Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Распознавание отдельных видов биологических тканей по результатам компьютерной томографии

Студент группы ИУ9-51Б _____ (Бакланова А. Д.)
(Подпись, дата)

Руководитель курсового проекта _____ (Домрачева А. Б.)
(Подпись, дата)

Москва, 2019 г.

СОДЕРЖАНИЕ

Введение

За последние годы в области науки и техники было совершено множество открытий. Прогресс не стоит на месте, а продолжает стремительно расти, давая людям доступ к изобретению нового. Так, наука и техника стали неотъемлемой частью медицины. Практические все ее области теперь прочно связаны с достижениями науки, что дает возможность точно диагностировать большинство болезней, следить за ними и лечить их. Одним из самых важных методов диагностики является томография. Под томографией понимается метод исследования, при котором используют рентген [1]. Благодаря ему происходит послойная съемка внутренней структуры объекта, а затем к полученным данным применяются методы обработки изображений, после чего на выходе получается удобный для чтения врачом снимок, где четко видны органы, скелет, различные ткани и, если таковые имеются, опухоли, раны и прочие отклонения.

Такая обработка изображений для обнаружения, слежения и классификации объектов относится к области компьютерного зрения. Методы этой области позволяют улучшать производительные процессы, используются для видеонаблюдения и ведения статистик, где нужно уметь различать объекты, организовывать информацию и выдавать ее в удобном и понятном для человека виде. Также компьютерное зрение дает возможность технике по-новому взаимодействовать с реальными миром. Многие люди трудятся над развитием этой области, так как за ней ожидается великое будущее. Также, благодаря развитию алгоритмов обработки изображений, сейчас существует нейронная сеть, которую обучают распознавать болезни, а в частности рак и любые, даже самые ранние его стадии. И чем точнее и лучше будут работать обработчики

снимков, тем успешнее нейронная сеть будет справляться с диагностикой опухолей и других заболеваний, что очень важно для здоровья людей, ведь благодаря этому будут спасены многие жизни. На начальных стадиях лечить опасные заболевания гораздо проще и безопаснее для жизни.

Итак, томография — это один из самых эффективных методов диагностики, который к тому же дает достаточно много информации. А когда речь заходит о медицине и лечении болезней, для которых требуется хирургическое вмешательство, точность и полнота информации о том, что именно надо извлечь — самые важные части. Только с помощью компьютерной томографии можно диагностировать потенциально опасные для жизни состояния: кровоизлияние, тромбы или рак на начальной стадии. Хорошая диагностика необходима для спасения жизней, поэтому тема курсовой работы актуальна в наши дни.

Целью данной курсовой работы является изучение метода распознавания отдельных видов биологических тканей по результатам компьютерной томографии.

Планируется рассмотреть самый известный и широко распространенный алгоритм детектора границ — алгоритм Кэнни и протестировать его работу на рентгеновских снимках.

1. Описание предметной области

Томография (др.-греч. τομή — сечение) — метод послойного исследования внутренней структуры объекта, который при этом не предполагает каких-либо его разрушений, путем многократного просвечивания под различными ракурсами (Рисунок 1). Сначала проводится совокупность процессов записи данных, затем к ним применяются алгоритмы математической обработки. Томографическим снимком назовем результат записи данных, полученных в результате компьютерной томографии. Именно такие снимки будут использоваться для анализа тканей и выделения границ.

Существует множество различных методов, позволяющих выделить границы, отличаются они применяемыми фильтрами сглаживания, типами фильтров для вычисления градиентов. Однако детектор границ Кэнни до сих пор считается одним из лучших и точных детекторов и трудно найти другой алгоритм, который работал бы существенно лучше.

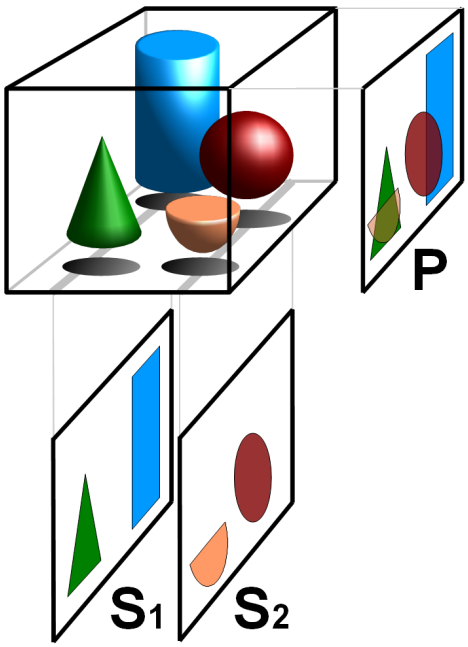


Рисунок 1. Проекции трехмерных объектов на разных уровнях глубины.

2. Чтение изображений в формате bmp и работа с ними

Для реализации алгоритма Кэнни очень важно полное понимание представления изображения. Мной была выбрана библиотека C++ Bitmap Library by Arash Partow [2]. Данная библиотека очень удобна тем, что изображение в ней представляется в виде матрицы, в каждой ячейке которой лежит пиксель типа rgb_t. И от него можно взять нужный цвет (Рисунок 2). Также у считанного изображения можно узнать размеры с помощью функций height() и width(). Менять значение пикселя через функцию set_pixel(x, y, red, green, blue).

```
40      for (int y = 0; y < height; ++y)
41      {
42         for (int x = 0; x < width; ++x)
43         {
44            rgb_t colour;
45             image.get_pixel(x, y, colour);
46             image.set_pixel(x, y, '0', '0', colour.blue);
47         }
48      }
```

Рисунок 2. Демонстрация самых частоиспользуемых функций библиотеки.

3. Алгоритм детектора границ Кэнни

3.1. Алгоритм Кэнни и его шаги

Детектор границ Кэнни до сих пор является одним из лучших детекторов в настоящее время, несмотря на то, что алгоритм был разработан в самом начале развития систем компьютерного зрения, в 1986 году. Джон Кэнни изучил проблему получения фильтра с математической точки зрения, и оптимизировал критерии выделения границ, а также показал, что фильтр может быть хорошо приближен первой производной Гауссианы. Также алгоритм был создан таким образом, что он содержит в себе несколько важных ступеней, что делает его настолько точным. Вместе с этим, было введено понятие подавления немаксимумов, которое стало одной из ступеней алгоритма. Оно означает, что пикселями границ объявляются точки, в которых достигается локальный максимум градиента в направлении вектора градиента.

Целью Джона Кэнни было разработать алгоритм, который был бы оптимальный в плане поиска границ, а также немаловажным была возможность удовлетворять следующим критериям:

1) хорошее обнаружение или повышение отношения сигнал/шум;
2) хорошая локализация — под этим имеется ввиду правильное определение границы;
3) единственный отклик на одну границу.

Все эти критерии в совокупности означают, что детектор должен реагировать на границы, но при этом игнорировать ложные, точно определять линию границы без её фрагментирования и реагировать на

каждую границу только один раз, что позволяет избежать восприятия широких полос изменения яркости как совокупность границ.

