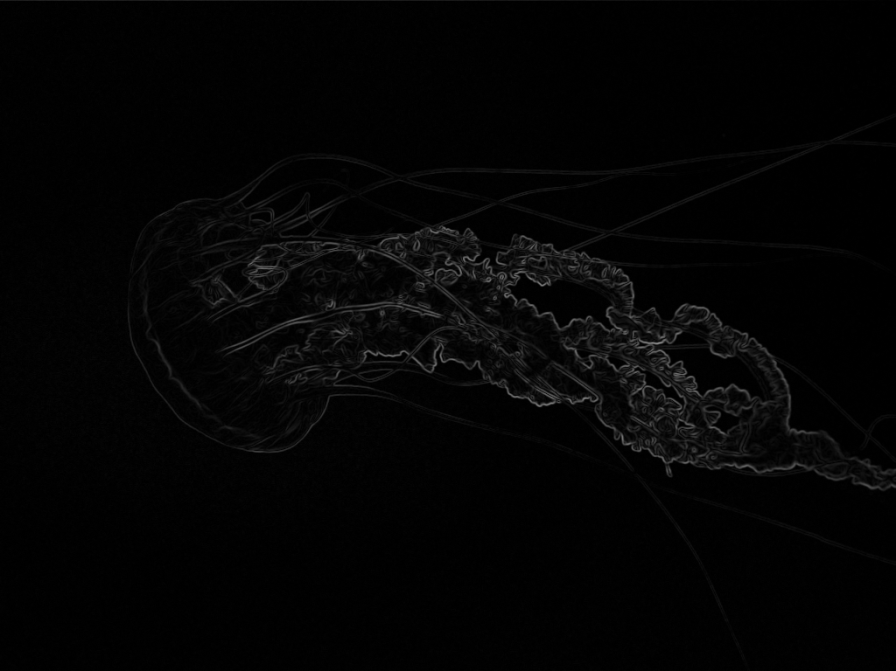
# The Computing Practical project



From Microsoft sample images

**Digital Image Manipulation**

|  |  |
| --- | --- |
| **Candidate Name** | Jonathan Keable |
| **Candidate Number** |  |
| **Centre Name** | Woodbridge School |
| **Centre Number** | 19269 |
| **Date** |  |

# Initial project background information and identification

Not soon after the first digital image was captured, it was desired to be able to edit digital images in ways never possible before such technology. In this project I will be creating a simple bitmap editing system able to carry out a number of editing techniques including edge detection, noise removal, smoothing, and red eye reduction; all of these techniques will be part of a learning tool designed around the image editing chapter of the AS physics syllabus. Other kernel imaging algorithms may be added later as expansion material on the syllabus.

The project was suggested to me by my A-Level physics teacher, Mr Cottrell, who has been a most cooperative ‘customer’ and I thank him sincerely for his aid. He suggested the project as many of the algorithms and ideas to be used are part of the AS physics syllabus. I have also however used other sources for more information on the subject, so that my knowledge on the wider subject of image editing is good enough for the system to be cohesive. The problem is that the current electronic learning system used by the physics department has been designed to cover the entire syllabus and thus is inadequate in certain areas, notably the image manipulation chapter of the AS course. Thus Mr Cottrell would like a simpler system that focuses entirely on image editing, specifically the algorithms I have already described. Some edge detection methods outside of the syllabus should be included to show how there are other techniques for this complex image editing device other than the simple Laplacian method covered in the AS syllabus. The system should be compact and self contained unlike the current system which requires multiple programs and has large amounts of material not related to the specific task in hand.

For clarity, I have explained the algorithms I will use below (n.b. all kernels are applied to the original image, not the image after some of the pixels have already had the kernel applied to them).

## Edge detection

### Simple Laplacian Edge Detection

|  |  |  |
| --- | --- | --- |
|  | -1 |  |
| -1 | 4 | -1 |
|  | -1 |  |

In the AS syllabus, the simple edge detection kernel is used. Though not the most effective algorithm, for the purposes of this project I will implement it. The kernel is a simple+ shape, as shown by this diagram.

