Липецкий государственный технический университет

Кафедра АСУ

Лабораторная работа №2

по организации графических систем и систем мультимедиа

Детектор границ Canny

Студент Бровкин А.Н.

Группа М-АС-19

Руководитель Кургасов В.В.

Липецк

2020

**Задание кафедры**

Реализовать программный продукт, в котором есть возможность применения оператора Canny на изображение, для нахождения его границ.

# Теоретические сведения

Приложение было разработано с использованием языка JavaScript.

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми». Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript.

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.JS поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Например, в браузере JavaScript может:

* Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
* Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
* Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии AJAX и COMET).
* Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
* Запоминать данные на стороне клиента («local storage»).

# Детектор границ Canny

Оператор Кэнни (детектор границ Кэнни, алгоритм Кэнни) в дисциплине компьютерного зрения — оператор обнаружения границ изображения. Был разработан в 1986 году Джоном Кэнни (англ. John F. Canny) и использует многоступенчатый алгоритм для обнаружения широкого спектра границ в изображениях.

Кэнни изучил математическую проблему получения фильтра, оптимального по критериям выделения, локализации и минимизации нескольких откликов одного края. Он показал, что искомый фильтр является суммой четырёх экспонент. Он также показал, что этот фильтр может быть хорошо приближен первой производной Гауссианы. Кэнни ввёл понятие подавления не максимумов (англ. Non-Maximum Suppression), которое означает, что пикселями границ объявляются пиксели, в которых достигается локальный максимум градиента в направлении вектора градиента.

Хотя его работа была проведена на заре компьютерного зрения, детектор границ Кэнни до сих пор является одним из лучших детекторов. Кроме особенных частных случаев трудно найти детектор, который бы работал существенно лучше, чем детектор Кэнни [1].

Алгоритм состоит из пяти отдельных шагов:

1. Сглаживание. Размытие изображения для удаления шума.

2. Поиск градиентов. Границы отмечаются там, где градиент изображения приобретает максимальное значение.

3. Подавление не-максимумов. Только локальные максимумы отмечаются как границы.

4. Двойная пороговая фильтрация. Потенциальные границы определяются порогами.

5. Трассировка области неоднозначности. Итоговые границы

Перед применением детектора, изображение преобразуется в оттенки серого, чтобы уменьшить вычислительные затраты. Этот этап характерен для многих методов обработки изображений [2].

## «Серый» фильтр

Чтобы изображение имело только оттенки серого (от черного до белого) требуется преобразовать каждый пиксель изображения следующим образом:

AVG = (R+G+B)/3,

R’ = AVG,

G’ = AVG,

B’ = AVG.

Где R, G, B – красный, зеленый, синий соответственно цвета модели пикселя исходного изображения, R’, G’, B’ - красный, зеленый, синий соответственно цвета модели пикселя преобразованного изображения.

## Фильтр «Негатив»

Чтобы изображение стало негативным требуется каждый пиксель сделать негативным, то есть требуется инвертировать каждый пиксель. Так как максимальное значение каждого цвета в пикселе равняется 255, то достаточно вычесть из этого значения исходные цвета, то есть:

R’ = 255 - R,

G’ = 255 - G,

B’ = 255 - B.

## Регулирование яркости

Для изменения яркости изображения достаточно вычисление значений каждого из каналов пикселя по заданному коэффициенту яркости (K), используя следующую формулу:

R’ = R + K,

G’ = G + K,

B’ = B + K.