Алгоритм Кэнни можно разделить на следующие ступени:

1) Сглаживание изображения — размытие изображения для удаления шума;
2) Поиск градиентов — границы отмечаются там, где градиент изображения приобретает максимальное значение;
3) Подавление немаксимумов — 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.

3.2. Преобразование изображения в оттенки серого

Перед применением детектора важно преобразовать цветное изображение в оттенки серого. Обычно этот этап характерен для многих методов обработки изображений, так как благодаря ему происходит уменьшение вычислительных затрат.

Для выполнения преобразования необходимо использовать коэффициенты перевода (1), которые определяются особенностями человеческого восприятия.

$$Y = 0.299R + 0.587G + 0.114B \qquad (1)$$

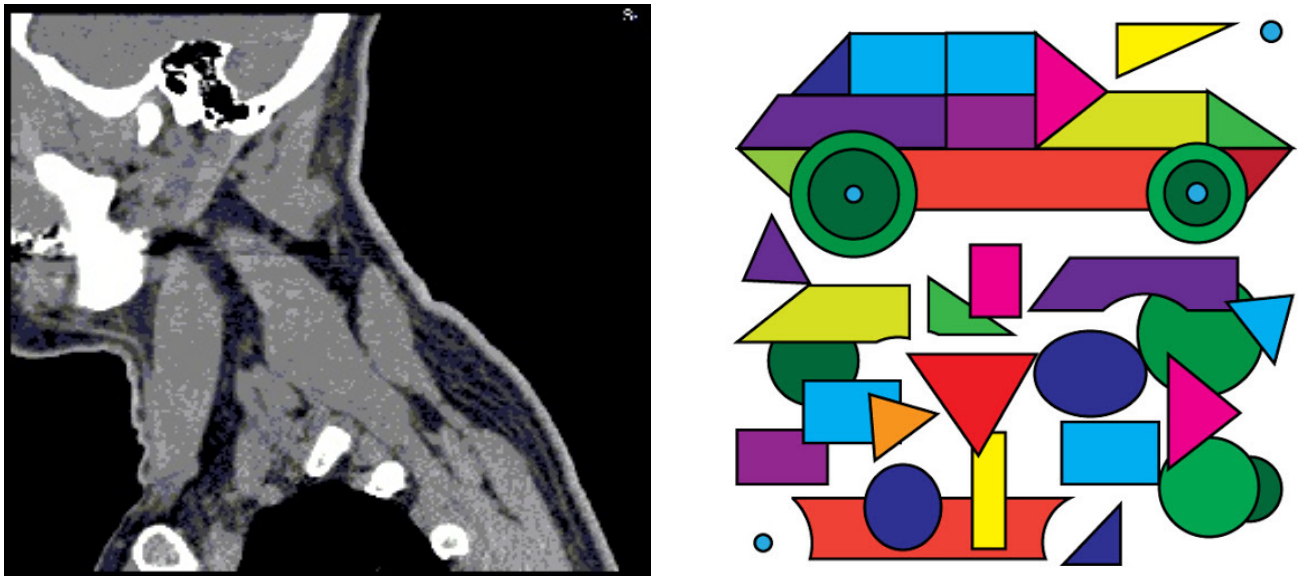
Пользуясь документацией библиотеки [2], была написана следующая функция преобразования изображения в оттенки серого (Рисунок 3).

```
166 bitmap_image grayscale(bitmap_image image, int height, int width) {
167     for (int y = 0; y < height; ++y)
168     {
169        for (int x = 0; x < width; ++x)
170        {
171           rgb_t colour;
172            image.get_pixel(x, y, colour);
173            float avg;
174            avg = 0.3*colour.red + 0.11*colour.blue + 0.59*colour.green;
175            //cout << avg << " " << endl;
176            image.set_pixel(x, y, avg, avg, avg);
177        }
178     }
179
          image.save_image("/Users/anemone/Desktop/hell1/canny/input_images/img_step_0
          .bmp");
180
181     return image;
182 }
```

Рисунок 3. Реализация алгоритма преобразования изображения в оттенки серого.

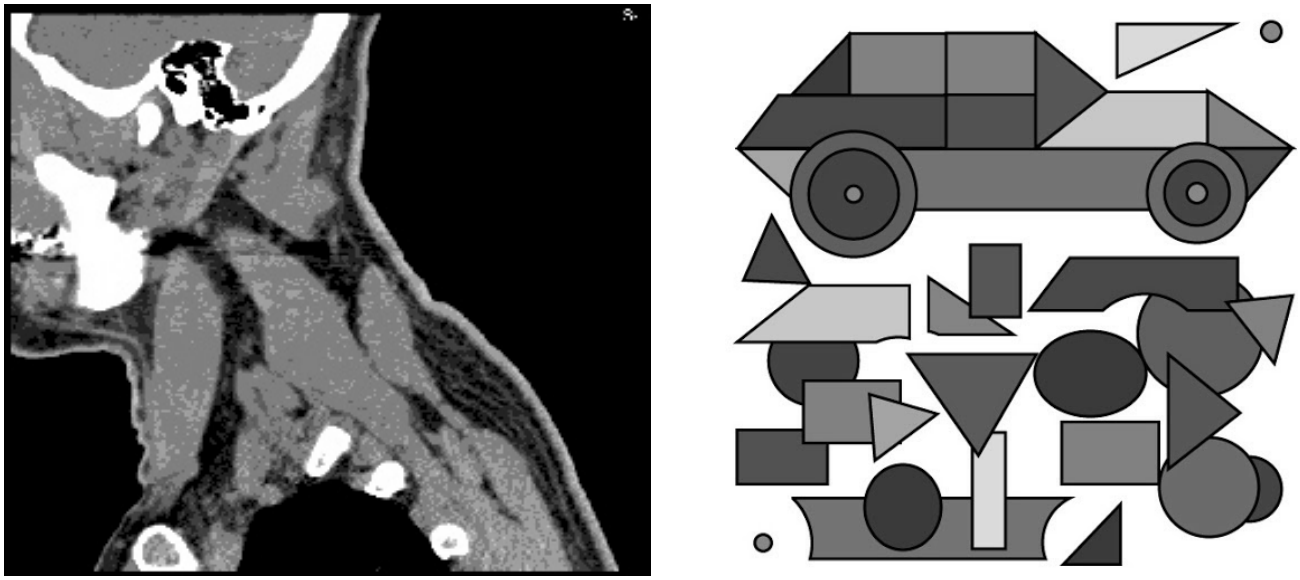
В аргументы функции поступают: изображение, его высота и ширина. Далее, после применения формулы (1), на выход имеется преобразованное изображение, которое сохраняется в папке проекта.

Тестирование функции: на вход имеются изображения: Рисунок 4 а) и б), а на выход получаются: Рисунок 5 а) и б).