By multiplying the pixel you are looking at by 4, and subtracting each one of its 4 immediate neighbours, and taking the modulus of an answer, we get a result that we can use to decide where edges are. The larger the values are the more likely they are edges. By setting the tolerance, we can decide how sensitive the kernel is. A slightly less noise sensitive, but more complex kernel is this:

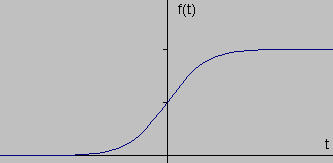
|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

This kernel works in a similar way, by multiplying the pixel by 8, and subtracting its direct and diagonal neighbours from it. Because it covers a larger area, it is less affected by a single pixel of noise. This second kernel is also discussed in the AS physics course and will likely be the one I will implement

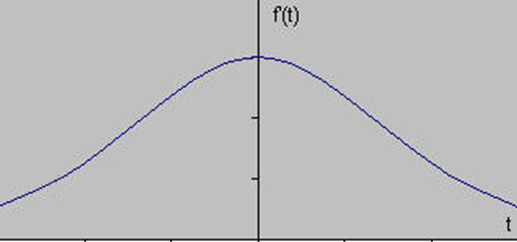
### Sobel Edge detection

I may also implement sobel edge detection if time allows, though this would be an extension of the AS physics course. I have explained the two methods below. [Some of The information in this section is paraphrased from [www.pages.drexel.edu/~weg22/edge.html](http://www.pages.drexel.edu/~weg22/edge.html). unless stated otherwise images are also from this website, and are under the copyright ownership of the author of the website].

The sobel method is of a group of edge detection method known as gradient methods. These methods use the gradient of a 1 dimensional signal, using the magnitude of that gradient to decide where the edge is. By taking the gradient in the x direction and the y direction edges can be detected on a 2D image. The gradient of a function is known as a derivative, and the sobel method treats every one dimensional line of pixels as a separate function. He is a simple example of what a function version of a line of pixels would look like.

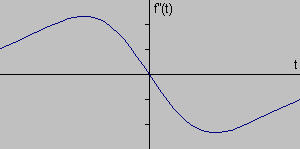


If we take the derivative for each point on the line of the function named t (f’(t)) we get this graph

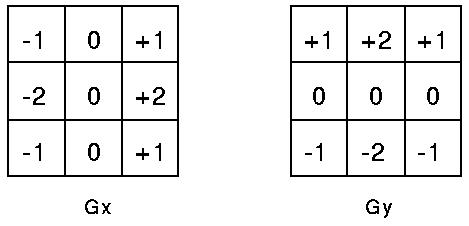


The graph clearly shows at what point the magnitude is greatest, and therefore where the edge is strongest on the image. If we set a tolerance, if the derivative at any point that which has a magnitude greater than the tolerance, then that point will be counted as an edge. An edge will either be assigned a black or a white value and a non edge will be assigned the opposite value. It would also be possible to set multiple thresholds and assign a different grey value to each value to different strengths of edge, for instance non-edges are black, and edges move closer and closer to white as they become stronger.

The Laplacian method explained in the physics textbook actually uses the 2nd derivative, i.e. the derivative of the first derivative. In this case it is the derivative graph of the graph above. By doing this, we see that at the point where the magnitude of the 1st derivative is greatest, the 2nd derivative of this point is 0. Thus the Laplacian method detects an edge where the 2nd derivative is equal to 0. [the below graph is the 2nd derivative of the function t, f’’(t)]



For sobel, in order to convert the 1D edge detection algorithm into a 2D one we use a pair of 3x3 convolution masks in the Sobel algorithm. Each one is an estimate of the gradient in either the x or y direction, and it is calculated by using the values of the 8 neighbouring pixels to get an approximation of the gradient. The algorithm for this is shown in the images: for the x direction subtract 2\* the value to direct left, subtract the values to the left and up and left and down, and then add 2\* the value to the direct right, and add the values to the right and up and right and down; the y direction works in a similar way. In order to combine the two gradient masks we must add the two gradient magnitudes together, but this only works if they are both positive. There are two ways of doing this: either use the squares of both gradients, add them together, and take the square root of the root, for a fairly accurate approximation, or take the modulus of both gradients and add those together to give us a less accurate approximation. The two masks and the two different approximation equations are given below.



|G| = √Gx2 + Gx2

|G| = |Gx| + |Gy|

A good explanation for how this works can be found on Wikipedia.

