**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.

# Теоретические сведения

Для реализации программного продукта была выбрана технология WebGL.

WebGL представляет собой технологию, базирующуюся на OpenGL ES 2.0 и предназначенную для рисования и отображения интерактивной 2D- и 3D-графики в веб-браузерах. При этом для работы с данной технологией не требуются сторонние плагины или библиотеки. 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.

# Основная часть

## 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.

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

## 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.

## 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 = 3.141592653589793238462643383279502884197169;

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;

// 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>

# Результат работы программы

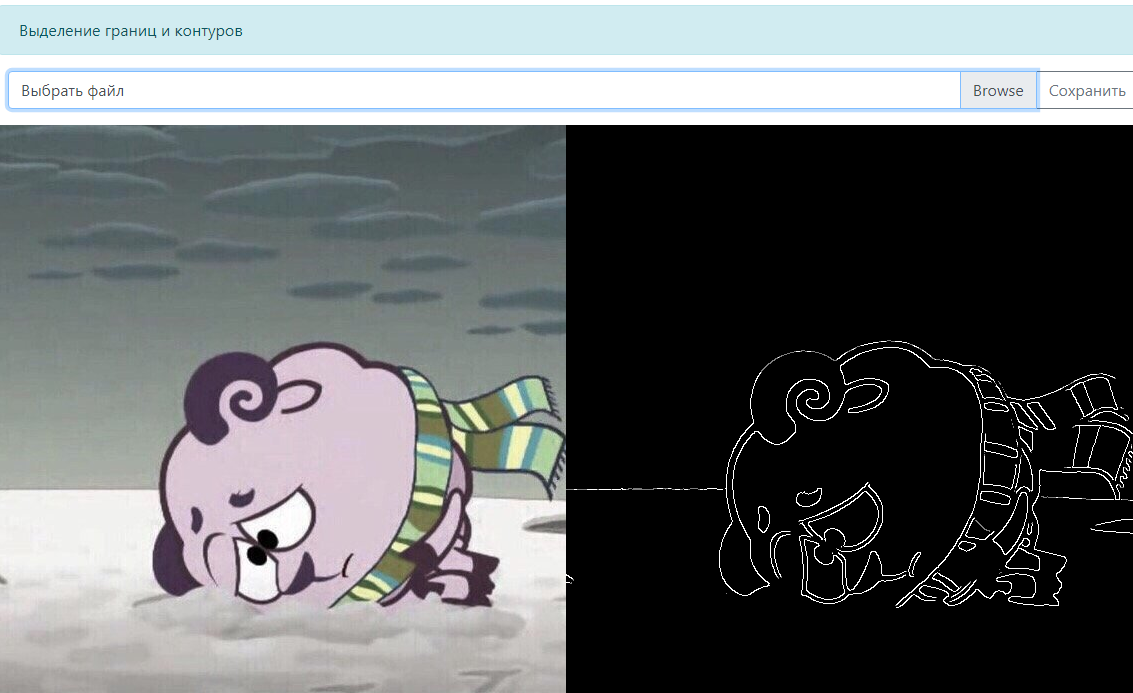


Рисунок 1 – Определение границ изображения

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.