]\*moveCoef), matrix);

break;

case MODES.SCALE:

matrix = matrixMul(scaleMatrix(scaleVector[0], scaleVector[1], scaleVector[2]), matrix);

break;

case MODES.REFLECT:

matrix = matrixMul(scaleMatrix(reflectVector[0], reflectVector[1], reflectVector[2]), matrix);

break;

case MODES.ROTATE:

angleXV = Math.acos(rotateVector[0]); // 1 0 0

angleYV = Math.acos(rotateVector[1]); // 0 1 0

if (angleYV == Math.PI) {

angleYV = 0;

}

if (angleXV == Math.PI) {

angleXV = 0;

}

matrix = matrixMul(moveMatrix(moveVector[0], moveVector[1], moveVector[2]), matrixMul(rotateZMatrix(-angleYV),

matrixMul(rotateYMatrix(-angleXV), matrixMul(rotateYMatrix(rotateAngle), matrixMul(rotateZMatrix(angleYV),

matrixMul(rotateYMatrix(angleXV), matrixMul(moveMatrix(-moveVector[0], -moveVector[1], -moveVector[2]), matrix)))))));

break;

}

let newDots = matrixMul(dots, matrix);

for (let i = 0; i < lines.length; i++) {

screenCoord1 = toScreen(newDots[lines[i][0]][xid], newDots[lines[i][0]][yid]);

screenCoord2 = toScreen(newDots[lines[i][1]][xid], newDots[lines[i][1]][yid]);

ctx.moveTo(x+screenCoord1[0], y+screenCoord1[1]);

ctx.lineTo(x+screenCoord2[0], y+screenCoord2[1]);

}

ctx.stroke();

ctx.beginPath();

ctx.strokeStyle = vectorColor;

let vec = matrixMul([projectionVector], matrix);

screenCoord1 = toScreen(0, 0);

screenCoord2 = toScreen(vec[0][xid], vec[0][yid]);

ctx.moveTo(x+screenCoord1[0], y+screenCoord1[1]);

ctx.lineTo(x+screenCoord2[0], y+screenCoord2[1]);

ctx.stroke();

}

function drawAxis3d(ctx, x, y, size) {

ctx.stroke();

ctx.lineWidth = 3;

ctx.beginPath();

ctx.moveTo(x, y + size/2);

ctx.lineTo(x + size, y + size/2);

ctx.moveTo(x + size, y + size/2);

ctx.lineTo(x + size-10, y + size/2+5)

ctx.moveTo(x + size, y + size/2);

ctx.lineTo(x + size-10, y + size/2-5)

ctx.moveTo(x+size/2, y);

ctx.lineTo(x+size/2, y+size);

ctx.moveTo(x+size/2, y);

ctx.lineTo(x+size/2-5, y+10)

ctx.moveTo(x+size/2, y);

ctx.lineTo(x+size/2+5, y+10)

if (xzAngle != 90) {

lx = Math.cos(xzAngle)\*size;

ly = Math.sin(xzAngle)\*size;

ctx.moveTo(x+size/2-lx, y+size/2-ly);

ctx.lineTo(x+size/2+lx, y+size/2+ly);

}

ctx.stroke();

}

function drawLabels3d(ctx, x, y, size, maxVal, gridStep, gridSize, n, xid, yid, zid) {

ctx.fillStyle = "black";

ctx.beginPath();

for (let i = 1; i < 2\*n; i++) {

ctx.textAlign = "center";

ctx.fillText((-maxVal+(i\*gridStep)).toFixed(accurancy), x+i\*gridSize-10, y+size/2+15);

ctx.textAlign = "right";

if (maxVal-i\*gridStep != 0) ctx.fillText((maxVal-i\*gridStep).toFixed(accurancy), x+size/2-5, y+i\*gridSize+15);

}

ctx.stroke();

ctx.fillStyle = vectorColor;

ctx.beginPath();

for (let i = 0; i < 2\*n+1; i++) {

if (xzAngleDeg != 90 && projectionAngle != 0) {

let zi = matrixMul([[0, 0, i-maxVal, 1]], parallelMatrix(0));

if (maxVal-i\*gridStep != 0) ctx.fillText(i-maxVal, x+zi[0][0]\*gridSize+size/2, y-zi[0][1]\*gridSize+size/2);

