Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра АСУ

ЛАБОРАТОРНАЯ РАБОТА №2

по организации графических систем и систем мультимедиа

Реализация алгоритма выделения границ и контуров Canny

Студент Лапшова А.Г.

Группа М-АС-19

Руководитель Кургасов В.В.

Липецк 2020г.

Задание кафедры

Реализовать алгоритм выделения границ и контуров Canny. Наличие графического меню пользователя. Обработка на GPU.

Цель работы

Освоить на практике преобразование загружаемых графических файлов.

1 Теоретические сведения

1.1 Общие сведения

Для реализации программного продукта была выбрана технология WebGL.

WebGL (Web Graphics Library) - программная библиотека для языка JavaScript предназначенная для визуализации интерактивной трехмерной графики и двухмерной графики в пределах совместимости веб-браузера без использования плагинов. WebGL приносит в веб трехмерную графику, вводя API, который построен на основе OpenGL ES 2.0, что позволяет его использовать в элементах canvas HTML5 [1].

Вся работа веб-приложений с использованием WebGL основана на коде JavaScript, а некоторые элементы кода - шейдеры могут выполняться непосредственно на графических процессорах на видеокартах, благодаря чему разработчики могут получить доступ к дополнительным ресурсам компьютера, увеличить быстродействие. Таким образом, для создания приложений разработчики могу использовать стандартные для веб-среды технологии HTML/CSS/JavaScript и при этом также применять аппаратное ускорение графики.

Если создание настольных приложений работающих с 2d и 3d-графикой нередко ограничивается целевой платформой, то здесь главным ограничением является только поддержка браузером технологии WebGL. А сами веб-приложения, построенные с использованием данной платформы, будут доступны в любой точке земного шара при наличии сети интернет вне зависимости от используемой платформы: то ли это десктопы с ОС Windows, Linux, Mac, то ли это смартфоны и планшеты, то ли это игровые консоли.

Преимуществами использования WebGL являются:

- Кроссбраузерность и отсутствие привязки к определенной платформе. Windows, MacOS, Linux - все это не важно, главное, чтобы ваш браузер поддерживал WebGL;

- Использование языка JavaScript, который достаточно распространен;
- Автоматическое управление памятью. В отличие от OpenGL в WebGL не надо выполнять специальные действия для выделения и очистки памяти;
- Поскольку WebGL для рендеринга графики использует графический процессор на видеокарте (GPU), то для этой технологии характерна высокая производительность, которая сравнима с производительностью нативных приложений [2].

1.2 Конвейер WebGL

Основными элементами при построении изображений с использованием данной технологии являются шейдеры. Именно через шейдерную программу задается положение и цвет каждой вершины наших линий. В нашей задаче используется два шейдера: вершинный и фрагментный. При построении линий в трехмерном пространстве вершинный шейдер отвечает за положение вершин в пространстве, основываясь на значениях видовой матрицы и матрицы перспективной проекции. Фрагментный шейдер используется для вычисления цвета наших линий [3].

На рисунке 1 представлен конвейер WebGL.

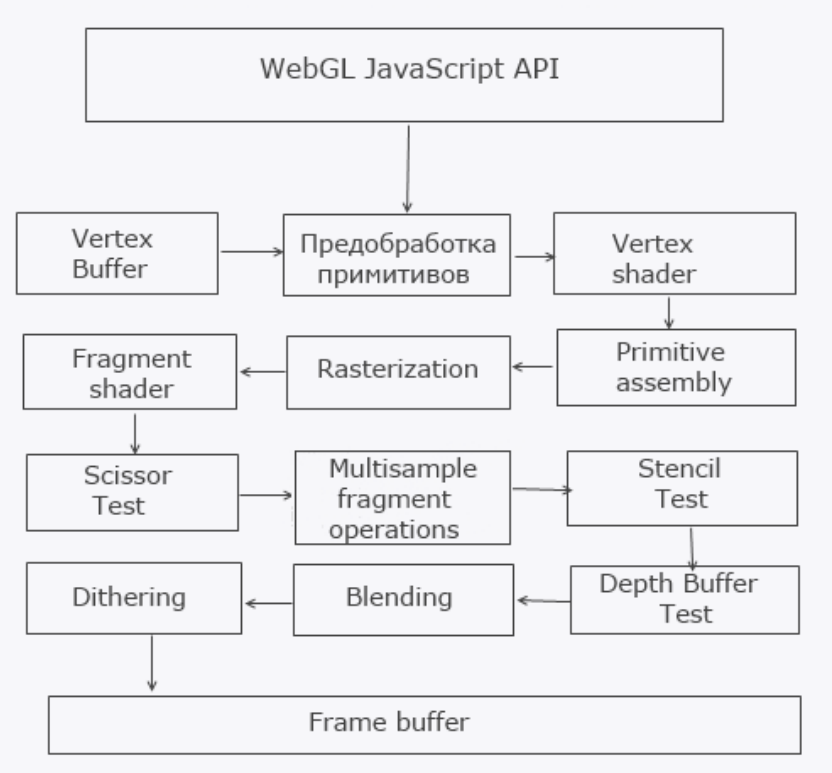

Рисунок 1 – Конвейер WebGL

Поэтапный разбор данного конвейра:

Вначале мы создаем набор вершин в буфере вершин (Vertex Buffer). По этим вершинам впоследствии будут составлены геометрические примитивы, а из примитивов - объекты. И проводим некоторую предобработку.