The result of the Sobel operator is a 2-dimensional map of the gradient at each point. It can be processed and viewed as though it is itself an image, with the areas of high gradient (the likely edges) visible as white lines. The following images illustrates this, by showing the computation of the Sobel operator on a simple image.

|  |  |
| --- | --- |
| Grayscale image of a brick wall & a bike rack | Normalized sobel gradient image of bricks & bike rack |
| Normalized sobel x-gradient image of bricks & bike rack | Normalized sobel y-gradient image of bricks & bike rack |

From wikipedia.org

It is important to note that the image above has been grey scaled before the edge detection algorithm has been carried out, and this is because grey scaling often improves the effectiveness of sobel edge detection. It would thus be useful to implement this algorithm also in the system, and it is explained below.

### Grey-scaling

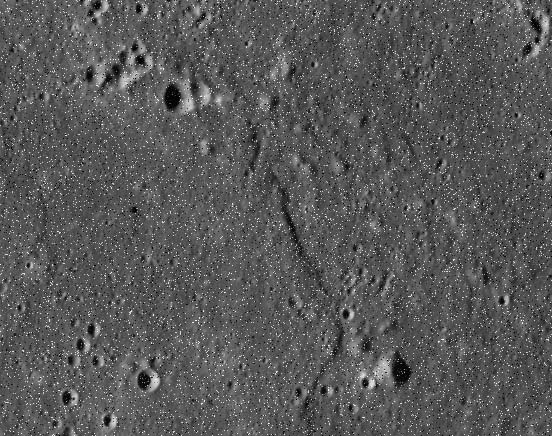
In order to convert a RGB image to greyscale you must look at the RGB values of each pixel. Then add together 30% of the Red Value, 59% of the green value, and 11% of the blue value for the best results for a typical image, and then store these as the linear luminescence values. These can be stored in 8 bit greyscale pixels, and the same edge detection algorithm applied to them.

In the Laplacian method we use just a single 3x3 convolution mask, as opposed to 2 3x3 grids in the sobel method. All of the values in the grid are subtracted from the central pixel multiplied by the number of pixels in the kernel excluding itself, and if the result is 0 an edge is said to be detected. This method is simpler, but more sensitive to noise, as only one mask is used.

For both methods the mask must be slid across the image either row by row, or column by column (usually the former). Because the masks are both 3x3, the algorithm cannot be applied to the pixels on edges and comers, without changing the kernel. I have a choice of either adding more kernels, or simply deleting the outer pixels of the image after the kernels are applied.

## Noise Removal

The algorithm for noise removal is relatively simple. Take the median value for the pixel itself and the 8 surrounding pixels in a 3x3 kernel, and set its value to that. This keeps most of the data, whilst still doing an effective job of removing noise. A noisy and cleaned up image are shown below.



[Images from ‘advancing physics AS edition’ learning tool, and ‘scion image editor’]

## Smoothing

The algorithm for smoothing, like noise removal, is relatively simple. Take the mean value for the pixel itself and the 8 surrounding pixels in a 3x3 kernel and set its value to that. This looses more data than median based noise removal, and makes the image appear blurred. By using the smoothing algorithm multiple times, noise can be removed, but this is not as effective as standard noise removal. An original image and the same image after multiple smoothing algorithms are shown below.



[Images from Microsoft (sample images)]

## Red Eye reduction

In order to reduce red eye the red colour in an image must be de de-saturated. This is achieved by searching for pixels within a certain tolerance of red (e.g. FF0000). In this example pixels from FFAAAA to AA0000 could be counted as red. Once these pixels are located, they can be changed to a grey value such as 888888. This will obviously cause all red pixels in the image to be changed to grey, so to avoid the loss of useful data it would be preferable for the user to select the area for red eye reduction to be applied before it is carried out.

# Description of the current system

Currently the Physics department uses a system called “advancing physics, AS edition”. It uses an “electronic textbook” with large amounts of reading material as well as question and answer sections. The system works well generally, but in order to demonstrate digital media editing it has a multiple program installation including programs such as ‘scion image editor’ and ‘audacity’. The scion program is a simple program that requires installation and can be difficult to use. For this problem Mr Cottrell would like a more ergonomic program that is stand alone and requires no installation. The current program (scion) has no real educational tools, and the new learning program should including some limited information on the kernels used, though its primary purpose will still be the implementation of the image editing techniques in the AS syllabus.

