Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Организация графических систем

Лабораторная работа №2

«Реализация построения контуров изображения на основе детектора Canny»

Студентка Комаричева А.Г.

Группа М-АС-19

Руководитель Кургасов В.В.

Липецк 2020 г.

Задание

На любом удобном студенту языке программирования разработать программу, осуществляющую применение детектора границ Канни к изображениям с последующим сохранением. Должна быть предусмотрена возможность преобразования файла векторного формата в растровое изображение для дальнейшей обработки. Вычисления при осуществлении функционала должны быть возложены на графический процессор.

Оглавление

Введение

Анализ изображения — это процесс выделения нужной информации из изображения с помощью автоматических систем. Системы анализа не ограничиваются разделением областей сцены на фиксированное число классов. Они предназначены для описания сложных сцен, разнообразие которых может быть настолько большим, что их нельзя описать с помощью заранее заданных терминов. В системе анализа также могут использоваться методы искусственного интеллекта для управления различными блоками системы и организации эффективного доступа к базе априорных сведений об объектах. Признак изображения — это его простейшая характеристика или свойство. Некоторые признаки являются естественными в том смысле, что они устанавливаются визуальным анализом изображения, тогда как другие, так называемые искусственные признаки, получаются в результате его специальной обработки и измерений. Естественные признаки: светлота (яркость), текстура различных областей изображения и форма контуров объектов. Обычно анализ изображения включает в себя получение внешнего контура изображенных объектов и запись координат точек этого контура. Чаще всего требуется получить внешний контур в виде замкнутой кривой или совокупности отрезков дуг.

Существует целый ряд алгоритмов, решающих задачу фильтрации контуров, такие как операторы Собеля, Лапласа, Прюитта, Робертса и Канни. Оператор Канни использует многоступенчатый алгоритм для обнаружения широкого спектра границ в изображениях [3, 4]. Он не ограничивается вычислением градиента сглаженного изображения. В контуре границы оставляются только точки максимума градиента изображения, а не максимальные точки, лежащие рядом с границей, удаляются. Здесь также используется информация о направлении границы для того, чтобы удалять точки именно рядом с границей и не разрывать саму границу вблизи локальных максимумов градиента. Затем с помощью двух порогов удаляются слабые

границы. Фрагмент границы при этом обрабатывается как целое. Если значение градиента где-нибудь на прослеживаемом фрагменте превысит верхний порог, то этот фрагмент остается также «допустимой» границей и в тех местах, где значение градиента падает ниже этого порога, до тех пор, пока она не станет ниже нижнего порога. Если же на всем фрагменте нет ни одной точки со значением, большим верхнего порога, то он удаляется.

В данной лабораторной работе мы реализуем детектор границ Канни. Языком реализации в данной работе выступает JavaScript.

1 Теоретические сведения

1.1 Детектор границ Канни

Контурный анализ — это область науки, посвященная обработке изображения, содержащая в себе набор алгоритмов и методов по нахождению границ (контуров) объектов и работе с границами объектов на изображении.

Существует множество различных алгоритмов поисков контуров, таких как:

— Оператор Робертса;

— Оператор Прюита;

— Оператор Собеля;

— Детектор границ Канни.

При работе с методами контурного анализа нельзя наверняка сказать, какой из них будет работать лучше на определенной выборке изображений, однако в силу наличия детектора границ Канни в библиотеке OpenCV, а также на основании его популярности в научном мире, будет рассмотрен именно этот алгоритм поиска контуров на изображении.

Детектор границ Канни — оператор для поиска границ объектов на изображении. Был изобретен Джоном Канни в 1986 и использует многоступенчатый алгоритм для поиска широкого спектра границ на изображениях. Работу алгоритма можно представить в виде последовательности шагов:

1. Сглаживание;
2. Поиск градиентов;
3. Подавление “немаксимумов”;
4. Двойная пороговая фильтрация;
5. Трассировка области неоднозначности.

Рассмотрим каждый из пунктов подробнее.

1. Сглаживание

Кадры, которые подаются на вход детектору в общем случае имеют множество шумов. Шумы мешают детектору найти истинные границы объектов, поэтому изначально следует сгладить изображение. В детекторе границ Канни используется фильтр Гаусса с ядром N × N. Фильтр Гаусса представляет из себя матрицу свертки заполненную по закону нормального распределения.