а) б)

Рисунок 4. а) Томографический снимок, б) Геометрические фигуры



а) *б)*

Рисунок 5. а), б) Результат перевода изображений в оттенки серого

3.3. Сглаживание

Изображение, переведенное в оттенки серого, впоследствии сглаживается за счет использования специального фильтра. Это является первым шагов алгоритма детектора границ Кэнни. Фильтр обнаруживает и устраняет найденные разрывы, применяя перемещаемую по изображению маску, которую зачастую также называют ядром, представляющим из себя квадратную матрицу. Элементами такой матрицы будут являться пиксели, над которыми в момент времени находится маска. Согласно значениям яркости этих пикселей, изменяется яркость пикселя под центром маски.

Обычно матрица заполняется по Гауссовому закону, при σ = 1.4 и размере матрицы 5, справедлива формула (2) [3]:

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \quad (2)$$

Или же матрицу можно составить с помощью формулы (3):

$$G(x, y, \sigma) = \frac{1}{2\pi \cdot \sigma^2} \cdot e^{\frac{-(x^2+y^2)}{2\sigma^2}} \quad (3)$$

От размера матрицы и сигмы зависит сила размытия. Также у всех матричных фильтров есть проблема с граничными условиями. У верхнего пикселя не существует «соседа» справа от него, и это значит, что становится не на что умножать коэффициент матрицы (Рисунок 6).

Существует 2 способа решения этой проблемы:

1) Применить фильтр только к «окну» изображения. Правда при

Рисунок 6. Демонстрация проблемы граничных условий [6]

таком методе недостатком будет то, что изображение будет иметь по периметру едва ли заметную рамку неразмытых пикселей (Рисунок 7). Но такой метод достаточно просто и быстро пишется.

Рисунок 7. Демонстрация рамки неразмытых пикселей [6]

2) Создание промежуточного изображения. Идея метода в том, чтобы создать временное изображение, в центр которого копируется исходная картинка, а края заполняются крайними пикселями изображения. Размытие применяется к временному изображению, а потом из него извлекается результат (Рисунок 8). Данный метод избегает недостатков с качеством и не имеет рамки, однако происходит нагрузка засчет лишних вычислений.

Реализация подсчета матрицы фильтра Гаусса, используя формулу (3) для матрицы размера 5 и $\sigma = 1.5$ изображена на Рисунке 9.

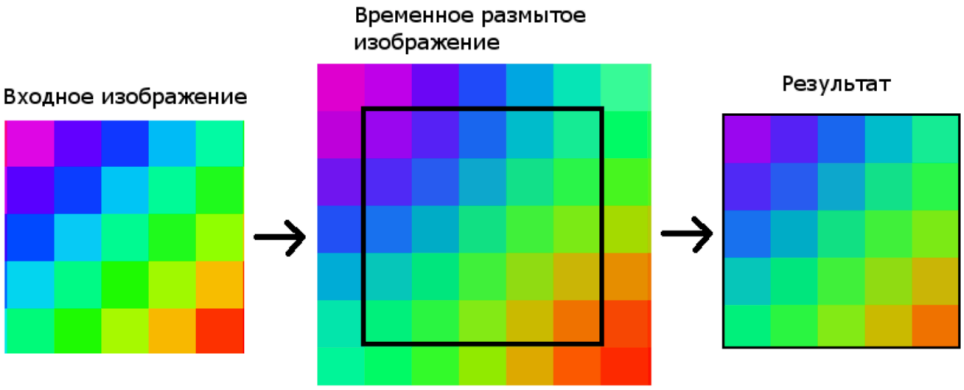


Рисунок 8. Второй метод решения проблемы граничных пикселей [6]

```
89  void gaussian_kernel(float **gauss, float sigma) {
90      float sum = 0.0;
91
92      for (int x = 0; x < 5; x++) {
93          for (int y = 0; y < 5; y++) {
94              gauss[x][y] = gaussian_model(x-5/2, y-5/2, sigma);
95              sum += gauss[x][y];
96          }
97      }
98
99      for (int i = 0; i < 5; i++) {
100         for (int j = 0; j < 5; j++) {
101             gauss[i][j] /= sum;
102             cout << gauss[i][j] << " ";
103         }
104         cout << endl;
105     }
106 }
107
108 float gaussian_model(float x, float y, float sigma) {
109     return (1/ (2*M_PI*sigma*sigma) * exp(-(x*x + y*y) / (2*sigma*sigma)));
110 }
```

Рисунок 9. Реализация подсчета матрицы фильтра Гаусса

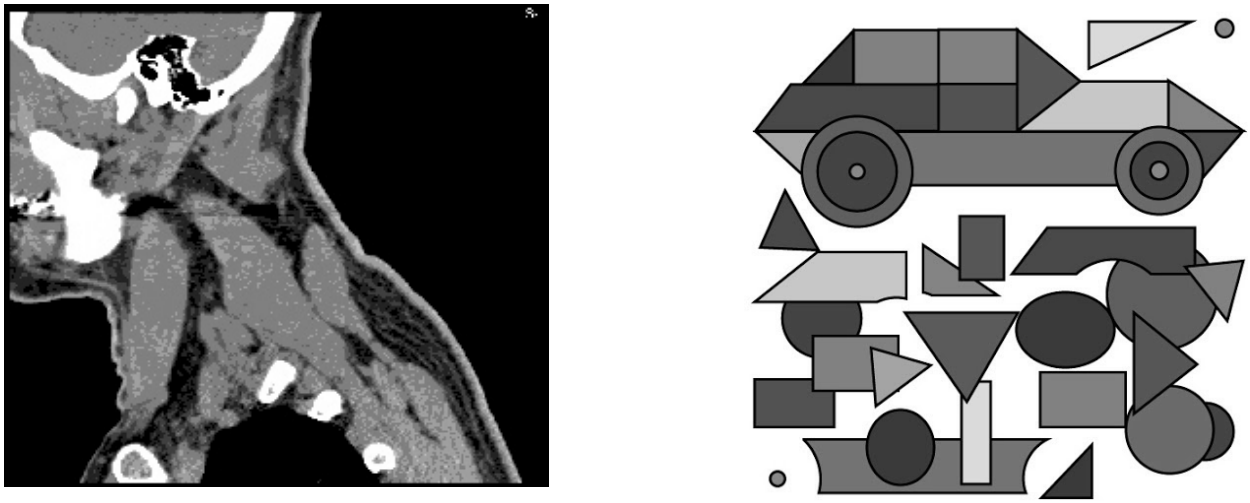
Реализация самого алгоритма применения фильтра Гаусса с решением проблемы с граничными пикселями первым способом изображена на Рисунке 10.

```
182 bitmap_image applyFilter(bitmap_image image, int height, int width, float **gauss) {
183     cout << height << " " << width << endl;
184 
185     for (int x=2; x<width-5; x++) {
186         for (int y=2; y<height-5; y++) {
187             //cout << " " << y << " " << endl;
188             float **distributedColourRed = new float*[5];
189             float **distributedColourGreen = new float*[5];
190             float **distributedColourBlue = new float*[5];
191 
192             for (int i = 0; i < 5; i++) {
193                 distributedColourRed[i] = new float[5];
194                 distributedColourGreen[i] = new float[5];
195                 distributedColourBlue[i] = new float[5];
196             }
197 
198             for (int gaussX=0; gaussX<5; gaussX++) {
199                 for (int gaussY=0; gaussY<5; gaussY++) {
200 
201                     int sampleX = x+gaussX;
202                     int sampleY = y+gaussY;
203 
204                     float currentGauss = gauss[gaussX][gaussY];
205 
206                     rgb_t colour;
207 
208                     image.get_pixel(sampleX, sampleY, colour);
209                     distributedColourRed[gaussX][gaussY] = currentGauss*colour.red;
210                     distributedColourGreen[gaussX][gaussY] = currentGauss*colour.green;
211                     distributedColourBlue[gaussX][gaussY] = currentGauss*colour.blue;
212                 }
213             }
214             image.set_pixel(x, y, getFilteredValue(distributedColourRed),
                 getFilteredValue(distributedColourGreen), getFilteredValue(distributedColourBlue));
215         }
216     }
217     image.save_image("/Users/anemone/Desktop/hell1/canny/input_images/img_step_1.bmp");
218     return image;
219 }
220 
221 float getFilteredValue(float **filteredColor) {
222     float sum = 0;
223 
224     for (int i=0; i<5; i++) {
225         for (int j=0; j<5; j++) {
226             sum += filteredColor[i][j];
227         }
228     }
229     return sum;
230 }
```