}

}

ctx.stroke();

ctx.fillStyle = "black";

ctx.beginPath();

ctx.strokeRect(x, y, size, size);

ctx.textAlign = "center";

ctx.fillText(AXIS[xid], x+size-10, y+size/2+15);

ctx.textAlign = "right";

ctx.fillText(AXIS[yid], x+size/2-5, y+15);

ctx.stroke();

}

function drawParallel(ctx, x, y, size, n, maxVal) {

ctx.strokeStyle = "black";

ctx.beginPath();

ctx.lineWidth = 1;

let gridStep = maxVal/n;

for (let i = -n+1; i < n; i++) {

ctx.moveTo(x, y + size/2 + gridSize\*i);

ctx.lineTo(x + size, y + size/2 + gridSize\*i);

ctx.moveTo(x + size/2 + gridSize\*i, y);

ctx.lineTo(x + size/2 + gridSize\*i, y + size);

}

ctx.stroke();

drawAxis3d(ctx, x, y, size);

drawLabels3d(ctx, x, y, size, maxVal, gridStep, gridSize, n, 0, 1, 2);

ctx.beginPath();

let screenCoord1;

let screenCoord2;

ctx.strokeStyle = color;

let newDots = matrixMul(dots, transformMatrix);

for (let i = 0; i < lines.length; i++) {

screenCoord1 = toScreen(newDots[lines[i][0]][0], newDots[lines[i][0]][1]);

screenCoord2 = toScreen(newDots[lines[i][1]][0], newDots[lines[i][1]][1]);

ctx.moveTo(x+screenCoord1[0], y+screenCoord1[1]);

ctx.lineTo(x+screenCoord2[0], y+screenCoord2[1]);

}

ctx.stroke();

ctx.beginPath();

ctx.fillStyle = "Lime";

for (let i = 0; i < dots.length; i++) {

let screenCoord = toScreen(newDots[i][0], newDots[i][1]);

ctx.textAlign = "center";

ctx.fillText(i, x+screenCoord[0], y+screenCoord[1]);

}

ctx.stroke();

ctx.strokeStyle = vectorColor;

}

var canvases = document.getElementsByTagName("canvas");

var contexts = [];

for (let canv of canvases) {

contexts.push(canv.getContext('2d'));

contexts[contexts.length-1].translate(0.5, 0.5);

contexts[contexts.length-1].font = textSize + "px sans-serif";

contexts[contexts.length-1].strokeStyle = "black";

contexts[contexts.length-1].lineCap = "square";

canv.width = offset\*2+graphicsSize;

canv.height = graphicsSize + offset\*2;

};

function redraw() {

for (let i = 0; i < contexts.length; i++) {

contexts[i].clearRect(0, 0, canvases[i].width, canvases[i].height);

}

drawSide(contexts[0], offset, offset, graphicsSize, gridFreq, graphVal, 0, 2, upMatrix());

drawParallel(contexts[1], offset, offset, graphicsSize, gridFreq, graphVal, 0, 1);

drawSide(contexts[2], offset, offset, graphicsSize, gridFreq, graphVal, 0, 1, frontalMatrix());

drawSide(contexts[3], offset, offset, graphicsSize, gridFreq, graphVal, 2, 1, rightMatrix());

};

function toGraph(screenX, screenY) {

let graphX = (screenX - graphicsSize/2) / gridSize;

let graphY = -(screenY - graphicsSize/2) / gridSize;

return [graphX, graphY];

}

function toScreen(graphX, graphY) {

let screenX = graphX \* gridSize + graphicsSize/2;

let screenY = -(graphY \* gridSize) + graphicsSize/2;

return [screenX, screenY, 1, 1];

}

let fileInputButton = document.getElementById("enterFile");

fileInputButton.addEventListener("click", getFile);

function getFile() {

if (document.getElementById("fileInput").files.length < 1) {

return;

}

let file = document.getElementById("fileInput").files[0];

let reader = new FileReader();

reader.readAsText(file);

reader.onload = function() {

dots = [];

lines = [];

parse(reader.result);

redraw();

};

reader.onerror = function() {

console.log(reader.error);

};

};

function isLineInLines(lines, line){

let str = JSON.stringify(line);

let contains = lines.some(function(el){

return JSON.stringify(el) === str;

});

if (contains) return true

str = JSON.stringify(line.reverse());

contains = lines.some(function(el){

return JSON.stringify(el) === str;

});

return contains;