2. Поиск градиентов

Метод основан на перепадах яркостей изображения, для поиска векторов направлений (градиентов) смены яркостей используется оператор Собеля. Изображение сворачивается в вертикальном и горизонтальном направлениях при помощи двух сепарабельных фильтров ядра 3 × 3. Суть этих свёрток заключается в поиске приближенных значений производных в соответствующих направлениях. Имеется исходное изображение A , Gx и Gy — результирующие изображения, где каждая точка содержит приближенные производные по горизонтальному и вертикальному направлениям соответственно. Они вычисляются следующим образом:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A \quad , \quad G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A$$

Рисунок 1 – Gx и Gy — результирующие изображения

При этом приближенное значение градиента можно вычислить по формуле:

$$G = \sqrt{{G_x}^2 + {G_y}^2},$$

Рисунок 2 – Приближенное значение градиента

а направление градиента по формуле:

$$\theta = arctg\left(\frac{G_y}{G_x}\right).$$

Рисунок 3 – Направление градиента

Потенциальными границами назначаются те точки изображения, в которых градиент принимает наибольшее значение.

3. Подавление “немаксимумов”

После того как определены все значения градиентов и их направления, находятся те пиксели, в которых достигается локальный максимум. Хорошей иллюстрацией этого процесса является рисунок 4.

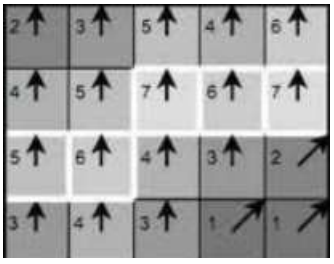

Рисунок 4 – Иллюстрация выбора локального максимума

Области, обведенные белым цветом, применяются как потенциальные границы.

4. Двойная пороговая фильтрация

Оператор использует так называемые пороги для определения существования границы в данном пикселе изображения. Их значения лежат в промежутке от 0 до 255. Чем меньше задается порог, тем больше будет находится границ, но в то же время результат станет более восприимчив к шуму, выделяя лишнее. Если же задается высокий порог, то это может привести к тому, что будут проигнорированы слабые края. Алгоритм выделения границ Канни использует два порога фильтрации: если значение пикселя выше верхней границы — он принимает максимальное значение (граница считается достоверной), если ниже — пиксель подавляется, точки со значением, попадающим в диапазон между порогов, принимают фиксированное среднее значение (они будут уточнены на следующем этапе).

5. Трассировка области неоднозначности

На данном этапе анализируются группы пикселей, попавших в среднюю область. Точки, лежащие по соседству с границей, присоединяются к ней, остальные — отбрасываются.

Данная работа реализована на языке JS и фреймворке Vue.js для быстрого и удобного написания пользовательского интерфейса. Сам алгоритм Канни реализован на шейдерном языке GLSL так как он проще в использовании по сравнению с другими средствами использования вычислительных средств видеокарты (например, CUDA).

1.2 Язык реализации

JavaScript — динамический, интерпретируемый язык со слабой типизацией, обычно используемый для написания скриптов на стороне клиента. JavaScript поддерживается всеми современными браузерами и позволяет вы-

полнять на стороне клиента достаточно сложные вычисления. Для JS существует большое количество фреймворков для простого рендеринга 3D-сцен, отрисовки пользовательских интерфейсов. Один из таких фреймворков – Vue.js. Vue.js — JavaScript-фреймворк с открытым исходным кодом для создания пользовательских интерфейсов. Легко интегрируется в проекты с использованием других JavaScript-библиотек. Может функционировать как веб-фреймворк для разработки одностраничных приложений в реактивном стиле. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами.

Кроме того, современные браузеры также поддерживают и использование GPU API WebGL. WebGL предполагает использование языка шейдеров GLSL, который имеет много общего с C. GLSL (OpenGL Shading Language, Graphics Library Shader Language) — язык высокого уровня для программирования шейдеров. Разработан для выполнения математики, которая обычно требуется для выполнения растеризации графики. Синтаксис языка базируется на языке программирования ANSI C, однако, из-за его специфической направленности, из него были исключены многие возможности, для упрощения языка и повышения производительности. В язык включены дополнительные функции и типы данных, например для работы с векторами и матрицами. Основное преимущество GLSL перед другими шейдерными языками — переносимость кода между платформами и ОС. Язык GLSL используется в OpenGL, в OpenGL ES и WebGL используется язык GLSL ES (OpenGL ES Shading Language).