Рисунок 10. Реализация алгоритма размытия изображения фильтром Гаусса.

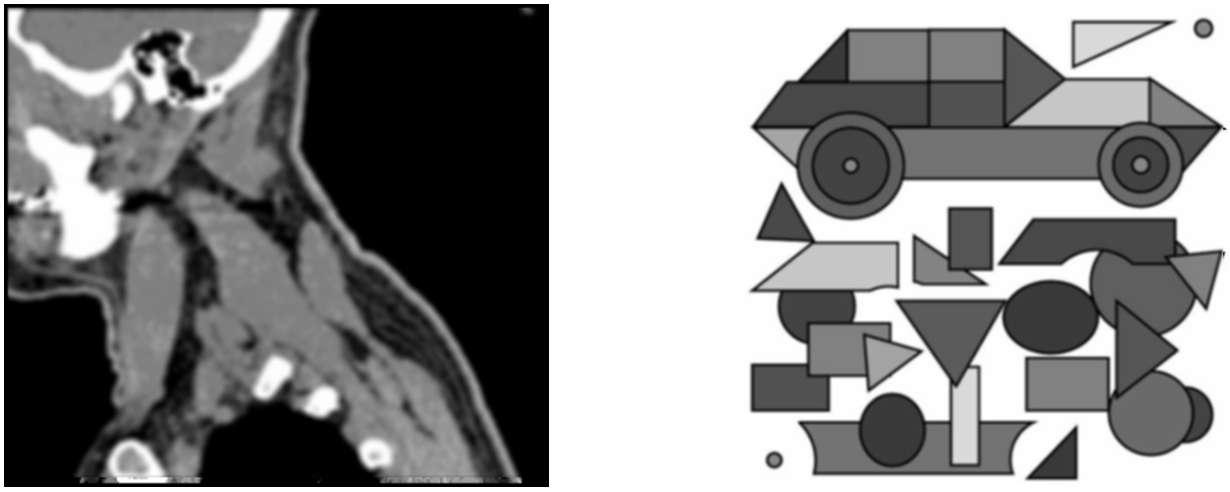
В аргументы функции поступают: изображение, его высота, ширина и матрица фильтра Гаусса. Далее, после подсчета матрицы с помощью формулы (3) с последующим применением фильтра к изображению, на выход имеется преобразованное изображение, которое сохраняется в папке проекта.

Тестирование функции: на вход имеются изображения: Рисунок 5 а) и б), которые получились в результате преобразования изображения в оттенки серого, а на выход получаются: Рисунок 11 а) и б).



а) *б)*

Рисунок 5. а), б) Изображения в оттенках серого



а) *б)*

Рисунок 11. а), б) Изображения размыты фильтром Гаусса

3.4. Поиск градиентов

Очень часто в алгоритмах выделения границ используется оператор Собеля. Это дискретный дифференциальный оператор, вычисляющий приближенное значение градиента яркости изображения. Результатом применения оператора Собеля для каждой точки изображения будет являться вектор градиента в этой точке. Алгоритм основан на свёртке изображения небольшими целочисленными фильтрами в вертикальном и горизонтальном направлениях (формулы (4) и (5)). Градиент определяет, как быстро изменяется яркость изображения в каждой точке, что дает возможность определить границу и ее ориентацию. Он вычисляется по формуле (6), где GX и GY — применение горизонтального и вертикального фильтров к изображению (Рисунок 12).

$$MG_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad (4)$$

$$MG_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad (5)$$

$$G = \sqrt{GX^2 + GY^2} \quad (6)$$

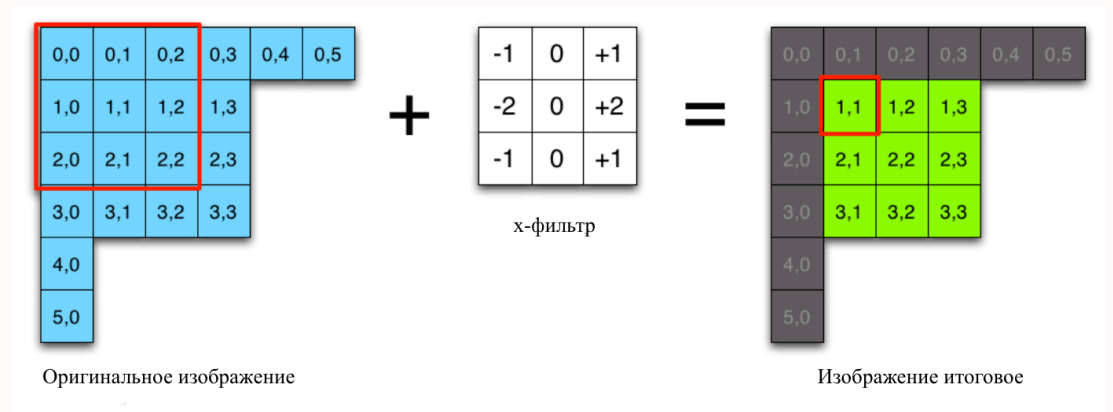


Рисунок 12. Визуализация алгоритма применения оператора Собеля

Реализация применения оператора Собеля в горизонтальном и вертикальном направлениях изображена на рисунке 13.