}

function parse(text) {

let d = text.split(/\r\n| |\n|\r/);

let l = parseInt(d[0]);

for (let i = 0; i < l; i++) {

dots.push([]);

dots[i][0] = parseInt(d[i\*3+1]);

dots[i][1] = parseInt(d[i\*3+2]);

dots[i][2] = parseInt(d[i\*3+3]);

dots[i][3] = parseInt(1);

}

let j = 1;

for (let i = 0; i < l; i++) {

while (d[l\*3+1+j] != ";") {

if (isLineInLines(lines, [parseInt(d[l\*3+1+j]), i])) {

j++;

continue;

}

lines.push([i, parseInt(d[l\*3+1+j])]);

j++;

};

j += 2;

}

};

let projectionAngleInput = document.getElementById("projectionAngleInput");

projectionAngleInput.value = 0;

projectionAngleInput.addEventListener("change", getProjectionAngle);

function getProjectionAngle() {

projectionAngleTransitionTo = projectionAngleInput.value;

}

let distanceEnterButton = document.getElementById("enterDistance");

document.getElementById("distanceInput").value = 0;

distanceEnterButton.addEventListener("click", getDistance);

function getDistance() {

distanceTransitionTo = document.getElementById("distanceInput").value;

}

let timeEnterButton = document.getElementById("enterTime");

document.getElementById("timeInput").value = 5;

timeEnterButton.addEventListener("click", getTime);

function getTime() {

let timeInput = document.getElementById("timeInput");

clearInterval(animationTimer);

console.log(timeInput.value);

animationTimer = setInterval(updateTransition, timeInput.value);

}

let xzAngleInput = document.getElementById("XZAngleInput");

xzAngleInput.value = 90;

xzAngleInput.addEventListener("change", getXZAngle);

function getXZAngle() {

xzAngleTransitionTo = xzAngleInput.value;

}

let moveButton = document.getElementById("move");

moveButton.addEventListener("click", ()=>{

let dot1 = parseInt(document.getElementById("move1").value);

let dot2 = parseInt(document.getElementById("move2").value);

let dot3 = parseInt(document.getElementById("move3").value);

let factor = parseInt(document.getElementById("move-size").value);

if (dot1 >= 0 && dot2 >= 0 && dot3 >= 0 && dot1 < dots.length && dot2 < dots.length && dot3 < dots.length) {

let a = [dots[dot2][0]-dots[dot1][0], dots[dot2][1]-dots[dot1][1], dots[dot2][2]-dots[dot1][2]];

let b = [dots[dot3][0]-dots[dot1][0], dots[dot3][1]-dots[dot1][1], dots[dot3][2]-dots[dot1][2]];

let n = [a[1]\*b[2]-a[2]\*b[1], a[2]\*b[0]-a[0]\*b[2], a[0]\*b[1]-a[1]\*b[0]];

let l = Math.sqrt(n[0]\*n[0] + n[1]\*n[1] + n[2]\*n[2]);

moveVector = [n[0]/l, n[1]/l, n[2]/l];

moveCoef = factor;

mode = MODES.MOVE;

}

});

let scaleButton = document.getElementById("scale");

scaleButton.addEventListener("click", ()=>{

let dot1x = parseInt(document.getElementById("scale1x").value);

let dot1y = parseInt(document.getElementById("scale1y").value);

let dot1z = parseInt(document.getElementById("scale1z").value);

let dot2x = parseInt(document.getElementById("scale2x").value);

let dot2y = parseInt(document.getElementById("scale2y").value);

let dot2z = parseInt(document.getElementById("scale2z").value);

let dot3x = parseInt(document.getElementById("scale3x").value);

let dot3y = parseInt(document.getElementById("scale3y").value);

let dot3z = parseInt(document.getElementById("scale3z").value);

let factor = parseFloat(document.getElementById("coeff-input").value);

let a = [dot2x-dot1x, dot2y-dot1y, dot2z-dot1z];

let b = [dot3x-dot1x, dot3y-dot1y, dot3z-dot1z];

let n = [a[1]\*b[2]-a[2]\*b[1], a[2]\*b[0]-a[0]\*b[2], a[0]\*b[1]-a[1]\*b[0]];

let l = Math.sqrt(n[0]\*n[0] + n[1]\*n[1] + n[2]\*n[2]);

scaleVector = [n[0]\*factor/l || 1, n[1]\*factor/l || 1, n[2]\*factor/l || 1];

mode = MODES.SCALE;