2 Исходный код

Фрагмент основного кода – файл index.html

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8">
<!—Определение стилей для окна -->
      <style>
        * {
          box-sizing: border-box;
        }
        body {
          padding: 20px;
        }
        .js_controls {
          display: none; /*Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе, none - временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было. Изменить значение и сделать вновь видимым элемент можно с помощью скриптов, обращаясь к свойствам через объектную модель. В этом случае происходит переформатирование данных на странице с учетом вновь добавленного элемента. */
        }
        .js_status {
          display: none;
        }
        .controls--blocked .column *:not(.cancel) {
          opacity: 0.5; /* Полупрозрачность элемента */

          user-select: none;  /*Управляет поведением выделения текста и других элементов на странице, в частности, позволяет запретить выделение текста*/
        }
        .cancel {
          display: none;
        }
        .input--file {
          width: 0.1px;
          height: 0.1px;
          opacity: 0;
          overflow: hidden; /* Свойство overflow управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров, hidden - отображается только область внутри элемента, остальное будет скрыто*/

          position: absolute;
          z-index: -1; /*Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, имитируя тем самым третье измерение, пер-
```

пендикулярное экрану. Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет z-index*/

```
        }
        …
      </style>
    </head>
    <body>
     <div class="container">
      <div class="row">
       <div class="column ta-c">
        <h1 class="ta-c pb-md">Canny Edge Detector</h1>  <!-- Определение заголовка -->
       </div>
      </div>
      <div class="row">
       <div class="column">
        <div class="upload-image ta-c pb-md">
         <input type="file" id="image" class="input--file js_image">
         <label for="image">Choose Image</label> <!-- Лейбл кнопки  "Выбор изображения"-->
        </div>
       </div>
      </div>
      <div class="row">
       <div class="column ta-c">
        <div class="js_controls controls">
         <div class="row">
          <div class="column">
           <input type="text" class="js_lt" placeholder="Lower Treshold (0-1)"> <!—Поле для
ввода "Нижняя точка опоры" -->
           <input type="text" class="js_ut" placeholder="Upper Treshold (0-1)"> <!—Поле для
ввода "Верхняя точка опоры" -->
          </div>
         </div>
         <div class="row">
          <div class="column">
           <button class="button js_submit">Find Edges</button> <!-- Кнопка "Найти ребра" -->

           <button class="button cancel js_cancel">Cancel</button> <!-- Кнопка "Закрыть" -->
…
     </div>
     <script src="./dist/main.js"></script>  <!—Подключение js файлов -->
     <script src="./dist/index.js"></script>
    </body>
</html>
```

Код файла main.js

```
const MAX_PRECISION = false;
const precision = 2;
const gaussMatrix = [
```

```
    [0.0121, 0.0261, 0.0337, 0.0261, 0.0121],
    [0.0261, 0.0561, 0.0724, 0.0561, 0.0261],
    [0.0337, 0.0724, 0.0935, 0.0724, 0.0337],
    [0.0261, 0.0561, 0.0724, 0.0561, 0.0261],
    [0.0121, 0.0261, 0.0337, 0.0261, 0.0121]
]; // Определение матрицы Гаусса
const xMatrix = [[1, 0, -1], [2, 0, -2], [1, 0, -1]];
const yMatrix = [[-1, -2, -1], [0, 0, 0], [1, 2, 1]];
const MAX_IMAGE_HEIGHT = 300;
function curry(f, n) {
    var args = Array.prototype.slice.call(arguments, 0);
    if (typeof n === 'undefined')
        args[1] = f.length;
    if (n === args.length - 2)
        return f.apply(undefined, args.slice(2));
    return function () {
        return curry.apply(undefined, args.concat(Array.prototype.slice.call(arguments, 0)));
    };
}
function loadImage(imageUrl) {
    return new Promise((resolve, reject) => {
        console.time('loadImage');
        const img = new Image();
        img.src = imageUrl;
        img.crossOrigin = 'Anonymous';
        img.onload = function () {
            console.timeEnd('loadImage');
            resolve(img);
        };
    });
}
function readFileAsDataURL(file) {
    return new Promise((resolve, reject) => {
        var reader = new FileReader();
        reader.onloadend = function () {
            resolve(reader.result);
        };
        if (file) {
            reader.readAsDataURL(file);
        }
        else {
            reject('');
        }
    });
}
function _drawImageOnCanvas(canvas, image) {
    canvas.getContext('2d').drawImage(image, 0, 0);
    return image;
}
var drawImageOnCanvas = curry(_drawImageOnCanvas);
function _setCanvasSizeFromImage(canvas, image) {
```