Если K > 0, то изображение становиться ярче, а если K < 0, то темнее

# Листинг кода

## JavaScript

function handleFile(files) {

    file = files[0];

    if (file == undefined || file == null) return;

    reader = new FileReader();

    reader.onload = function () {

        img = new Image();

        img.onload = function () {

            var canvas = document.getElementById("canvas");

            canvas.width = img.width;

            canvas.height = img.height;

            render(img);

        }

        img.src = reader.result;

    }

    reader.readAsDataURL(file);

};

function saveImage() {

    var link = document.createElement("a");

    link.setAttribute("href", document.getElementById("canvas").toDataURL());

    link.setAttribute("download", "");

    link.click();

};

var gl, program, canvas;

// Функция создания шейдера по типу и id источника в структуре DOM

function getShader(type, id) {

    var source = document.getElementById(id).innerHTML;

    // Создаем шейдер по типу

    var shader = gl.createShader(type);

    // Установка источника шейдера

    gl.shaderSource(shader, source);

    // Компилируем шейдер

    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS)) {

        console.error("Ошибка компиляции шейдера: " + gl.getShaderInfoLog(shader));

        gl.deleteShader(shader);

        return null;

    }

    return shader;

};

window.onload = function () {

    canvas = document.getElementById("canvas");

    try {

        gl = canvas.getContext("webgl", { preserveDrawingBuffer: true }) || canvas.getContext("experimental-webgl", { preserveDrawingBuffer: true });

    }

    catch (e) { }

    if (!gl) {

        alert("Ваш браузер не поддерживает WebGL");

    }

    // Получаем шейдеры

    var fragmentShader = getShader(gl.FRAGMENT\_SHADER, '2d-fragment-shader');

    var vertexShader = getShader(gl.VERTEX\_SHADER, '2d-vertex-shader');

    // Создаем объект программы шейдеров

    program = gl.createProgram();

    // Прикрепляем к ней шейдеры

    gl.attachShader(program, vertexShader);

    gl.attachShader(program, fragmentShader);

    // Связываем программу с контекстом webgl

    gl.linkProgram(program);

    if (!gl.getProgramParameter(program, gl.LINK\_STATUS)) {

        alert("Не удалсь установить шейдеры");

    }

    gl.useProgram(program);

    var texCoordLocation = gl.getAttribLocation(program, "a\_texCoord");

    // Координаты текстур для прямоугольника

    var texCoordBuffer = gl.createBuffer();

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

    gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array([

        0.0, 0.0,

        1.0, 0.0,

        0.0, 1.0,

        0.0, 1.0,

        1.0, 0.0,

        1.0, 1.0]), gl.STATIC\_DRAW);

    gl.enableVertexAttribArray(texCoordLocation);

    gl.vertexAttribPointer(texCoordLocation, 2, gl.FLOAT, false, 0, 0);

}

var stateLocation

function render(image) {

    gl.viewport(0, 0, image.width, image.height);

    // Инициалиируем данные вершин

    var positionLocation = gl.getAttribLocation(program, "a\_position");

    // Создаем текстуры

    var texture = gl.createTexture();

    gl.bindTexture(gl.TEXTURE\_2D, texture);

    // Установка параметров, чтобы можно было отобразить изображение любого размера

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.NEAREST);

    gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.NEAREST);

    // Загрузка изображения в текстуры

    gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

    // Установка форм

    var resolutionLocation = gl.getUniformLocation(program, "u\_resolution");

    var textureSizeLocation = gl.getUniformLocation(program, "u\_textureSize");

    stateLocation = gl.getUniformLocation(program, "u\_state");

    // Установка разрешения

    gl.uniform2f(resolutionLocation, canvas.width, canvas.height);

    // Установка размера изображения

    gl.uniform2f(textureSizeLocation, image.width, image.height);

    // Создаем буфер для положения углов прямоугольника

    var buffer = gl.createBuffer();

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

    gl.enableVertexAttribArray(positionLocation);

    gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0, 0);

    // Установка прямоугольника такого же размера как изображение

    setRectangle(gl, 0, 0, image.width, image.height);

    //простая отрисовка изображения

    draw(0)

};

function setCanny() {

    draw(1);

}

function draw(n) {

    if (n > 5) return;

    var image = new Image();

    image.onload = function () {

        gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED\_BYTE, image);

        if (n != 0) return draw(n + 1);

    }

    gl.uniform1i(stateLocation, n);

    gl.drawArrays(gl.TRIANGLES, 0, 6);

    image.src = document.getElementById("canvas").toDataURL();