# Identification of the prospective user(s)

At the moment the only user who will definitely want to use the system is Mr Cottrell, as a learning tool on the projector, and the program could also be distributed to the physics laptops, though is less likely. Having said this, I will also be able to use the system of course, and other students may wish to view the source code and program to be created for the purposes of learning, and thus I will likely make it available through the use of the shared documents system, specifically the sandbox area allocated for programs created by students, at some point. The sharing of code for the purposes of the improvement of programming is common in the school computing community and the sharing of code through the sandbox is very likely to occur at some point.

# Identification of user needs and acceptable limitations

Here I will explain what the system needs to be able to do for the user and what it would be useful for it to do if possible in the specified time with the resources available. Firstly the system *must* include the four algorithms which are outlined in the AS advancing physics textbook which are:

* Laplacian Edge Detection
* Red-Eye Reduction
* Smoothing
* Noise Reduction

Mr Cottrell must have these algorithms both implemented into the system as well as described in detail to the user of the system, namely students, so that they understand how they work. It is important that the students can be set tasks to do using the system by the teacher which they can do both in and out of class. To this end a simple menu based system has been suggested, although the most important point is that all 4 of the main algorithms are in one place.

However there are other elements to imaging which though not explicitly explained in the AS syllabus are useful to understand for more adept students. These are:

* Grey scaling: Students do not need to know how grey scaling works, but should be aware that edge detection on colour images is much less reliable due to the way the algorithm works. As such Mr Cottrell would like a grey scaling algorithm implemented into the system so that students can compare results of edge detection with grey and colour images. Alternatively Images could be provided with the system which are in grey and colour if this is more practical
* Negative: Edge Detection leaves edges white with not edges black when carried out on an image. Some people find it difficult to distinguish white on a black background, so it would be useful for students to be able to invert the colour so that they can clearly see the edge detection algorithm has worked.
* Red Tolerance Editing: The AS course tells students that red eye reduction works by finding all “red” pixels on an image and simply replacing them with an arbitrary colour, usually grey. However more advanced student will understand that there are many shades of “red” and it would be useful, though not essential, to have a part of the program which allows the user to tell the program what “red” is, so that they can see the effect this has on the application of the algorithm.

Because the system will be a learning tool it must be provided with a number of predefined images which exhibit well the imaging algorithms in the AS course. These images should be chosen to show the algorithms working on both a large scale (on full size images) and on a small scale (small grids of pixels) So that they can be used in a structured way as part of teaching the particular part of the course. These images should have a combined size in terms of memory used so that the whole system can fit on portable media, most likely CD’s ,although personal USB drives or even a downloadable form of the system on the school’s Virtual Learning Environment may also be used. Ultimately however the result of any of these distribution methods is that the system will be limited in size.

The system should also be able to manipulate images other than those provided however so that users can look at how the algorithm affects other images, to help encourage learning outside class and interest in this art of the course. For instance users may wish to see how effective the red eye reduction algorithm is on an image of their face with redeye, and by seeing how these algorithms work on a variety of images Mr Cottrell hopes users of the system will be more interested in the algorithms used.

The system must also work using the resources available to the Physics department. This means it must be able to be installed on the laptops which are situated there. These machines are fairly low spec so the system should work efficiently on slow processors. These machines are also in the process of being replaced with new machines running “Windows 7” as opposed to “Windows XP”, so the system should work on both systems. The laptops also support multiple users, and the system should be available to all AS Physics teachers and students.

# Data Source and Destination

## Inputs

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Source | Input by | Frequency |
| Input Bitmap Image | User Documents | User | N/A |
| Editing Required | User | User | Every time the user wishes to edit the image |
|  |  |  |  |

## Outputs

|  |  |  |
| --- | --- | --- |
| Item | Destination | Frequency |
| Output Bitmap Image | Cached to Disk | Whenever the user saves the new image |
| Temporary Bitmap Image | Temporarily Cached to Disk, Displayed to user | Every time Image editing algorithms are carried out |
| Input Bitmap Image | Cached to Disk | Whenever the user Inputs an image |
| Algorithm Data | Cached to Disk | Accessed by the program when editing is carried out |
| Kernel Data | Cached to Disk | Accessed by the program when editing is carried out, part of Algorithm Data |
|  |  |  |

# Data Volumes

All images will have a size which varies depending on size and detail. For instance a 32 bit per pixel image of 1000 by 2000 pixels would take up 4\*2000\*1000 = 8MB. Thus significant memory may need to be allocated in order to store these files.

Algorithm data and kernel data will be negligible in size.