```
    const ratio = image.naturalWidth / image.naturalHeight;
    canvas.style.width = '';
    canvas.getContext('2d').clearRect(0, 0, image.width, image.height);
    canvas.height = image.height;
    canvas.width = image.width;
    return image;
}
var setCanvasSizeFromImage = curry(_setCanvasSizeFromImage);
function _drawBytesOnCanvas(width, height, canvas, bytes) {
    canvas
        .getContext('2d')
        .putImageData(new ImageData(new Uint8ClampedArray(bytes), width, height), 0, 0);
}
var drawBytesOnCanvas = curry(_drawBytesOnCanvas);
function toGrayscale(bytes, width, height) {
    console.time('toGrayscale');
    const grayscale = [];
    for (let i = 0; i < bytes.length; i += 4) {
        var gray = .299 * bytes[i + 2] + .587 * bytes[i + 1] + .114 * bytes[i];
        grayscale.push(gray);
    }
    console.timeEnd('toGrayscale');
    return grayscale;
}
function _toConvolution(width, height, kernel, radius, bytes) {
    console.time('toConvolution');
    const convolution = [];
    let newValue, idxX, idxY, kernx, kerny;
    for (let i = 0; i < width; i++) {
        for (let j = 0; j < height; j++) {
            newValue = 0;
            for (let innerI = i - radius; innerI < i + radius + 1; innerI++) {
                for (let innerJ = j - radius; innerJ < j + radius + 1; innerJ++) {
                    idxX = (innerI + width) % width;
                    idxY = (innerJ + height) % height;
                    kernx = innerI - (i - radius);
                    kerny = innerJ - (j - radius);
                    newValue += bytes[idxY * width + idxX] * kernel[kernx][kerny];
                }
            }
            convolution[j * width + i] = newValue;
        }
    }
    console.time('toConvolution');
    return convolution;
}
const toConvolution = curry(_toConvolution);
/**
 * From image bytes (0 - 255) to values between 0 and 1
 * @param  {Array<number>} bytes
 * @return {Array}      normalized values
```

```
 */
function toNormalized(bytes) {
    console.time('toNormalized');
    const normalized = [];
    for (let i = 0; i < bytes.length; i += 4) {
        normalized.push(bytes[i] / 255);
    }
    console.timeEnd('toNormalized');
    return normalized;
}
/**
 * Из нормализованного массива, который имеет значения от 0 до 1
  * для данных изображения со значениями от 0 до 255
  * @param {Array} нормализован
  * @return {Array} денормализован
 */

function toDenormalized(normalized) {
    console.time('toDenormalized');
    const denormalized = normalized.map(value => value * 255);
    console.timeEnd('toDenormalized');
    return denormalized;
}
function toGradientMagnitude(xDerived, yDerived, width, height, lt = 0, ut = 0) {
    console.time('toGradientMagnitude');
    const gradientMagnitude = [];
    const gradientDirection = [];
    let index;
    let pom;
    for (let y = 0; y < height; y++) {
        for (let x = 0; x < width; x++) {
            index = y * width + x;
            gradientMagnitude[index] = Math.sqrt(xDerived[index] * xDerived[index] +
yDerived[index] * yDerived[index]);
            pom = Math.atan2(xDerived[index], yDerived[index]);
            if ((pom >= -Math.PI / 8 && pom < Math.PI / 8) || (pom <= -7 * Math.PI / 8 && pom > 7 *
Math.PI / 8)) {
                gradientDirection[index] = 0;
            }
            else if ((pom >= Math.PI / 8 && pom < 3 * Math.PI / 8) || (pom <= -5 * Math.PI / 8 &&
pom > -7 * Math.PI / 8)) {
                gradientDirection[index] = Math.PI / 4;
            }
            else if ((pom >= 3 * Math.PI / 8 && pom <= 5 * Math.PI / 8) || (-3 * Math.PI / 8 >= pom
&& pom > -5 * Math.PI / 8)) {
                gradientDirection[index] = Math.PI / 2;
            }
            else if ((pom < -Math.PI / 8 && pom >= -3 * Math.PI / 8) || (pom > 5 * Math.PI / 8 && pom
<= 7 * Math.PI / 8)) {
                gradientDirection[index] = -Math.PI / 4;
            }
```