Реализация самого алгоритма подсчета градиентов с помощью оператора Собеля изображена на Рисунке 14.

```
296 bitmap_image convolve(bitmap_image image, float **g) {
297     int height = image.height();
298     int width = image.width();
299 
300     for (int i=1; i<width-2; i++) {
301         for (int j=1; j<height-2; j++) {
302             float sumR = 0;
303             float sumG = 0;
304             float sumB = 0;
305             for (int k=0; k<3; k++) {
306                 for (int l=0; l<3; l++) {
307 
308                     rgb_t colour;
309                     image.get_pixel(i+k, j+l, colour);
310 
311                     float currentG = g[k][l];
312                     sumR += colour.red * currentG;
313                     sumG += colour.green * currentG;
314                     sumB += colour.blue * currentG;
315                 }
316             }
317             sumR = abs(sumR);
318             sumG = abs(sumG);
319             sumB = abs(sumB);
320             image.set_pixel(i, j, sumR, sumG, sumB);
321         }
322     }
323 
324     return image;
325 }
```

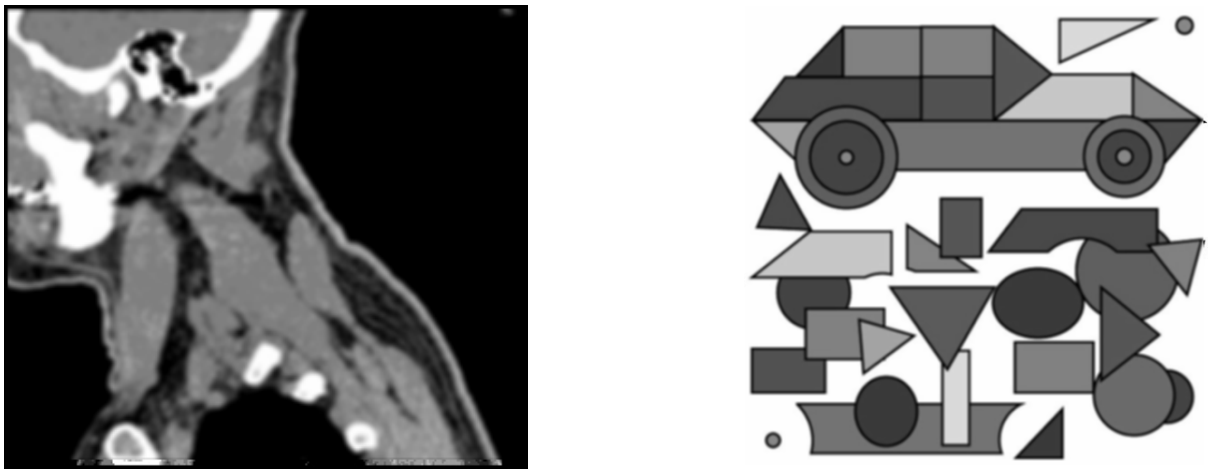
Рисунок 13. Реализация применения оператора Собеля

```
232 bitmap_image gradientComputing(bitmap_image image, int height, int width) {
233     float **gx = new float*[3];
234     for (int i = 0; i < 3; i++) {
235         gx[i] = new float[3];
236     }
237     gx[0][0] = 1; gx[0][1] = 0; gx[0][2] = -1;
238     gx[1][0] = 2; gx[1][1] = 0; gx[1][2] = -2;
239     gx[2][0] = 1; gx[2][1] = 0; gx[2][2] = -1;
240 
241     float **gy = new float*[3];
242     for (int i = 0; i < 3; i++) {
243         gy[i] = new float[3];
244     }
245     gy[0][0] = 1; gy[0][1] = 2; gy[0][2] = 1;
246     gy[1][0] = 0; gy[1][1] = 0; gy[1][2] = 0;
247     gy[2][0] = -1; gy[2][1] = -2; gy[2][2] = -1;
248 
249     bitmap_image img_x, img_y;
250     img_x = convolve(image, gx);
251     img_y = convolve(image, gy);
252 
253     float magR, magG, magB;
254     float maxValR, maxValG, maxValB;
255     maxValR=maxValG=maxValB = 0;
256     for (int i=0; i<width; i++) {
257         for (int j=0; j<height; j++) {
258             rgb_t colourx;
259             img_x.get_pixel(i, j, colourx);
260 
261             rgb_t coloury;
262             img_y.get_pixel(i, j, coloury);
263             
264             magR = sqrt((colourx.red*colourx.red) + (coloury.red*coloury.red));
265             magG = sqrt((colourx.green*colourx.green) + (coloury.green*coloury.green));
266             magB = sqrt((colourx.blue*colourx.blue) + (coloury.blue*coloury.blue));
267             if (magR > maxValR) {
268                 maxValR = magR;
269             }
270             if (magG > maxValG) {
271                 maxValG = magG;
272             }
273             if (magB > maxValB) {
274                 maxValB = magB;
275             }
276             image.set_pixel(i, j, magR, magG, magB);
277         }
278     }
279 
280     for (int i = 2; i < width-2; i++) {
281         for (int j = 2; j < height-2; j++) {
282             //mag[i][j] = mag[i][j] / maxVal * 255;
283             rgb_t colour;
284             image.get_pixel(i, j, colour);
285 
286             image.set_pixel(i, j, colour.red/maxValR *255, colour.green/maxValG *255, colour.blue/maxValB *255);
287         }
288     }
289     image.save_image("/Users/anemone/Desktop/hell1/canny/input_images/img_step_2.bmp");
290     return image;
291 }
```

Рисунок 14. Реализация алгоритма подсчета градиентов

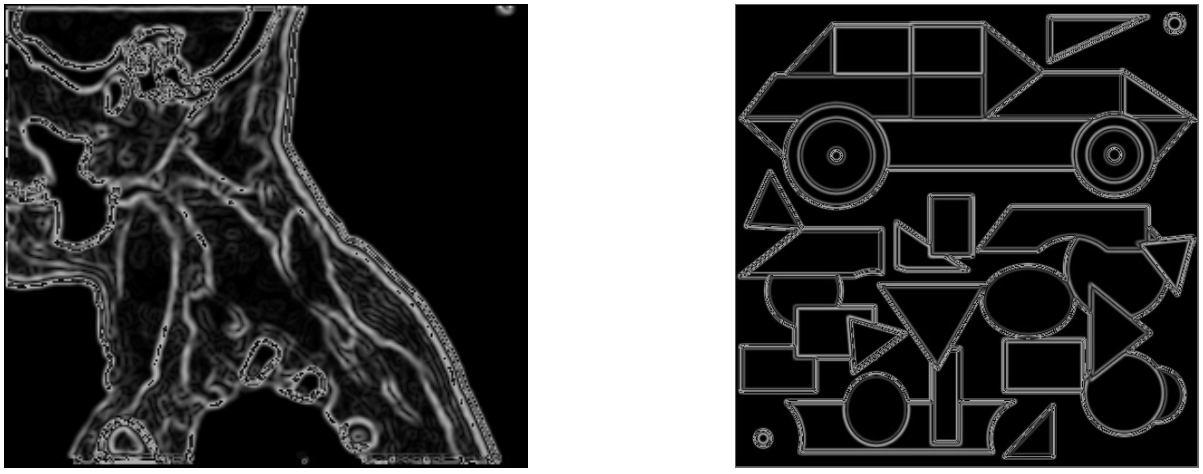
В аргументы функции поступают: изображение, его высота, ширина. Далее, после применения оператора Собеля с помощью формул (4) и (5) выполняется преобразование изображения и подсчет градиентов.