};

function setRectangle(gl, x, y, width, height) {

    var x1 = x;

    var x2 = x + width;

    var y1 = y;

    var y2 = y + height;

    gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array([

        x1, y1,

        x2, y1,

        x1, y2,

        x1, y2,

        x2, y1,

        x2, y2]), gl.STATIC\_DRAW);

};

## HTML

<!DOCTYPE html>

<html>

    <head>

        <link rel="stylesheet" href="style.css" type="text/css"/>

        <script src="script.js"></script>

    </head>

    <body>

        <main>

            <nav>

                <label class="but" for="inputImageElement" >Открыть картинку</label>

                <input style="display: none;" type="file" id="inputImageElement" accept="image/\*" onchange="handleFile(this.files)"/>

                <span class="but" onclick="setCanny()">Оператор Canny</span>

                <span class="but" style="margin-top: 100px;" id="FitWidth" onclick="saveImage()">Сохранить</span>

            </nav>

            <section>

                <div id="ImageConteyner">

                    <canvas id="canvas" width="400px" height="400px"></canvas>

                </div>

            </section>

        </main>

            <!-- vertex shader -->

    <script id="2d-vertex-shader" type="x-shader/x-vertex">

        attribute vec2 a\_position;

        attribute vec2 a\_texCoord;

        uniform vec2 u\_resolution;

        varying vec2 v\_texCoord;

        void main() {

           vec2 a = a\_position / u\_resolution;

           vec2 b = a \* 2.0;

           vec2 clipSpace = b - 1.0;

           gl\_Position = vec4(clipSpace \* vec2(1, -1), 0, 1);

           v\_texCoord = a\_texCoord;

        }

    </script>

    <!-- fragment shader -->

    <script id="2d-fragment-shader" type="x-shader/x-fragment">

            precision highp float;

            uniform sampler2D u\_image;

            uniform vec2 u\_textureSize;

            uniform int u\_state;

            varying vec2 v\_texCoord;

            const float PI = 3.141592653589793238462643383279502884197169;

            const mat3 X\_COMPONENT\_MATRIX = mat3(

                1., 0., -1.,

                2., 0., -2.,

                1., 0., -1.

            );

            const mat3 Y\_COMPONENT\_MATRIX = mat3(

                1., 2., 1.,

                0., 0., 0.,

                -1., -2., -1.

            );

            vec2 onePixel;

            float Q;

            float mid(vec4 pix){

                return (pix.r + pix.g + pix.b) / 3.;

            }

            float round(float A){

                if(mod(A, 1.) < .5){

                    return floor(A);

                }

                else{

                    return ceil(A);

                }

            }

            float convoluteMatrices(mat3 A, mat3 B){

                return dot(A[0], B[0]) + dot(A[1], B[1]) + dot(A[2], B[2]);

            }

            float grayScale(){

                vec4 pix = texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 0));

                return dot(pix.rgb, vec3(0.299, 0.587, 0.114));

            }

            float gaussianBlur(){

            vec4 colorSum =

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-2, -2)) \* 2. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-2, -1)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-2, 0)) \* 5. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-2, 1)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-2, 2)) \* 2. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-1, -2)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-1, -1)) \* 9. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-1, 0)) \* 12. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-1, 1)) \* 9. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(-1, 2)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, -2)) \* 5. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, -1)) \* 12. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 0)) \* 15. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 1)) \* 12. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 2)) \* 5. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(1, -2)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(1, -1)) \* 9. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(1, 0)) \* 12. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(1, 1)) \* 9. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(1, 2)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(2, -2)) \* 2. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(2, -1)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(2, 0)) \* 5. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(2, 1)) \* 4. +

                texture2D(u\_image, v\_texCoord + onePixel \* vec2(2, 2)) \* 2.;

                return mid(colorSum / 159.);

