#### Отчет по лабораторным работам по курсу "Машинное зрение и графика"

Студента Горобца Валерия магистранта ВПиИТ 2 курс

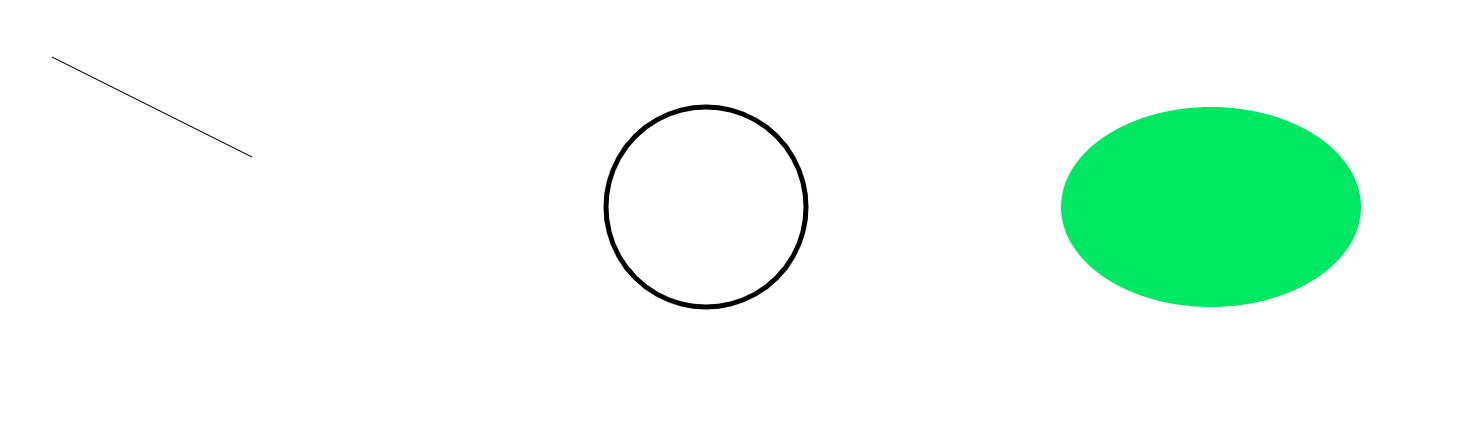
Репозиторий с кодом <https://github.com/valeraGorobets/machineGraphics/tree/master/src>

Лабораторные выполнены на языке Javascript с использованием компилятора Svelte

#### Лаб 1: Рисование отрезков, окружности и эллипса

#### Код: <https://github.com/valeraGorobets/machineGraphics/blob/master/src/Lab1.svelte>

|  |  |  |
| --- | --- | --- |
| Рисование отрезка:  Метод [***lineTo(x, y)***](https://developer.mozilla.org/ru/docs/Web/API/CanvasRenderingContext2D/lineTo)  Рисует линию с текущей позиции до позиции, определенной x и y.  context.beginPath();  context.moveTo(50, 50);  context.lineTo(250, 150);  context.stroke(); | Рисование окружности:  Для рисования дуг и окружностей, используем методы arc().  [***arc(x, y, radius, startAngle, endAngle, anticlockwise***)](https://developer.mozilla.org/ru/docs/Web/API/CanvasRenderingContext2D/arc)  Рисуем дугу с центром в точке (x,y) радиусом r, начиная с угла startAngle и заканчивая в endAngle в направлении против часовой стрелки anticlockwise (по умолчанию по ходу движения часовой стрелки).  context.lineWidth = 5;  context.arc(200, 200, 100, 0, 2 \* Math.PI,true);  context.stroke(); | Рисование эллипса:    Для рисования эллипса, используем метод ellipse ().  Метод ellipse () создает эллиптическую дугу с центром в точке (x, y) с радиусами radiusX и radiusY. Путь начинается в startAngle и заканчивается в endAngle и движется в направлении, заданном против часовой стрелки (по умолчанию по часовой стрелке).  context.ellipse(200, 200, 150, 100, 0, 0, Math.PI\*2);  context.fillStyle = "#00e764";  context.fill(); |