Тестирование функции: на вход имеются изображения: Рисунок 11 а) и б), которые получились в результате преобразования изображения с помощью размытия фильтром Гаусса, а на выход получаются: Рисунок 15 а) и б).



а) *б)*

Рисунок 11. Томографический снимок (а) и геометрические фигуры (б) после применения фильтра Гаусса.



а) *б)*

Рисунок 15. а), б) Результат применения оператора Собеля к Рисункам 11 а), б).

3.5. Подавление немаксимумов

После того, как оператор Собеля был вычислен, нужно определить угол направления вектора границы. Направления вектора округляется до ближайшего угла, кратного 45° (0, 45, 90 и 135°). Этот угол вычисляется по формуле (7). Затем проверяется достижение локального максимума величиной градиента в найденном направлении вектора [4]. Единственным отличием этого шага от предыдущего станет то, что в результате подавления локальных неопределенностей толщина линии границы становится равномерной и тонкой, что увеличивает точность ее расположения. Этот шаг также позволяет не разрывать саму границу.

$$Angle(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{7}$$

Направление градиента есть направление максимального возрастания функции. На этом основана процедура подавления немаксимумов. При этой процедуре для каждой точки рассматривается отрезок длиной в несколько пикселей, ориентированный по направлению градиента и с центром в рассматриваемом пикселе. Пиксель считается максимальным тогда и только тогда, когда длина градиента в нем максимальна среди всех длин градиентов пикселей отрезка. Граничными можно признать все максимальные пиксели с длинами градиента больше некоего порога.

Градиент яркости в каждой точке перпендикулярен границе, поэтому после подавления немаксимумов жирных линий не остается: на каждом перпендикулярном сечении жирной линии останется один пиксель с максимальной длиной градиента.

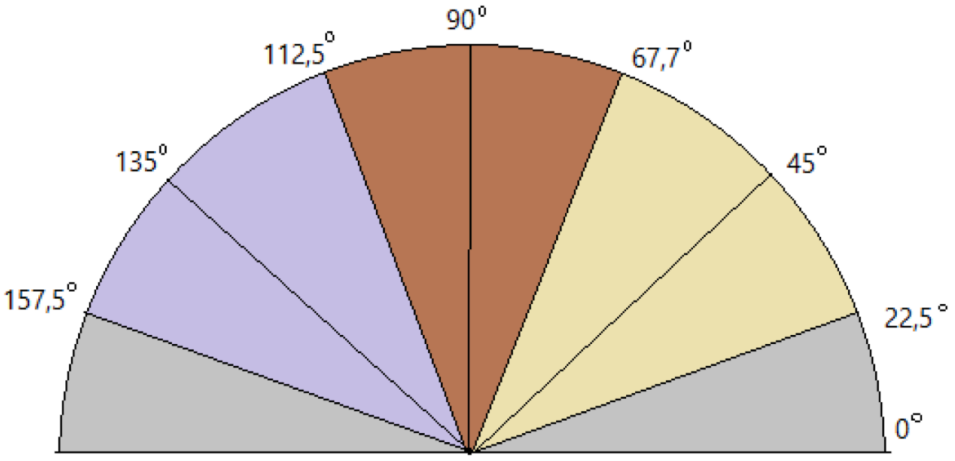
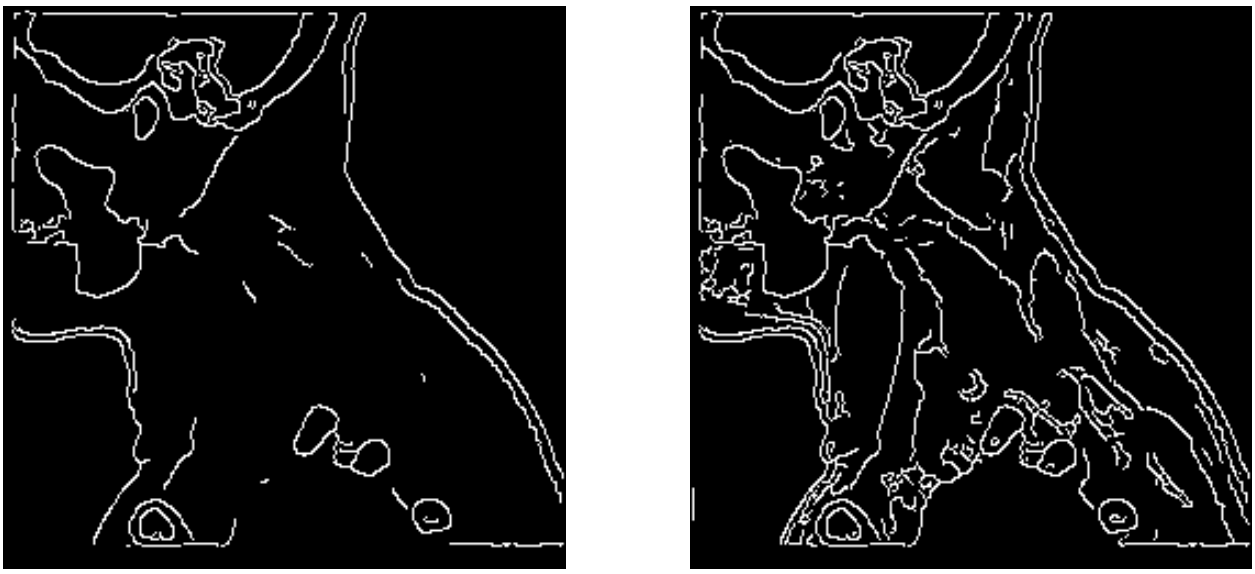


Рисунок 16. Сектора округления значений направления градиента [5]

3.6. Двойная пороговая фильтрация

Предпоследним шагом алгоритма Кэнни является применение порога. Он нужен для определения нахождения границы в данной точке изображения. Чем меньше порог, тем больше границ будет находиться, но тем более восприимчивым к шуму станет результат, выделяя лишние данные изображения. Высокий порог может проигнорировать слабые края или получить границу фрагментами, что приведет к потере целостности снимка (Рисунок 17 а) в сравнении с Рисунком 17 б)). Поэтому важно подобрать правильное значение порога. Если в каком-либо месте на просматриваемом фрагменте значение градиента превышает верхний порог, то этот элемент остается также допустимой границей и в тех местах, где значение градиентов падает ниже этого установленного порога, до тех пор, пока она не станет ниже нижнего порога. Если на всем фрагменте нет ни одной точки со значением большим верхнего порога, то он удаляется. Таким образом, детектор границ Кэнни 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 (Рисунок 18). При таком подходе фрагмент границы обрабатывается как целое, что способствует удалению слабых границ.



а) *б)*

Рисунок 17. а) завышенный порог, threshold = 100,

б) пониженный порог, threshold = 50.