Затем содержимое буфера вершин поступает на обработку в вершинный шейдер (Vertex Shader). Шейдер производит над вершинами некоторые трансформации, например, применяет матрицы преобразования и т.д. Шейдеры пишутся самим разработчиком, поэтому программист может применить различные преобразования по своему усмотрению.

На следующем этапе (Primitive Assembly) конвейер получает результат вершинного шейдера и пытается измененные вершины сопоставить в отдельные примитивы - линии, треугольники, спрайты. Также на этом этапе определяется, входит ли примитив в видимое пространство. Если нет, то он обрезается. Оставшиеся примитивы передаются на следующий этап конвейера.

Далее на этапе растеризации (Rasterization) полученные примитивы преобразуются в фрагменты, которые можно представить как пиксели, которые затем будут отрисованы на экране

И затем в дело вступает фрагментный шейдер (Fragment shader). (В технологиях Direct3D, XNAпрямым аналогом является пиксельный шейдер). Фрагментый шейдер производит преобразования с цветовой составляющей примитивов, наполняет их цветом, точнее окрашивает пиксели, и в качестве вывода передает на следующий этап измененные фрагменты.

Следующий этап представляет собой ряд преобразований над полученными с фрагментного шейдера фрагментами. Собственно он состоит из нескольких подэтапов:

- Scissor Test: на этом этапе проверяется, находится ли фрагмент в пределах отсекающего прямоугольника. Если фрагмент находится в пределах этого прямоугольника, то он передается на следующий этап. Если же нет, то он отбрасывается и больше не принимает участия в обработке.
- Multisample Fragment Operations: на данном этапе у каждого фрагмента изменяются цветовые составляющие, производится сглаживание (anti-alising), чтобы объект выглядел более плавно на экране.
- Stencil Test: здесь фрагмент передается в буфер трафаретов (stencil buffer). Если вкратце, то в этом буфере дополнительно отбрасываются те фрагменты, которые не должны отображаться на экране. Как правило, данный буфер используется для создания различного рода эффектов, например, эффект теней.
- Depth Buffer Test - тест буфера глубины. В буфере глубины (depth buffer, а также называется, z-buffer) сравнивается z-компонента фрагмента, и если она больше значения в буфере глубины, то, следовательно, данный фрагмент расположен к смотрящему на

трехмерную сцену ближе, чем предыдущий фрагмент, поэтому текущий фрагмент проходит тест. Если же z-компонента больше значения в буфере глубины, то, следовательно, данный фрагмент находится дальше, поэтому он не должен быть виден и отбрасывается.

- Blending: на данном этапе происходите небольшое смешение цветов, например, для создания прозрачных объектов.
- Dithering: здесь происходит смешение цветов, для создания тонов и полутонов.

Frame Buffer: и здесь наконец полученные после предобработки фрагменты превращаются в пиксели на экране.

2 Основная часть

2.1 Описание реализованного алгоритма

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.

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 определяются путём подавления всех краёв, несвязанных с определенными (сильными) границами [4].

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

2.2 Описание работы программы

В начале работы программы происходит проверка доступности WebGL контекста и его получение, инициализация шейдерных программ, буферов вершин объекта (в последствии будет загружена текстура изображения).

После загрузки изображения запускается алгоритм обработки. Перерисовка осуществляется путем обновления всей сцены (указывается также примитив, с помощью которого производится обновление). Во время работы алгоритма, изображение передается на следующий этап в виде

текстуры. В момент перерисовки шейдер получает номер этапа обработки и совершает выбор манипуляций с исходным изображением, которое соответствуют определенному этапу алгоритма Кэнни.

Первый этап характеризуется переводом исходного изображения в оттенки серого (производится путем перемножения значения цветом на коэффициента перевода: 0.299 для красного, 0.587 для зеленого и 0.114 для синего цветов изображения).

На втором этапе происходит подавление шума с помощью размытия путем перемножения значений пикселей исходного изображения на соответствующую матрицу.

Третий этап реализует поиск градиентов и подавление не максимумов. Поиск градиентов происходит с помощью оператора Собеля. Оператор Собеля основан на свёртке изображения небольшими целочисленными фильтрами в вертикальном и горизонтальном направлениях. После чего получаем градиент и угол его направления. При подавлении не максимумов пикселями границ объявляются пиксели, в которых достигается локальный максимум градиента в направлении вектора градиента.

При выполнении четвертого этапа выполняется двойная пороговая фильтрация. Ее смысл в том, что используется две границы фильтрации. Если значение пикселя выше верхней границы – он принимает максимальное значение, если ниже – пиксель подавляется, точки со значением, попадающим в диапазон между порогов, принимают фиксированное среднее значение.