```
    }
  }
  const max = getMax(gradientMagnitude);
  const gradientMagnitudeCapped = gradientMagnitude.map(x => x / max);
  if (!ut && !lt) {
    let res = getTresholds(gradientMagnitudeCapped);
    ut = res.ut;
    lt = res.lt;
  }
  const gradientMagnitudeLt = gradientMagnitudeCapped.map(value => value < lt ? 0 : value);
  for (var y = 1; y < height - 1; y++) {
    for (var x = 1; x < width - 1; x++) {
      index = y * width + x;
      if (gradientDirection[index] == 0 && (gradientMagnitudeLt[index] <=
gradientMagnitudeLt[y * width + x - 1] || gradientMagnitudeLt[index] <= gradientMagnitudeLt[y *
width + x + 1]))
        gradientMagnitudeLt[index] = 0;
      else if (gradientDirection[index] == Math.PI / 2 && (gradientMagnitudeLt[index] <=
gradientMagnitudeLt[(y - 1) * width + x] || gradientMagnitudeLt[(y + 1) * width + x] >=
gradientMagnitudeLt[index]))
        gradientMagnitudeLt[index] = 0;
      else if (gradientDirection[index] == Math.PI / 4 && (gradientMagnitudeLt[index] <=
gradientMagnitudeLt[(y + 1) * width + x - 1] || gradientMagnitudeLt[index] <=
gradientMagnitudeLt[(y - 1) * width + x + 1]))
        gradientMagnitudeLt[index] = 0;
      else if (gradientDirection[index] == -Math.PI / 4 && (gradientMagnitudeLt[index] <=
gradientMagnitudeLt[(y - 1) * width + x - 1] || gradientMagnitudeLt[index] <=
gradientMagnitudeLt[(y + 1) * width + x + 1]))
        gradientMagnitudeLt[index] = 0;
    }
  }
  for (let y = 2; y < height - 2; y++) {
    for (let x = 2; x < width - 2; x++) {
      if (gradientDirection[y * width + x] == 0)
        if (gradientMagnitudeLt[y * width + x - 2] > gradientMagnitudeLt[y * width + x] ||
gradientMagnitudeLt[y * width + x + 2] > gradientMagnitudeLt[y * width + x])
          gradientMagnitudeLt[y * width + x] = 0;
      if (gradientDirection[y * width + x] == Math.PI / 2)
        if (gradientMagnitudeLt[(y - 2) * width + x] > gradientMagnitudeLt[y * width + x] ||
gradientMagnitudeLt[(y + 2) * width + x] > gradientMagnitudeLt[y * width + x])
          gradientMagnitudeLt[y * width + x] = 0;
      if (gradientDirection[y * width + x] == Math.PI / 4)
        if (gradientMagnitudeLt[(y + 2) * width + x - 2] > gradientMagnitudeLt[y * width + x] ||
gradientMagnitudeLt[(y - 2) * width + x + 2] > gradientMagnitudeLt[y * width + x])
          gradientMagnitudeLt[y * width + x] = 0;
      if (gradientDirection[y * width + x] == -Math.PI / 4)
        if (gradientMagnitudeLt[(y + 2) * width + x + 2] > gradientMagnitudeLt[y * width + x] ||
gradientMagnitudeLt[(y - 2) * width + x - 2] > gradientMagnitudeLt[y * width + x])
          gradientMagnitudeLt[y * width + x] = 0;
    }
  }
```