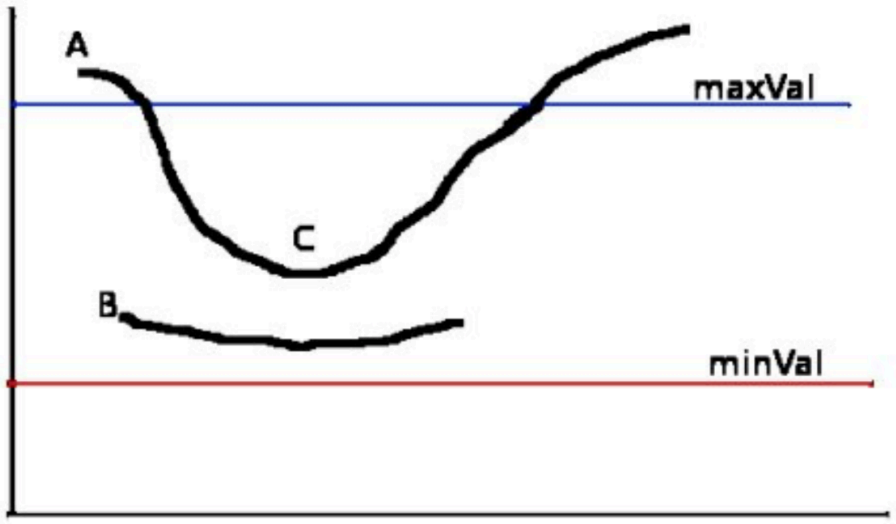


Рисунок 18. Демонстрация порогов

3.7. Трассировка области неоднозначности

Последним шагом нужно выделить группы пикселей, получивших в предыдущем этапе промежуточное значение. Так что нужно отнести их к границе, если они соединены с одной из установленных границ. Или же их нужно подавить, если они не соединены с границами.

Этот шаг не входит в основной базовый алгоритм детектора Кэнни. Трассировка области неоднозначности является мощной модификацией одного из самых больших недостатков алгоритма Кэнни, поэтому многие относят этот шаг к основному алгоритму.

Пиксель добавляется к группе, если он соприкасается с ней по одному из восьми направлений. Пример изображен на Рисунке 19.

Иными словами, результатом трассировки будет избавление от лишнего шума и исправление недостатка с тем, что предыдущий шаг порой забирал нужные линии и изображение теряло целостность. Особенно это болезненно сказывается на томографических снимках так как многие органы удаляются с изображения и в результате мы получаем голый скелет, только кости, так как их ярче всего видно при рентгене.

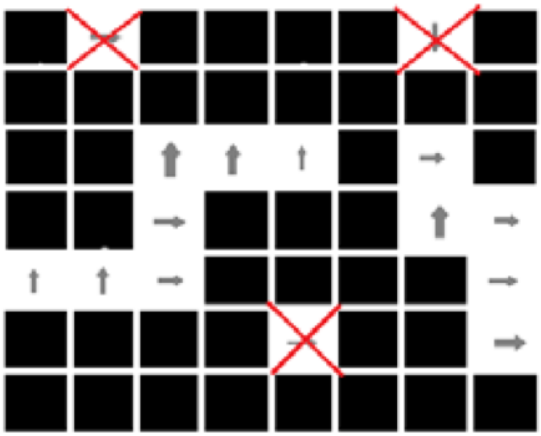


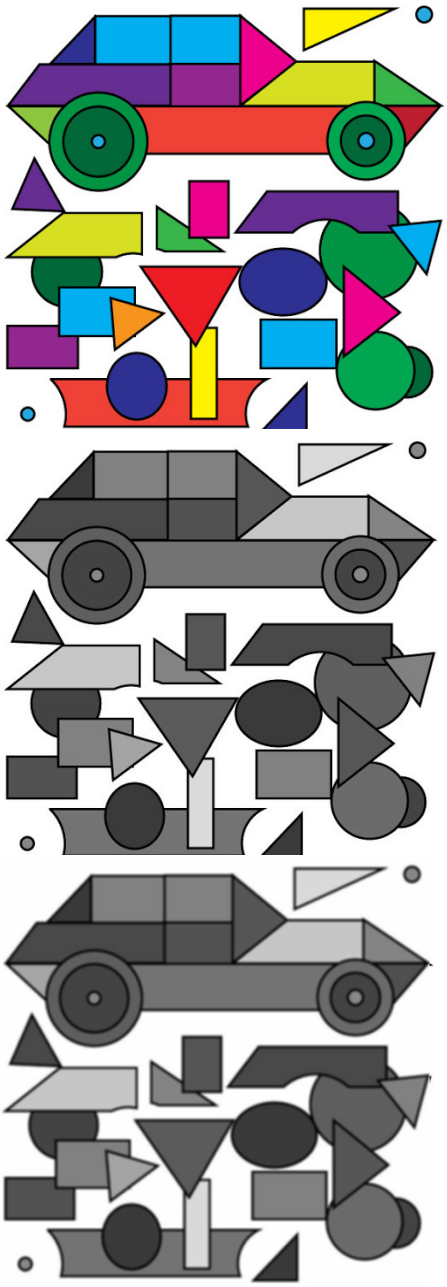
Рисунок 19. Подавленные пиксели зачеркнуты.

3.8. Обзор результатов

Итак, в общем виде алгоритм Кэнни выглядит как серия последовательного применения разных алгоритмов.