});

let reflectButton = document.getElementById("reflect");

reflectButton.addEventListener("click", ()=>{

let dot1 = parseInt(document.getElementById("reflect1").value);

let dot2 = parseInt(document.getElementById("reflect2").value);

let dot3 = parseInt(document.getElementById("reflect3").value);

if (dot1 >= 0 && dot2 >= 0 && dot3 >= 0 && dot1 < dots.length && dot2 < dots.length && dot3 < dots.length) {

let a = [dots[dot2][0]-dots[dot1][0], dots[dot2][1]-dots[dot1][1], dots[dot2][2]-dots[dot1][2]];

let b = [dots[dot3][0]-dots[dot1][0], dots[dot3][1]-dots[dot1][1], dots[dot3][2]-dots[dot1][2]];

let n = [a[1]\*b[2]-a[2]\*b[1], a[2]\*b[0]-a[0]\*b[2], a[0]\*b[1]-a[1]\*b[0]];

let l = Math.sqrt(n[0]\*n[0] + n[1]\*n[1] + n[2]\*n[2]);

reflectVector = [n[0]\*(-1)/l || 1, n[1]\*(-1)/l || 1, n[2]\*(-1)/l || 1];

mode = MODES.REFLECT;

}

});

let rotateButton = document.getElementById("rotate");

rotateButton.addEventListener("click", ()=>{

let dot1x = parseInt(document.getElementById("rotate1x").value);

let dot1y = parseInt(document.getElementById("rotate1y").value);

let dot1z = parseInt(document.getElementById("rotate1z").value);

let dot2x = parseInt(document.getElementById("rotate2x").value);

let dot2y = parseInt(document.getElementById("rotate2y").value);

let dot2z = parseInt(document.getElementById("rotate2z").value);

let angle = parseInt(document.getElementById("angle-input").value);

let n = [dot2x-dot1x, dot2y-dot1y, dot2z-dot1z];

let l = Math.sqrt(n[0]\*n[0] + n[1]\*n[1] + n[2]\*n[2]);

moveVector = [-dot1x, -dot1y, -dot1z];

rotateVector = [n[0]/l, n[1]/l, n[2]/l, 1];

rotateAngleTransitionTo = angle;

mode = MODES.ROTATE;

});

let resetButton = document.getElementById("reset");

resetButton.addEventListener("click", ()=>{

mode = MODES.REGULAR;

});

function doTransform() {

switch (mode) {

case MODES.REGULAR:

transformMatrix = parallelMatrix();

break;

case MODES.MOVE:

transformMatrix = matrixMul(moveMatrix(moveVector[0]\*moveCoef, moveVector[1]\*moveCoef, moveVector[2]\*moveCoef), parallelMatrix());

break;

case MODES.SCALE:

transformMatrix = matrixMul(scaleMatrix(scaleVector[0], scaleVector[1], scaleVector[2]), parallelMatrix());

break;

case MODES.REFLECT:

transformMatrix = matrixMul(scaleMatrix(reflectVector[0], reflectVector[1], reflectVector[2]), parallelMatrix());

break;

case MODES.ROTATE:

angleXV = Math.acos(rotateVector[0]); // 1 0 0

angleYV = Math.acos(rotateVector[1]); // 0 1 0

if (angleYV == Math.PI) {

angleYV = 0;

}

if (angleXV == Math.PI) {

angleXV = 0;

}

transformMatrix = matrixMul(moveMatrix(moveVector[0], moveVector[1], moveVector[2]), matrixMul(rotateZMatrix(-angleYV),

matrixMul(rotateYMatrix(-angleXV), matrixMul(rotateYMatrix(rotateAngle), matrixMul(rotateZMatrix(angleYV),

matrixMul(rotateYMatrix(angleXV), matrixMul(moveMatrix(-moveVector[0], -moveVector[1], -moveVector[2]), parallelMatrix())))))));

break;

}

}

function updateTransition() {

if (rotateAngleDeg != rotateAngleTransitionTo) {

if (rotateAngleDeg > rotateAngleTransitionTo) rotateAngleDeg -= 1;

if (rotateAngleDeg < rotateAngleTransitionTo) rotateAngleDeg += 1;

rotateAngle = degToRad(rotateAngleDeg);

}

if (projectionAngleDeg != projectionAngleTransitionTo) {

if (projectionAngleDeg > projectionAngleTransitionTo) projectionAngleDeg -= 1;

if (projectionAngleDeg < projectionAngleTransitionTo) projectionAngleDeg += 1;

projectionAngle = degToRad(projectionAngleDeg);

projectionVector = [Math.cos(-xzAngle), Math.sin(-xzAngle), 1/Math.tan(projectionAngle), 1];

