

### МИНОБРНАУКИ РОССИИ

### федеральное государственное бюджетное образовательное учреждение высшего образования

### «Московский государственный технологический университет «СТАНКИН» (ФГБОУ ВО «МГТУ «СТАНКИН»)

### Институт информационных систем и технологий

### Кафедра инженерной графики

**Отчет по выполнению лабораторной работы №3**

## по дисциплине «Геометрическое моделирование и компьютерная графика»

## Направление подготовки: 09.03.03 «Прикладная информатика»

## Профиль: «Управление данными»

## Руководитель: Толок А.В.

Оценка Подпись Дата

## Студент:

## Баканов П.О.

## Группа ИДБ-20-11

## 

Подпись Дата

## Москва, 2023

# Задание

30 за рабочую программу.

По аналогии со второй лабораторной, выполнение каждого дает дополнительные баллы

1. Сделать кнопку очистки путей градиентного спуска **[+5]**

# Код

sf::Color RFuncSprite::getPixelColor(float Ai, float Bi, float Ci, float Di, unsigned x, unsigned y)

{

float nx, ny, nz, nw;

const float det = std::sqrt(Ai \* Ai + Bi \* Bi + Ci \* Ci + Di \* Di);

nx = Ai / det;

ny = Bi / det;

nz = Ci / det;

nw = Di / det;

nxMatrix[y][x] = nx;

nyMatrix[y][x] = ny;

float shapeValue;

switch (\_shapeType)

{

case Nx:

shapeValue = nx;

break;

case Ny:

shapeValue = ny;

break;

case Nz:

shapeValue = nz;

break;

case Nw:

shapeValue = nw;

break;

default:

throw std::runtime\_error("Wrong shape");

}

return interpolateColors(\_firstColor, \_secondColor, (1.f + shapeValue) / 2);

}

void RFuncSprite::clearGradients()

{ gradientPaths.clear(); }

float RFuncSprite::calculateDeterminant(const std::vector<std::vector<float>> &matrix)

{

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

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

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

}

void RFuncSprite::create(const sf::Vector2u &size, Shapes shapeType)

{

\_image.create(size.x, size.y, sf::Color::Cyan);

\_texture.loadFromImage(\_image);

setTexture(\_texture);

\_shapeType = shapeType;

}

sf::Color RFuncSprite::interpolateColors(const sf::Color &colorFirst, const sf::Color &colorSec, float k)

{

sf::Color val;

val.r = static\_cast<sf::Uint8>(colorFirst.r + (colorSec.r - colorFirst.r) \* k);

val.g = static\_cast<sf::Uint8>(colorFirst.g + (colorSec.g - colorFirst.g) \* k);

val.b = static\_cast<sf::Uint8>(colorFirst.b + (colorSec.b - colorFirst.b) \* k);

val.a = static\_cast<sf::Uint8>(colorFirst.a + (colorSec.a - colorFirst.a) \* k);

return val;

}

void RFuncSprite::drawRFunc(const std::function<float(const sf::Vector2f &)> &rfunc, const sf::FloatRect &subSpace)

{

nxMatrix = new float \*[\_image.getSize().y];

nyMatrix = new float \*[\_image.getSize().y];

for (int i = 0; i < \_image.getSize().y; ++i)

{

nxMatrix[i] = new float[\_image.getSize().x];

nyMatrix[i] = new float[\_image.getSize().x];

}

const sf::Texture &texture = getTexture();

sf::Vector2f spaceStep = {

subSpace.width / static\_cast<float>(\_image.getSize().x),

subSpace.height / static\_cast<float>(\_image.getSize().y),

};

for (unsigned y = 0; y < \_image.getSize().y; ++y)

{

for (unsigned x = 0; x < \_image.getSize().x; ++x)

{

int x1 = x;

int y1 = y;

int x2 = x + 1;

int y2 = y;

int x3 = x;

int y3 = y + 1;

sf::Vector2f spacePoint1 = {subSpace.left + static\_cast<float>(x1) \* spaceStep.x,

subSpace.top + static\_cast<float>(y1) \* spaceStep.y};

sf::Vector2f spacePoint2 = {subSpace.left + static\_cast<float>(x2) \* spaceStep.x,

subSpace.top + static\_cast<float>(y2) \* spaceStep.y};

sf::Vector2f spacePoint3 = {subSpace.left + static\_cast<float>(x3) \* spaceStep.x,

subSpace.top + static\_cast<float>(y3) \* spaceStep.y};

const float z1 = rfunc(spacePoint1);

const float z2 = rfunc(spacePoint2);

const float z3 = rfunc(spacePoint3);

const float Ai = calculateDeterminant({

{spacePoint1.y, z1, 1},

{spacePoint2.y, z2, 1},

{spacePoint3.y, z3, 1},

});

const float Bi = calculateDeterminant({

{spacePoint1.x, z1, 1},

{spacePoint2.x, z2, 1},

{spacePoint3.x, z3, 1},

});

const float Ci = calculateDeterminant({

{spacePoint1.x, spacePoint1.y, 1},

{spacePoint2.x, spacePoint2.y, 1},

{spacePoint3.x, spacePoint3.y, 1},

});

const float Di = calculateDeterminant({

{spacePoint1.x, spacePoint1.y, z1},

{spacePoint2.x, spacePoint2.y, z2},

{spacePoint3.x, spacePoint3.y, z3},

});

\_image.setPixel(x, y, getPixelColor(Ai, Bi, Ci, Di, x, y));

# Результаты работы

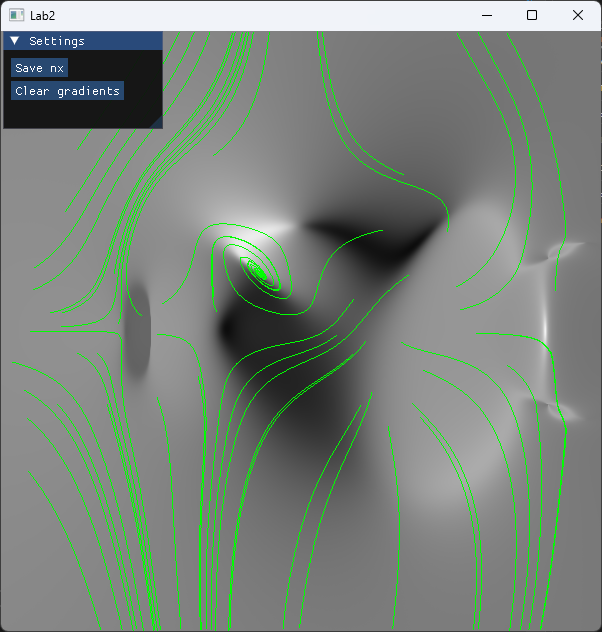


Рис1. Основное окно