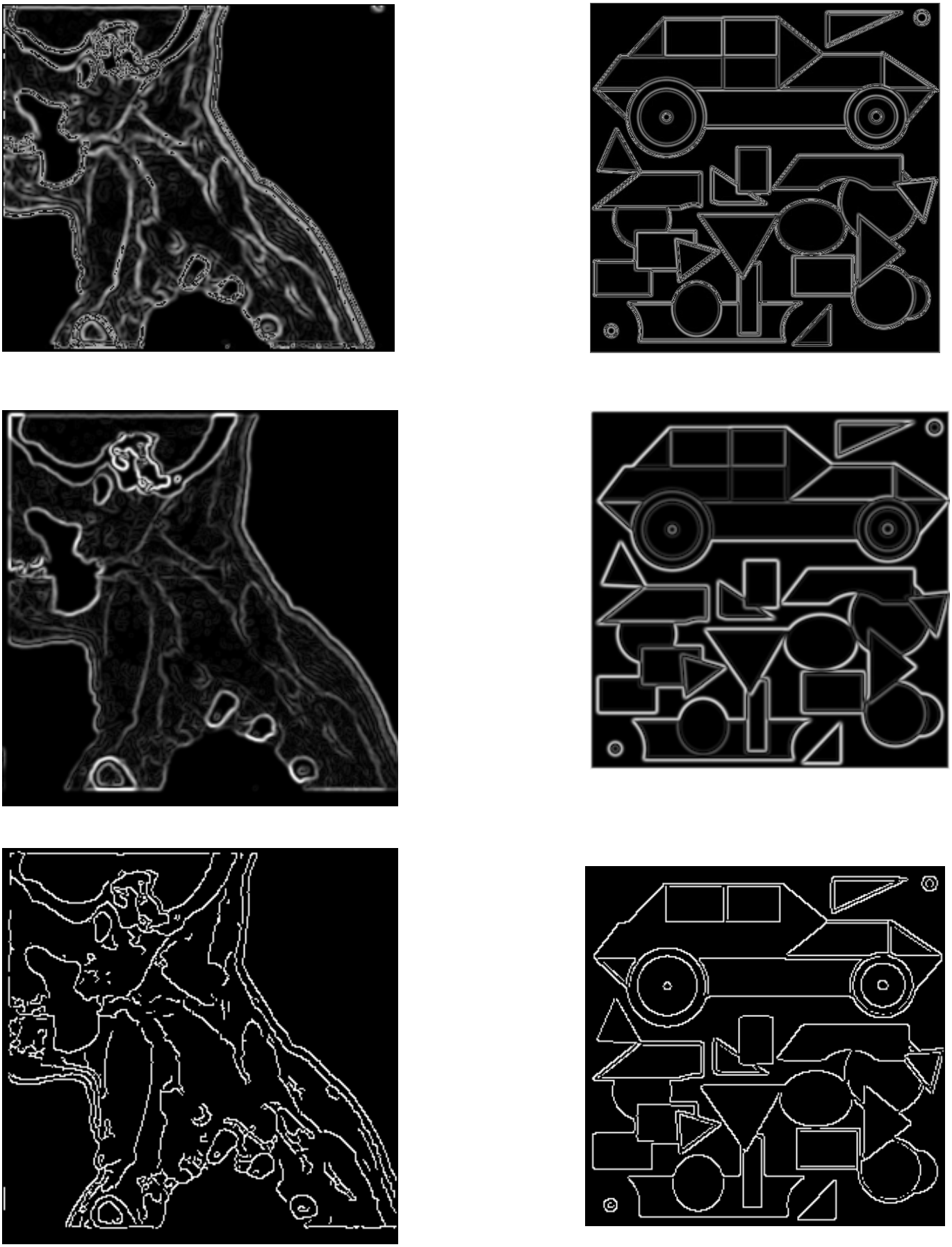
Далее сверху вниз показаны стадии обработки изображений (Рисунок 20 а) и б)) на каждый шаг от начального до результативного.

а)



б)

Рисунок 20. а), б) Демонстрация начальных шагов алгоритма



а) *б)*

Рисунок 22. а), б) Демонстрация финальных шагов алгоритма

4. Достоинства и недостатки алгоритма детектора границ Кэнни

4.1. Положительные стороны алгоритма Кэнни

Достоинствами данного алгоритма можно назвать слабую чувствительность к шумам и ориентации границ областей. Также алгоритм достаточно четко выделяет границы объектов, что в свою очередь позволяет лучше выявлять внутреннюю структуру объекта и исключает ошибочное обнаружение контура там, где объектов нет, что является хорошим качеством при использовании этого алгоритма в медицинских целях. Так как по снимкам впоследствии врач сможет увидеть возможные опухоли и принять решение о лечении или хирургических вмешательствах.

В сравнении с другими алгоритмами, у алгоритма Кэнни отсутствует проблема пропуска границ. В операторах использующих однопороговый фильтр возможен проскок истинной границы. Это значит, что если градиент края лежит чуть выше или чуть ниже установленного порога, то оператор удаляет полезную часть края, оставляя его незаконченным. Чтобы исключить это в алгоритме Кэнни используется двухпороговая фильтрация. Также он обеспечивает ориентацию градиента кромки, что приводит к хорошей локализации края, в то время как в большинстве других алгоритмах этого не предусмотрено [7].

4.2. Отрицательные стороны алгоритма Кэнни

Применение алгоритма Кэнни в базовом варианте имеет множество недостатков, которые дают некорректный вывод. Алгоритм сам по себе является универсальным распознавателем границ, а универсальность, как правило, сбалансирована сложностью вычислений или реализации. Также универсальность обязывает человека подкручивать некоторые данные, например, порог, так как его значение влияет на каждое изображение по-своему. Иными словами, появляется необходимость настройки детектора практически для каждого изображения индивидуально. Это требует человеческих и временных затрат, поэтому на этапе предварительной обработки изображения по результатам оценки гистограммы необходимо определить пороговые значения коэффициентов, которые будут использованы при дальнейшей обработке.

Определение пороговых значений может зависеть от многих факторов: яркость картинки, количество пикселей одной яркости, распределение объектов по изображению, а также в какой-то мере размер самого изображения.

Ещё одним существенным недостатком является то, что алгоритм Кэнни формирует незамкнутые контуры там, где они замкнуты. Также из-за фильтра Гаусса, который убирает шум и сглаживает края, увеличивается вероятность пропуска слабых краев.

Заключение

На основе проведенного исследования был изучен алгоритм детектора границ Кэнни, выявлены его сильные и слабые стороны. В медицине, а особенно в диагностике заболеваний очень важна точность для досконального понимания где именно и в каком состоянии находится интересующий врача орган. Алгоритм Кэнни же позволяет точно определить границы, не порождая лишние несуществующие линии там, где их быть не должно. Благодаря этому можно увидеть реальное состояние внутренних объектов.

В итоге выполнения данной работы были изучены способы работы со сложной обработкой изображений, использующие математический анализ и численные методы, а также приобретены базовые навыки из области контурного анализа. Также были изучены алгоритмы используемые в области компьютерного зрения.

В результате реализации поставленных задач была разработана программа для обработки снимков, полученных с томографа. Также в папке проекта сохраняются все промежуточные выводы изображения после каждого из отработавших алгоритмов.

Список использованной литературы

1. Gabor T. Herman. Fundamentals of Computerized Tomography: Image Reconstruction from Projections (Advances in Computer Vision and Pattern Recognition). Springer; 2nd ed. 2010 edition (November 13, 2009).

2. Arash Partow. C++ Bitmap Library. [Электронный ресурс] - https://www.partow.net/programming/bitmap/index.html

3. Документация библиотеки OpenCV. [Электронный ресурс] - https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

4. Власов А.В, Цапко И.В. Модификация алгоритма Канни применительно к обработке рентгенографических изображений. [Электронный ресурс] - https://core.ac.uk/download/pdf/53080194.pdf

5. Маврин Е.М. Сравнение алгоритмов выделения контуров на цифровом изображении и выбор наилучшего алгоритма для реализации на ПЛИС. [Электронный ресурс] - https://cyberleninka.ru/article/n/sravnenie-algoritmov-vydeleniya-konturov-na-tsifrovom-izobrazhenii-i-vybor-nailuchshego-algoritma-dlya-realizatsii-na-plis/viewer

6. Матричные фильтры обработки изображений. [Электронный ресурс] - https://www.mathworks.com/help/images/f18-12508.html;jsessionid=f9b490967a5eb1b3b0f0ea7088c1#f18-20972

7. Rashmi, Makes Kumar, Rohini Saxena. Algorithm and Technique on Various Edge Detection : A Survey. [Электронный ресурс] - https://www.researchgate.net/publication/283766898_Algorithm_and_Technique_on_Various_Edge_Detection_A_Survey