```
    const gradientMagnitudeUt = gradientMagnitudeLt.map(value => value > ut ? 1 : value);
    // histeresis start
    let pomH = 0;
    let pomOld = -1;
    let pass = 0;
    let nastavi = true;
    let gradientMagnitudeCappedBottom = [];
    while (nastavi) {
        pass = pass + 1;
        pomOld = pomH;
        for (let y = 1; y < height - 1; y++) {
            for (let x = 1; x < width - 1; x++) {
                if (gradientMagnitudeUt[y * width + x] <= ut && gradientMagnitudeUt[y * width +
x] >= lt) {
                    let pom1 = gradientMagnitudeUt[(y - 1) * width + x - 1];
                    let pom2 = gradientMagnitudeUt[(y - 1) * width + x];
                    let pom3 = gradientMagnitudeUt[(y - 1) * width + x + 1];
                    let pom4 = gradientMagnitudeUt[y * width + x - 1];
                    let pom5 = gradientMagnitudeUt[y * width + x + 1];
                    let pom6 = gradientMagnitudeUt[(y + 1) * width + x - 1];
                    let pom7 = gradientMagnitudeUt[(y + 1) * width + x];
                    let pom8 = gradientMagnitudeUt[(y + 1) * width + x + 1];
                    if (pom1 === 1 || pom2 === 1 || pom3 === 1 || pom4 === 1 || pom5 === 1 || pom6 ===
1 || pom7 === 1 || pom8 === 1) {
                        gradientMagnitudeUt[y * width + x] = 1;
                        pomH = pomH + 1;
                    }
                }
            }
        }
        if (MAX_PRECISION) {
            nastavi = pomH != pomOld;
        }
        else {
            nastavi = pass <= precision;
        }
        gradientMagnitudeCappedBottom = gradientMagnitudeUt.map(x => x <= ut ? 0 : x);
    }
    console.timeEnd('toGradientMagnitude');
    return {
        data: gradientMagnitudeCappedBottom,
        threshold: {
            ut: ut,
            lt: lt
        }
    };
}
function getMax(values) {
    return values.reduce((prev, now) => now > prev ? now : prev, -1);
}
function getTresholds(gradientMagnitude) {
```