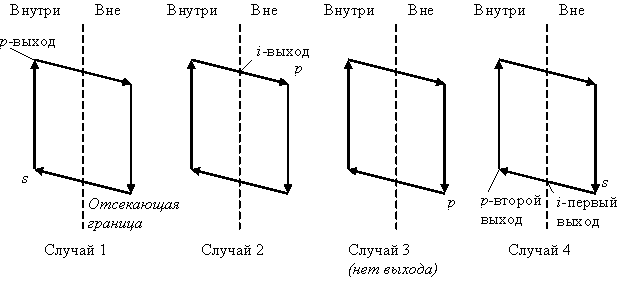


#### Лаб 2: Закрашивание многоугольников, Отсечение

#### Код: <https://github.com/valeraGorobets/machineGraphics/blob/master/src/Lab2.svelte>

|  |  |
| --- | --- |
| Закрашивание многоугольников  Для закрашивания многоугольника вначале по координатам строится фигура и методом ***fillStyle*** закрашивается выбранным цветом. |  |

Отсечение многоугольников проведу  алгоритмом Сазерленда-Ходгмана. На вход алгоритма поступает последовательность вершин многоугольника *V*1*, V*2*, …, Vn*. Ребра многоугольника проходят от *Vi* к *Vi+*1 от *Vn* к *V*1. С помощью алгоритма производится отсечение относительно ребра и выводится другая последовательность вершин, описывающая усеченный многоугольник. Алгоритм «обходит» вокруг многоугольника  от *Vn* к *V*1 и обратно к  *Vn*, проверяя на каждом шаге соотношение между последовательными вершинами и отсекающей границей. Необходимо проанализировать четыре случая:



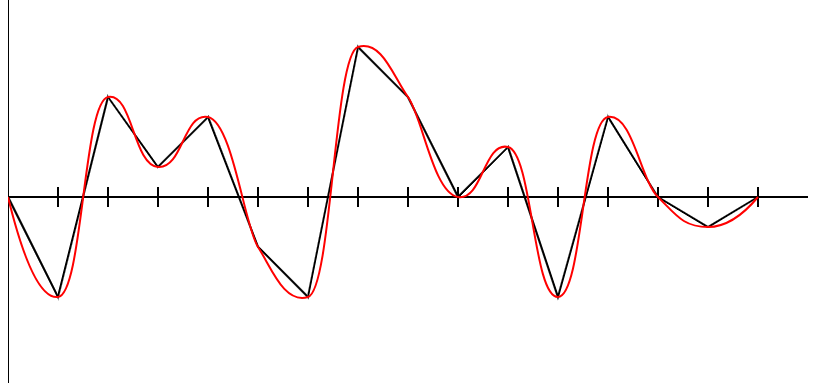
Ребро многоугольника, соединяющее вершину s с вершиной p. В первом случае ребро полностью лежит внутри отсекающей границы, к выходному списку добавляется вершина p. Во втором случае в качестве вершин выводится точка пересечения i, поскольку ребро пересекает границу, а начальная точка была выведена при анализе первого случая. В третьем случае обе вершины находятся за пределами границы и ни одна из них не выводится. В четвертом случае к выходному списку добавляется и точка пересечения i, и вершина p.

|  |  |  |
| --- | --- | --- |
| Многоугольник | Отсекающяя граница | Отсечение |

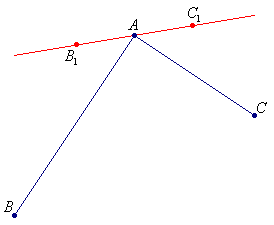


#### Лаб 3: Аппроксимация кривых с помощью кривых Безье

#### Код: <https://github.com/valeraGorobets/machineGraphics/blob/master/src/Lab3.svelte>



Кривая Безье выходит из первой точки касательно к прямой начальная\_точка-первая\_опорная\_точка. Аналогично и в конце — кривая заходит касательно прямой последняя\_опорная\_точка-конечная\_точка. Таким образом получается, что если у нас одна кривая заканчивается в точке А и зашла касательно к прямой а, а другая кривая выходит из этой точки А касательно к той-же прямой а, то этот переход между двумя кривыми Безье у нас получится плавным. Получается, что у нас опорные точки слева и справа относительно точки А должны лежать на одной прямой, то есть эта прямая должна быть такой, чтобы *∠BAB1=∠CAC1*, где точки *B1* и *C1* — опорные.