            }

            float calcG(float x, float y, int S){

                mat3 imgMat = mat3(0.);

                for (int i = 0; i < 3; i++) {

                    for (int j = 0; j < 3; j++) {

                        imgMat[i][j] = mid(texture2D(u\_image, v\_texCoord + onePixel \* vec2(j-int(x), i-int(y))));

                    }

                }

                float gradX = convoluteMatrices(X\_COMPONENT\_MATRIX, imgMat);

                float gradY = convoluteMatrices(Y\_COMPONENT\_MATRIX, imgMat);

                float G = sqrt(gradX \* gradX + gradY \* gradY);

                if(S == 1)

                    if(G != 0.0){

                        Q = round(atan(gradX, gradY)/(PI/4.))\*(PI/4.)-(PI/2.);

                    }

                    else{

                        Q = -10.5;

                    }

                return G;

            }

            float nonMaximumSuppression(float Q, float T){

                if(Q == -10.5) return 0.;

                float dx = sign(cos(Q));

                float dy = -sign(sin(Q));

                float TH = calcG(dx, dy, 0);

                float TL = calcG(-dx, -dy, 0);

                if(TH <= T && T >= TL) return T; else return 0.;

            }

            float gradient(){

                float G = calcG(0., 0., 1);

                return nonMaximumSuppression(Q, G);

            }

            float dThreshold(float down, float up){

                float pix = mid(texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 0)));

                if (pix >= up) return 1.;

                if (pix <= down) return 0.;

                return .5;

            }

            float hysteresis(float low, float high){

                int x = 0, y = 0, p = 0, count = 0;

                float k = mid(texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 0)));

                if(k != 0.){

                    if(k >= high) return k;

                    for (int i = -1; i < 2; i++) {

                        for (int j = -1; j < 2; j++) {

                            if(i != 0 && j != 0 ){

                                p = 0;

                                for(int s = 0; s < 15000; s++){

                                    x += j;

                                    y += i;

                                    if(y < 0 || x < 0 || x >= int(u\_textureSize.x) || y >= int(u\_textureSize.y)) break;

                                    k = mid(texture2D(u\_image, v\_texCoord + onePixel \* vec2(x, y)));

                                    if(k <= low) break;

                                    p++;

                                }

                                if(p >= 1) count++;

                            }

                        }

                    }

                    if(count >= 1) return 1.;

                }

                else{

                    return 0.;

                }

            }

            void main() {

                onePixel = vec2(1.) / u\_textureSize;

                float result;

                //Простая отрисовка

                if (u\_state == 0){

                    vec4 pix = texture2D(u\_image, v\_texCoord + onePixel \* vec2(0, 0));

                    gl\_FragColor = vec4(pix);

                    return;

                }

                // Обесцвечивание

                if(u\_state == 1){

                    result = grayScale();

                }

                // Сглаживание

                if(u\_state == 2){

                    result = gaussianBlur();

                }

                // Поиск градиентов и подавление не-максимумов

                if(u\_state == 3){

                    result = gradient();

                }

                // Двойная пороговая фильтрация

                if(u\_state == 4){

                    result = dThreshold(.5, .6);

                }

                // Трассировка области неоднозначности

                if(u\_state == 5){

                    result = hysteresis(.5, 0.75);

                }

                gl\_FragColor = vec4(vec3(result), 1.);

            }

        </script>

    </body>

</html>

# Результаты выполнения работы

На рисунках 1-2 показаны результаты выполнения работы.



1. Оригинал изображения



1. Изображение с примененным оператором Canny

**Вывод**

В ходе выполнения данной лабораторной работы были получил практические навыки обработки графики, а именно написание шейдеров с использованием языка GLSL и использования их в библиотеке WebGL с помощью языка JavaScript. Так же был изучен и реализован алгоритм Canny для обнаружения границ изображения. Все расчеты, связанные с обработкой изображений, выполняются на GPU. Мной было написано веб-приложение с простым интерфейсом позволяющее загрузить изображение и быстро сохранить получившийся результат обработки.