}

if (Math.abs(distance-distanceTransitionTo) > 0.01) {

if (distance > distanceTransitionTo) distance -= 0.1;

else if (distance < distanceTransitionTo) distance += 0.1;

}

if (xzAngleDeg != xzAngleTransitionTo) {

if (xzAngleDeg > xzAngleTransitionTo) xzAngleDeg -= 1;

if (xzAngleDeg < xzAngleTransitionTo) xzAngleDeg += 1;

xzAngle = degToRad(xzAngleDeg);

projectionVector = [Math.cos(-xzAngle), Math.sin(-xzAngle), 1/Math.tan(projectionAngle), 1];

}

doTransform();

redraw();

}

function matrixMul(a, b) {

let r = [];

for (let i = 0; i < a.length; i++) {

r.push([]);

for (let j = 0; j < b[0].length; j++) {

let s = 0;

for (let x = 0; x < b.length; x++) {

s += a[i][x] \* b[x][j];

}

r[i][j] = s;

}

}

return r;

}

function degToRad(deg) {

return deg \* (Math.PI/180);

}

function frontalMatrix() {

return [[1, 0, 0, 0],

[0, 1, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 1]];

}

function upMatrix() {

return [[1, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]];

}

function rightMatrix() {

return [[0, 0, 0, 0],

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]];

}

function rotateYMatrix(angle) {

return [[Math.cos(angle), 0, Math.sin(angle), 0],

[0, 1, 0, 0],

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

[0, 0, 0, 1]];

}

function rotateXMatrix(angle) {

return [[1, 0, 0, 0],

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

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

[0, 0, 0, 1]];

}

function rotateZMatrix(angle) {

return [[Math.cos(angle), -Math.sin(angle), 0, 0],

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

[0, 0, 1, 0],

[0, 0, 0, 1]];

}

function scaleMatrix(a, b, c) {

return[[a, 0, 0, 0],

[0, b, 0, 0],

[0, 0, c, 0],

[0, 0, 0, 1]];

}

function moveMatrix(a, b, c) {

return[[1, 0, 0, 0],

[0, 1, 0, 0],

[0, 0, 1, 0],

[a, b, c, 1]];

}

function parallelMatrix(dist=distance) {

return [[1, 0, 0, 0],

[0, 1, 0, 0],

[-(projectionVector[0]/projectionVector[2]), -(projectionVector[1]/projectionVector[2]), 0, 0],

[-dist\*projectionVector[0]/projectionVector[2], -dist\*projectionVector[1]/projectionVector[2], 0, 1]];

}

function regularMatrix() {

return [[1, 0, 0, 0],

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]];

}

getXZAngle();

getTime();

getProjectionAngle();

getDistance();

redraw();

**Результаты работы**

На рисунках 1-2 представлен результат выполнения программы, построение графика с разными значениями параметров, разным масштабом, а также с отключением сетки, осей и подписей.