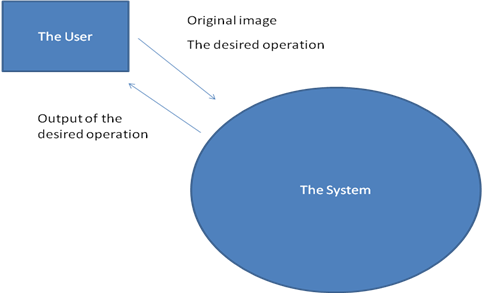
# Analysis Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable (e.g. myvariable) | Type (e.g. integer) | Field size | example | comment |
| OriginalImage | File (Bitmap) | An acceptably large sized and below bitmap with an acceptably high and below resolution | Chrysanthemum.bmp | The original bitmap image will be read from directly from the hard drive, and the system must be able to convert the image to a 2D array of binary for editing, and back to a bitmap for viewing by the user |
| TemporaryImage | File (Bitmap) | An acceptably large sized and below bitmap with an acceptably high and below resolution | Chrysanthemum.bmp | The temporary image will be stored in a temporary location for editing. It  will be stored as a file (Bitmap) for viewing by the user, but will also be stored as a 2D array of binary for editing |
| TempImageBin | Array of binary (2D) [x..y][z..w] | An acceptably large sized and below bitmap with an acceptably high and below resolution, converted to binary | Chrysanthemum.bin | The binary version of the temporary image for editing |
| EditedImage | File (Bitmap) | An acceptably large sized and below bitmap with an acceptably high and below resolution | Chrysanthemum.bmp | When the user saves the image it will be saved to a permanent location as a bitmap file only |
| pixeldata | record | A record of Binarydata, RGBvalue, and neighbouringpixels | n/a | A record of both the data stored in the pixel, and of where the pixel is in the image and what pixels neighbour it |
| Binarydata | Binary integer | From 1 bit to 64 bits | 1101 0001 | The raw binary data in each pixel |
| RGBvalue | HEX integer | From 1 bit to 64 bits stored as Hex | FA156B | The Hexadecimal value for each pixel so that it can easily be manipulated |
| neighbouringpixels | array | An array of 2d array coordinates with 4 different storagespaces | [(2,2),(2,4),(3,1),(3,3)] | An array of the locations of pixels in the bitmap file  which neighbour a certain pixel |
| redtolerance | record | A record containing redeyemin and redeyemax | n/a | Record containing the red eye tolerance values |
| redeyemin | integer | A Hex integer for the minimum value which will be accepted as red | 660000 | The minimum HEX value for which the red eye part of the system will take as red |
| redeyemax | integer | A Hex integer for the maximum value which will be accepted as red | FF6666 | The maximum HEX value for which the red eye part of the system will take as red |
| edgetolerance | integer | An integer value from -2147483648 to 2147483647 | 5 | When the sobel edge detection function outputs a value, if it is above or equal to this number then an edge has been detected. |
| Functionrequired | string | A string of up to 255 characters | RedEye | Each function will have a string value assigned to it so it can be identified as the required process. |
| neighbourmean | integer | An integer value from -2147483648 to 2147483647 | 12324128 | The mean of the 8 neighbours of a pixel |
| neighbourmedian | Integer | An integer value from -2147483648 to 2147483647 | 12324128 | The mean of the 8 neighbours of a pixel |
| … | … | … | … | … |