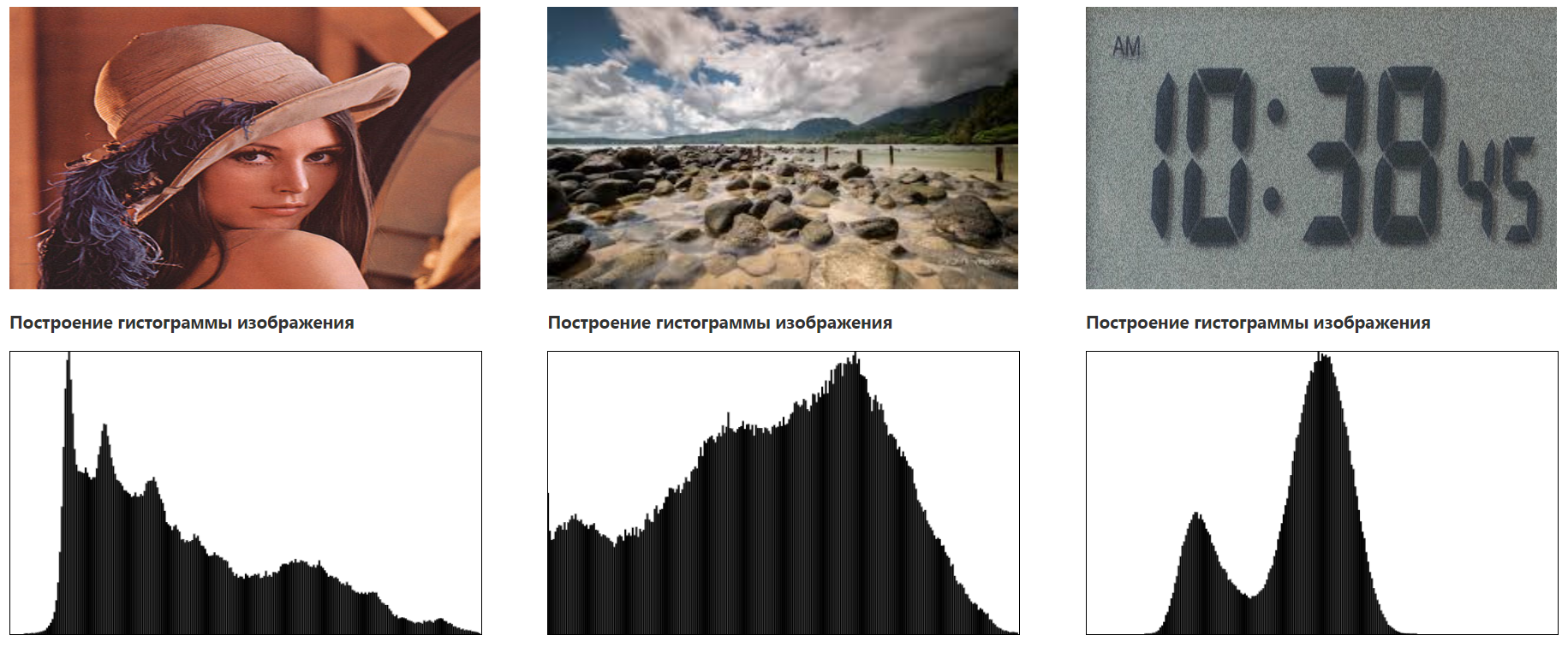
Расстояние от точки *А* до точек *B1* и *C1* равно половине шага по X между точками *B* и *A*, *A* и *C*. Таким образом получается, что задача сводится к поиску прямой (*B1C1*) и, собственно, опорных точек B1 и C1, по которым мы потом будем строить кривые Безье.

С помощью функций [quadraticCurveTo()](http://www.html5canvastutorials.com/tutorials/html5-canvas-quadratic-curves/) и [bezierCurveTo()](http://www.html5canvastutorials.com/tutorials/html5-canvas-bezier-curves/) по найденым опорным точкам для каждой тоски графика рисуем кривые Безье.



#### Лаб 4: Построение гистограммы изображения, Преобразование цветного изображения в полутоновое, Бинаризация полутоновых изображений, Устранение шумов на бинарном изображении, Устранение шумов на полутоновом изображении (усредняющий и медианный фильтр), Выделение границ объектов на бинарном изображении, Выделение границ объектов на полутоновом изображении