На пятом этапе происходит трассировка области неоднозначности. Это означает то, что те пиксели, которые на предыдущем этапе получили среднее значение проверяются на наличие возле них пикселя с максимальным значением, если таковой есть, то эти пикселям устанавливается максимальное значение, в противном случае минимальное значение.

2.3 Листинг кода

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <style type="text/css">
        body {
            margin: 0px;
            text-align: center;
        }

        #menu {
            margin: 0.5rem 0 1rem 0;
        }
    </style>
        <link rel="stylesheet" href="../styles.css">
        <link rel="stylesheet" href="../css/bootstrap.min.css">
    <title>Canny</title>
</head>

<body>
    <div id="menu">
            <div class="alert alert-info">Выделение границ и контуров</div>
            <div class="input-group">
                    <div class="custom-file">
                      <input type="file" class="custom-file-input"
id="loadImage" type="file" accept="image/*">
                      <label class="custom-file-label"
for="inputGroupFile04">Выбрать файл</label>
                    </div>
                    <div class="input-group-append">
                      <button class="btn btn-outline-secondary"
type="button" id="saveImage">Сохранить</button>
                    </div>
            </div>
    </div>
    <canvas id="canvas" width="500" height="500" style="display: none;">
        Canvas не поддерживается
    </canvas>
    <div id="changes">
    </div>
    <!-- vertex shader -->
    <script id="2d-vertex-shader" type="x-shader/x-vertex">
        //<!-- Передает координаты вершины в шейдер -->
        attribute vec2 a_position;
        attribute vec2 a_texCoord;

        //<!-- Константные значения, задаются для всего примитива -->
        uniform vec2 u_resolution;

        //<!-- Задается в вершинном - передается во фрагментный где может
быть использована -->
        varying vec2 v_texCoord;

        //<!-- Генерация окончательных координат вершин -->
        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 = 3.14159265358979323846264338327950288419716939;
            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;
            // 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>
    <script type="text/javascript">
        "use strict";

        var el = document.getElementById("loadImage");
        el.addEventListener("change",
            function () {
                var input, file, reader, img;
                input = document.querySelector('input');

                file = input.files[0];
                if (file == undefined || file == null) return;
                reader = new FileReader();
                reader.onload = function () {
                    img = new Image();
                    img.onload = function () {
                        var bd = document.getElementById("changes");
                        bd.innerHTML = "";
                        bd.append(img);
                        var canvas = document.getElementById("canvas");
                        canvas.width = img.width;
                        canvas.height = img.height;
                        render(img);
                    }
                    img.src = reader.result;
                }
                reader.readAsDataURL(file);
            });

        el = document.getElementById("saveImage");
        el.addEventListener("click",
            function () {
                var link = document.createElement("a");
                link.setAttribute("href",
document.getElementById("canvas").toDataURL());
                link.setAttribute("download", "");
                link.click();
            });

        /********** Работа с WebGL **********/
        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();
            //Привязка набора координат в качестве буфера вершин
            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); //Включение
атрибута вершин
            //Установка указателя на чтение из буфера вершин
            gl.vertexAttribPointer(texCoordLocation, 2, gl.FLOAT, false, 0,
0); // 2 означает кол-во координат
        }

        function render(image) {
            //Установка области рисования
            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");
            var 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);//Включение
атрибута вершин
            gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0,
0);

            // Установка прямоугольника такого же размера как изображение
            setRectangle(gl, 0, 0, image.width, image.height);

            var image
            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 == 4)
                    {
                        var bd = document.getElementById("changes");
                        bd.append(image);
                    }
                    return draw(n + 1);
                }
                gl.uniform1i(stateLocation, n);
                //Отрисовка в WebGL какой примитив, первая вершина
примитива, сколько вершин для отрисовки
                console.log(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);
        };
    </script>
</body>

</html>
```

3 Результат работы программы

Репозиторий проекта находится на GitHub [5].

Рисунок 2 – Определение границ изображения

Вывод

В ходе выполнения данной лабораторной работы удалось реализовать алгоритм определения границ и контуров Canny. Также были получены практические навыки обработки графики, а именно создание шейдеров на языке GLSL, который входит в состав библиотеки WebGL. Все расчеты, связанные с обработкой изображений, выполняются на GPU.

Данная работа реализована без использования фреймворков и имеет простой и понятный интерфейс.

Список источников

1. WebGL – Интерфейсы веб API | MDN [Электронный ресурс]. – Электрон. текст. дан. – режим доступа: https://developer.mozilla.org/ru/docs/Web/API/WebGL_API, свободный.

2. Введение в WebGL [Электронный ресурс]. – Электрон. текст. дан. – режим доступа: https://metanit.com/web/webgl/1.1.php, свободный.

3. Знакомство с WebGL [Электронный ресурс]. – Электрон. текст. дан. – режим доступа: https://habr.com/ru/post/112430/, свободный.

4. Детектор границ Канни [Электронный ресурс]. – Электрон. текст. дан. – режим доступа: https://habr.com/ru/post/114589/, свободный.

5. Репозиторий проекта [Электронный ресурс]. – Электрон. текст. дан. – режим доступа: https://github.com/nwdles/Graphics-and-Multimedia, свободный.