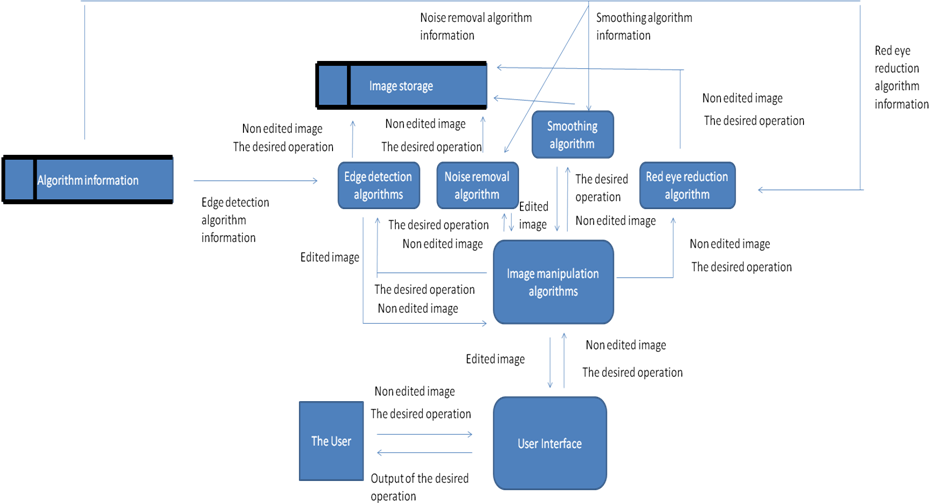
# Data Flow Diagrams

Due to the nature of the problem, no original system DFD is possible, however the DFD for the new system is:

## L0



## L1



# Numbered SMART objectives

1. The system must accurately and effectively implement the following algorithms
   1. Edge detection
      1. the Laplacian method must be implemented
      2. A sobel method could be implemented
      3. For the sobel method the user should be able to change for what value it is said an edge has been detected.
      4. The user must be able to choose whether the edges are white on black, or black on white.
      5. The system must be able to convert an image to greyscale before any edge detection is carried out
      6. The conversion of images to edge detection images should take no longer than 30 seconds
   2. Smoothing
      1. The system should be able to smooth images using the mean technique
      2. It could be possible to enter how many times the smoothing process should be repeated, if any.
      3. It should take no longer than 30 seconds for images to be smoothed
   3. Noise removal
      1. The system should be able to remove noise using the median method
      2. There should be as little loss of data as possible during noise removal
      3. It could be possible to enter how many times the noise removal process should be repeated, if any.
      4. It should take no longer than 30 seconds for noise to be removed from an image
   4. Red eye reduction
      1. It should be possible for the system to convert red pixels in an image to grey pixels
      2. It should be possible for the user to change the red tolerance of the process
      3. It could be possible for the user to select an area for which red eye is to be reduced
      4. It should take no longer than 30 seconds for red eye to be reduced.
2. There should be certain User Interface functions
   1. The user should be able to save an edited image without overwriting the original
   2. The user could be able to undo changes to an image for at least 10 image edits
   3. The user should be able to revert to the original image
   4. There could be certain selection tools
      1. The user could be able to select a rectangular part of the image
      2. The user could be able to deselect a part of the image
      3. The user could be able to select more than one part of the image at a time
      4. The user could be able to apply the red eye algorithm only to a selection
      5. These tools may be based either on a curser system or on a coordinates system if this is impractical.
   5. The user could be able to preview changes before they are made
3. The system should be easy to use
   1. It should not crash or have problems frequently
   2. It should be clear where different tools are
   3. It should not be possible to easily lose an edited image or delete an existing one
   4. The system must contain brief summaries of which each function does
4. The system should not damage other systems
5. The System should be compact enough to be sent by e-mail or to be put on a CD-ROM, DVD, or Flash memory Stick (it does not have to fit onto a Floppy Disk).

# Potential and chosen solutions

## Edge Detection

The reason I have chosen to approach the problem in a software based environment because it is very difficult to show how image processing algorithms work without a computer. Although students can work on a small scale, looking at for instance a 8x8 grid of pixels with values between 1 and 16, it is almost impossible to look at actual images with 1000s of pixels and colours. The reason I used Turbo Delphi to write the program is that is the language I am most comfortable with, and also contains a GUI building interface, so that I do not have to create classes for buttons, forms, images etc. Asides form these things it also possesses all the functionality required to write image editing software.

## Algorithm Choices

There are 4 main algorithms that are used for edge detection

1. Laplacian (zero edge detection)
2. Sobel (gradient edge detection)
3. Canny
4. Difference of two Gaussians

I must implement Laplacian edge detection as it is in the AS Physics course. None of the other algorithms are in the AS physics course, but it would be nice if maybe one of them could be implemented if time allows, and this would likely be the sobel algorithm. I have already explained Sobel and Laplacian edge detection, so I will explain how Canny and Difference of two Gaussians work and why I have chosen not to implement them

The Smoothing and Noise Reduction algorithms are taken directly from the AS physics book, using a mean and median 3x3 kernel respectively. There is no easy way to expand on these algorithms so I will not attempt to as I only need to cover the kernels in the syllabus anyway.

The Red eye reduction kernel is a simple “find colour red and replace with grey” process so I need not explain it in detail, and for the AS physics course it is the only procedure students need to know about.