#### Код: <https://github.com/valeraGorobets/machineGraphics/blob/master/src/Grayscale.svelte>



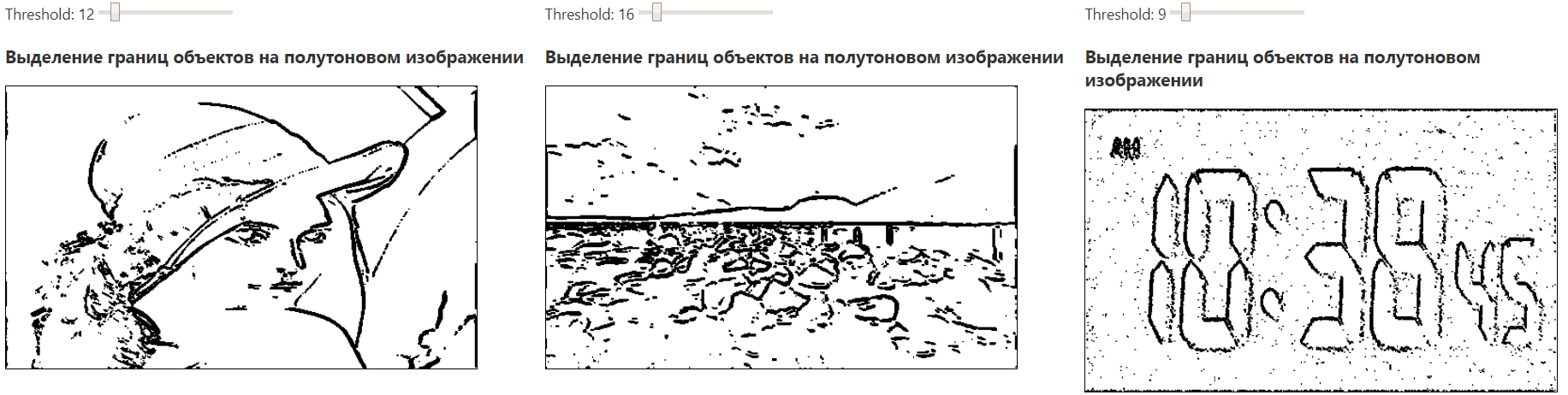
|  |  |
| --- | --- |
|  | Гистограмма – это график распределения тонов на изображении. На горизонтальной оси - шкала яркостей тонов от белого до черного, на вертикальной оси - число пикселей заданной яркости. |
|  |  |



|  |  |
| --- | --- |
|  | Полутоновое изображение получено заменой значений красного, зеленого и синего пикселя на усредненное их значение. |







|  |  |
| --- | --- |
|  | Рассматривается пиксель с окрестностью 3х3, если разница текущего значения пикселя с соседом больше порогового значения, то считаем что граница найдена и заносим черный пиксель в результирующий массив, иначе – белый. |



|  |  |
| --- | --- |
|  | Проходимся по каждому пикселю полутонового изображения, если его значение больше порогового, то перезаписываем его белым пикселем, иначе - черным. Для вычисления порога бинаризации использовался Метод Оцу: |







Алгоритм медианного фильтра

1. Вычислить

* M1=median(C, Cnorth, Csouth);
* M2=median(C, Ceast, Cwest);
* M3=median(C, Cne, Csw);
* M4=median(C, Cnw, Csw);

здесь Cnort, Cne,...Cnw — восемь соседних пикселей из окрестности 3x3, C — центральный пиксель

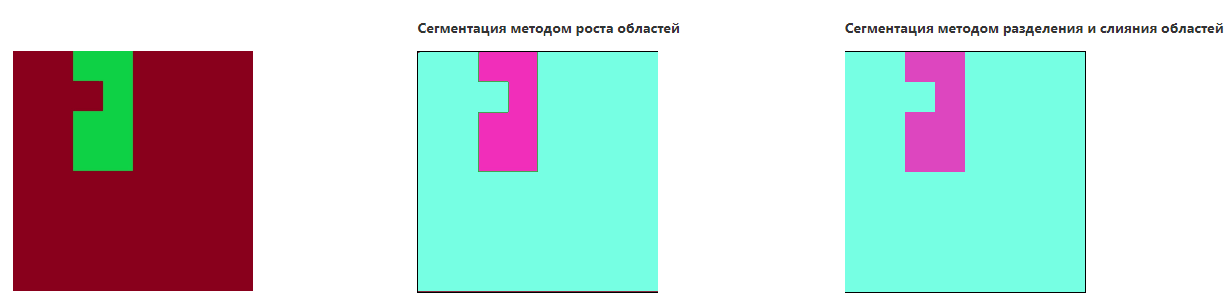
1. Вычислить
   * Ma=median(C, M1, M2);
   * Mb=median(C, M3, M4);
2. вычислить Csmooth=median(C, Ma, Mb);
3. заменить C на Csmooth.
4. отсортировать пиксели из окрестности 3x3 по возрастанию, P[1]...P[9].
5. Если центральный пиксель равен P[1] — заменить его на P[2], если центральный пиксель равен P[9] — заменить его на P[8], в других случаях оставить без изменения.