```
    let sum = 0;
    let count = 0;
    sum = gradientMagnitude.reduce((memo, x) => x + memo, 0);
    count = gradientMagnitude.filter(x => x !== 0).length;
    const ut = sum / count;
    const lt = 0.4 * ut;
    return { ut, lt };
}
/**
* Принимает массив значений (0-255) и возвращает
* расширенный массив [x, x, x, 255] для каждого значения.
* @param {Array} значения
* @return {Array} расширенные значения
 */
function toPixels(values) {
    console.time('toPixels');
    const expanded = [];
    values.forEach(x => {
        expanded.push(x);
        expanded.push(x);
        expanded.push(x);
        expanded.push(255);
    });
    console.timeEnd('toPixels');
    return expanded;
}
```

Основные функции используемые в main.js

Array.prototype.slice.call – используется для того, чтобы из аргументов JavaScript функции "отрезать" первые X значений. Используется вызов через Array.prototype.call(array, params) потому, что arguments - псевдо-массив и не содержит методов прототипа массива. С помощью call мы вызываем нужный метод из прототипа Array с контекстом arguments. Таким образом, такой подход используется когда нужно срезать входящие параметры функции JavaScript.

loadImage(imageUrl) - Загрузка изображения, предоставленные в виде объектов File или Blob, либо по URL-адресу. Извлеките опционально масштабируемый, обрезанный или повернутый элемент HTML img или canvas.

drawImageOnCanvas – Прорисовка изображения на холсте.

setCanvasSizeFromImage – Задание размера холста документа и управление размером выходного изображения.

drawBytesOnCanvas = curry(_drawBytesOnCanvas) – прорисовка границ с помощью метода Curry.

toConvolution - Метод обработки изображений, который изменяет интенсивность пикселя, чтобы отразить интенсивность окружающих пикселей. Обычно свертка используется для создания фильтров изображений. Используя свертку, вы можете получить популярные эффекты изображения, такие как размытие, резкость и обнаружение краев.