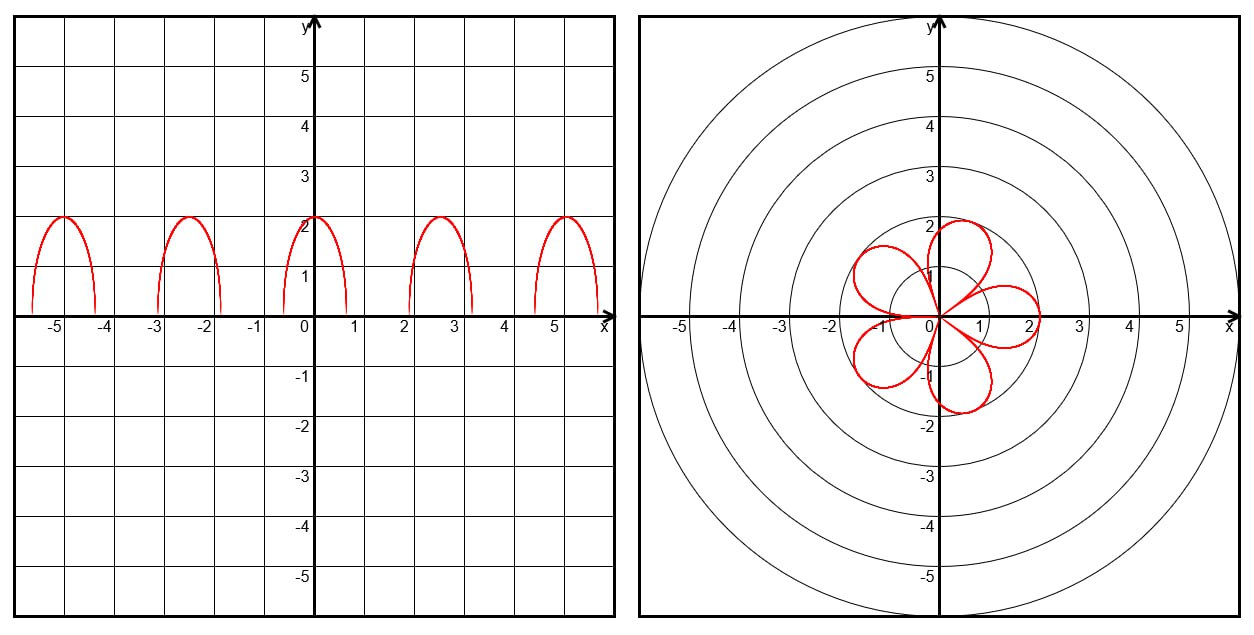


Рисунок 1 – Результат выполнения программы при gridFreq = 6, graphVal = 6, accurancy = 0 и значениях параметров функции a=2, m=2.5

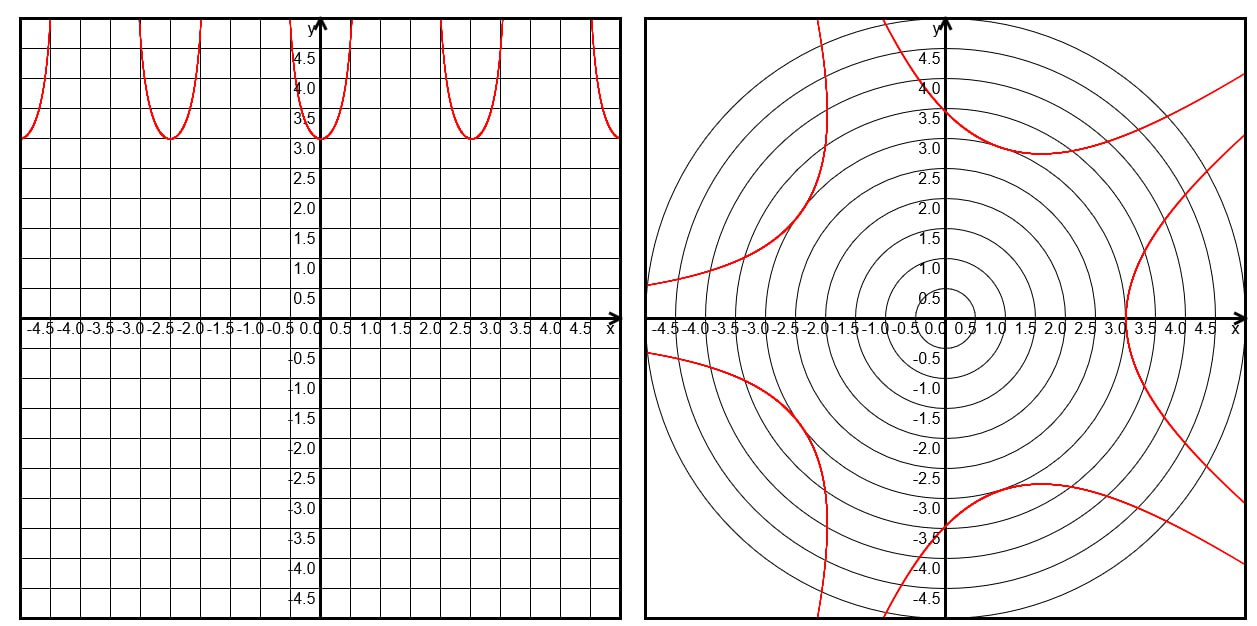


Рисунок 2 – Результат выполнения программы при gridFreq = 10, graphVal = 5, accurancy = 1 и значения параметров функции a=3, m=-2.5

**Вывод:**

В результате выполнения лабораторной работы я ознакомился   
с основными возможностями JavaScript по созданию произвольных изображений и освоил работу с графическими процедурами и функциями.