|  |  |
| --- | --- |
|  | Рассматривается пиксель с окрестностью 3х3, если среди соседей есть пиксель отличного цвета от рассматриваемого, то считаем что граница найдена и заносим черный пиксель в результирующий массив, иначе – белый. |

#### Лаб 5 Сегментация изображений. Метод роста областей. Разделение и слияние областей.

#### Код: <https://github.com/valeraGorobets/machineGraphics/blob/master/src/Lab5.svelte>



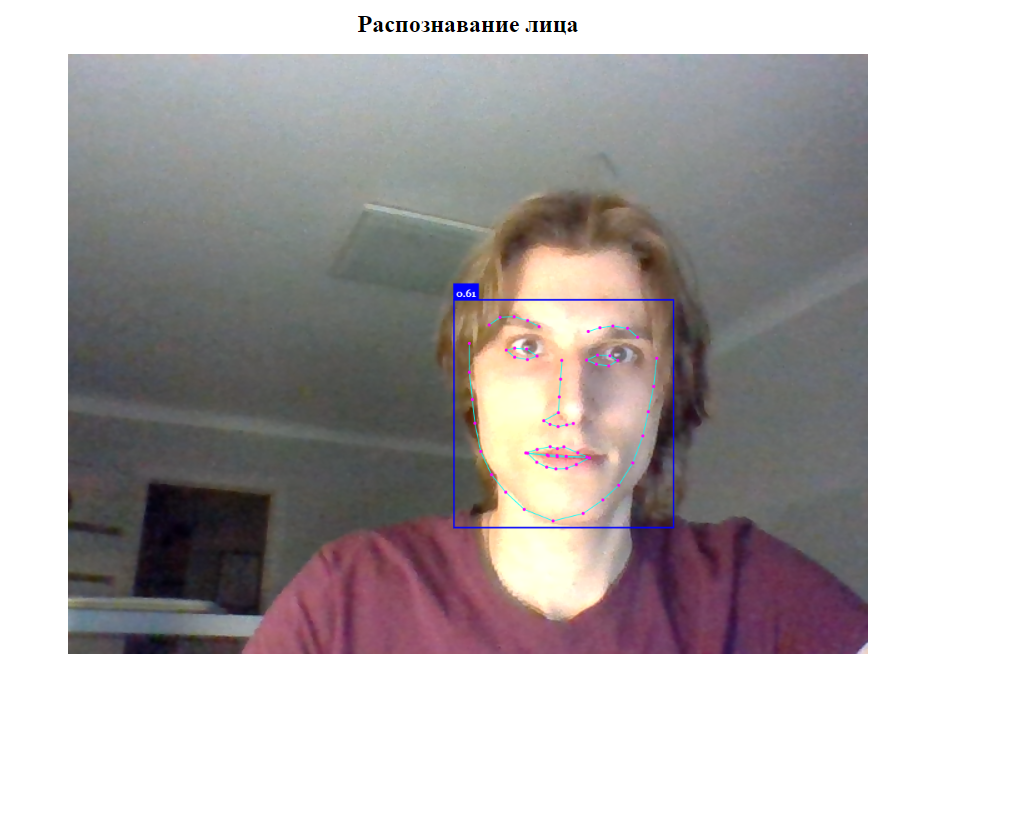
|  |  |
| --- | --- |
|  | Когда рост одной области закончен, выбирается другой новый пиксель, который еще не принадлежит ни одной области и начинаем снова. Процесс продолжается до тех пор, пока все пиксели изображения не будут принадлежать некоторой области. |

|  |  |
| --- | --- |
|  | Большие неоднородные области разбиваются на более мелкие области, которые могут быть однородными. Область изображения делится на 4 области. Затем каждая делится еще на 4.  Этот процесс, продолжается до достижения однородности области.  Соседние области сравниваются и объединяются, если они достаточно близки по некоторому свойству. |

#### Лаб 6: Распознавание объектов изображений.

#### Код: <https://github.com/valeraGorobets/machineGraphics/tree/master/src/lab6>

Для изучения была выбрана тема распознования лиц в реальном времени



|  |  |
| --- | --- |
|  | [face-api.js](https://github.com/justadudewhohacks/face-api.js) - API распознавания лиц JavaScript для браузера и nodejs, реализованный поверх ядра [tenorflow.js](https://github.com/tensorflow/tfjs) Раз в 10 мс кадр с фронтальной камеры проходит процедуру распознавания лица и нанесение маски поверх видео потока. |