const gradientMagnitudeUt – Определение величины градиента.

3 Результаты работы программы

Пользователь может как загружать изображения для их обработки в различных форматах JPEG, png, screenshot. Для запуска приложения необходимо открыть файл index.html в любом из удобных браузеров и в открывшемся окне загрузить фотографию.

Рисунок 5 – Начальное меню

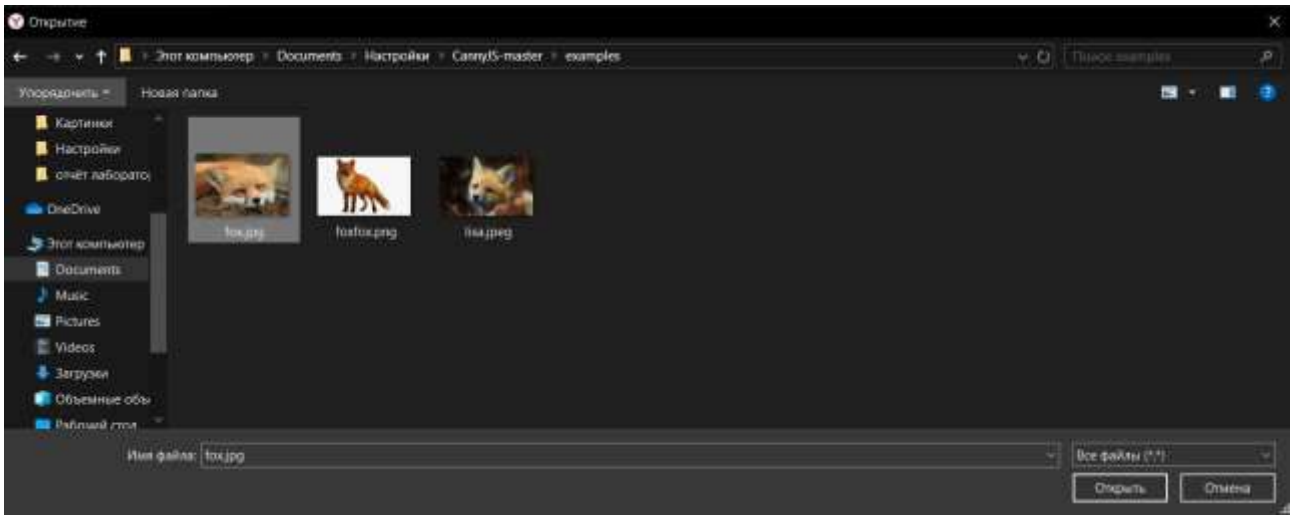

Рисунок 6 – Загрузка исходного изображения

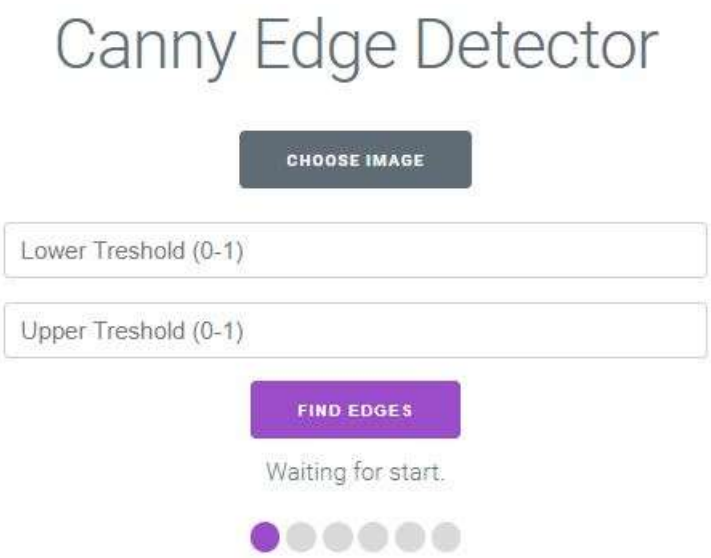

Рисунок 7 – Обнаружение краев Canny

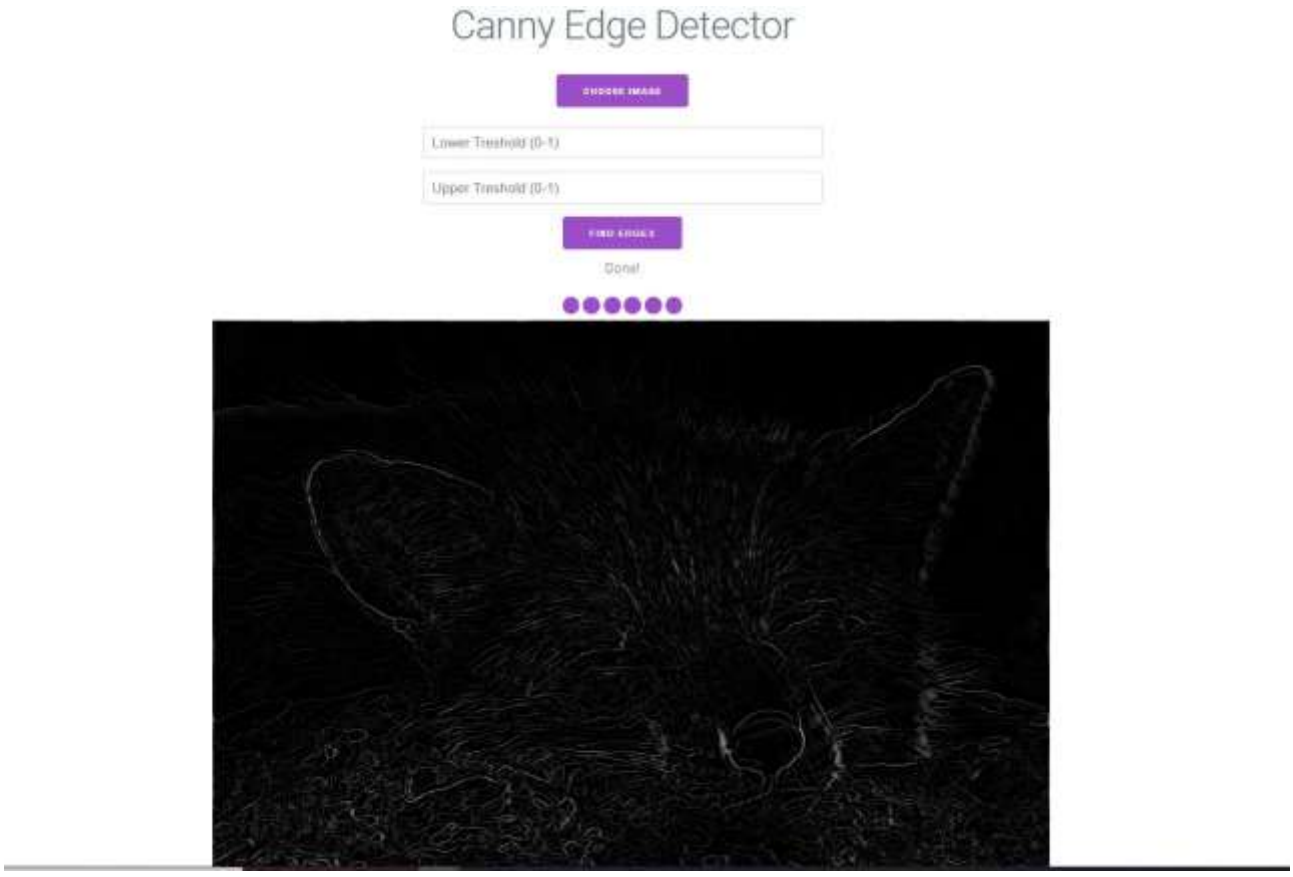

Рисунок 8 – Результат обработки

Вывод

В ходе выполнения данной лабораторной работы была реализована программа определения границ объектов на изображениях при помощи алгоритма Канни. Разработанная программа использует вычислительные мощности графического процессора. Пользователь может загружать собственные изображения и обрабатывать их.

Список литературы

1. Детектор границ Канни [Электронный ресурс]: статья / **– Режим доступа:** https://habr.com/ru/post/114589/, дата обращения: 17.04.2020.

2. К вопросу о выделении границ детектором кенни [Электронный ресурс]: статья / **– Режим доступа: https://sworld.com.ua/konfer26/102.pdf**, дата обращения: 17.04.2020.

3. Детектор ребер Канни [Электронный ресурс]: лекционный материал / **– Режим доступа: https://www.intuit.ru/studies/courses/10622/1106/lecture/18030?page=7**, дата обращения: 17.04.2020.

4. Контурный анализ [Электронный ресурс]: статья / **– Режим доступа:** file:///C:/Users/Алена/Downloads/Контурный%20анализ%20.pdf, дата обращения: 17.02.2020.

5. JavaScript Application framework [Электронный ресурс]: электронный материал / **– Режим доступа:** https://controlsjs.com/#, дата обращения: 17.02.2020.

6. https://github.com/Fire-Phoenix07/Progects