In order to allow edges to be black on white or white or black a negative process will be implemented. It simply inverts colour by doing 255 – colour, and is not in the AS course so will not be explained in educational information. The Greyscaling algorithm I use converts colour to grey using set values for how much of each colour (RGB) is included in grey. It too is purely for the purpose of making the edge detection algorithm more effective so also needs no explanation in educational information.

### Canny Edge Detection

The **Canny** [**edge detection**](http://en.wikipedia.org/wiki/Edge_detection) operator was developed by [John F. Canny](http://en.wikipedia.org/wiki/John_F._Canny) in 1986 and uses a multi-stage [algorithm](http://en.wikipedia.org/wiki/Algorithm) to detect a wide range of edges in images. Most importantly, Canny also produced a *computational theory of edge detection* explaining why the technique works.

From wikipedia.org

The Canny edge detection is a multistage algorithm that has 6 main stages:

* + [2.1 Noise reduction](http://en.wikipedia.org/wiki/Canny_edge_detector#Noise_reduction)
  + [2.2 Finding the intensity gradient of the image](http://en.wikipedia.org/wiki/Canny_edge_detector#Finding_the_intensity_gradient_of_the_image)
  + [2.3 Non-maximum suppression](http://en.wikipedia.org/wiki/Canny_edge_detector#Non-maximum_suppression)
  + [2.4 Tracing edges through the image and hysteresis thresholding](http://en.wikipedia.org/wiki/Canny_edge_detector#Tracing_edges_through_the_image_and_hysteresis_thresholding)
  + [2.5 Differential geometric formulation of the Canny edge detector](http://en.wikipedia.org/wiki/Canny_edge_detector#Differential_geometric_formulation_of_the_Canny_edge_detector)
  + [2.6 Variational-geometric formulation of the Haralick-Canny edge detector](http://en.wikipedia.org/wiki/Canny_edge_detector#Variational-geometric_formulation_of_the_Haralick-Canny_edge_detector)

From wikipedia.org

This method is one of the most effective edge detection methods known. Unfortunately it is also one of the most complex methods of edge detection, so is probably too complex for the A2 course. Basically it uses a Gaussian noise removal kernel, a gradient edge detection kernel, and the direction of the edges, to trace the image as if traced manually. This makes its outputs very accurate, but the sheer complexity of the algorithm means it is not often used. The image below shows just how effective it is.



From Wikipedia.org, licensed under the Creative Commons Attribution-ShareAlike 3.0 License; author: JonMcLoone

The images clearly show the effectiveness of the algorithm: the blurred edges that are not useful in the background are ignored, but even the individual hairs are identified as edges.

|  |  |
| --- | --- |
| ***Advantages*** | ***Disadvantages*** |
| The most effective Edge Detection Algorithm, as it uses multiple algorithms to work out all the useful properties of an edge | The most complex algorithm to implement |
| The algorithm least effected by noise due to the Gaussian blur part of the algorithm | Due to the complexity of the algorithm the processing of large and/or high resolution images is slow |
| Is able to detect even very thin edges | Would require an amount of time to program which would be unreasonably considering the timescale involved  Far beyond what is studied in the AS syllabus |
|  |  |

### Difference of Gaussians

In [computer vision](http://en.wikipedia.org/wiki/Computer_vision), **Difference of Gaussians** is a [grayscale](http://en.wikipedia.org/wiki/Grayscale) image enhancement algorithm that involves the subtraction of one blurred version of an original grayscale image from another, less blurred version of the original. The blurred images are obtained by [convolving](http://en.wikipedia.org/wiki/Convolution) the original grayscale image with Gaussian kernels having differing standard deviations. Blurring an image using a [Gaussian](http://en.wikipedia.org/wiki/Gaussian_blur) [kernel](http://en.wikipedia.org/wiki/Convolution_kernel) suppresses only [high-frequency spatial](http://en.wikipedia.org/wiki/Spatial_frequencies) information. Subtracting one image from the other preserves spatial information that lies between the range of frequencies that are preserved in the two blurred images. Thus, the difference of Gaussians is similar to a [band-pass filter](http://en.wikipedia.org/wiki/Band-pass_filter) that discards all but a handful of spatial frequencies that are present in the original grayscale image. [[1]](http://en.wikipedia.org/wiki/Difference_of_Gaussians#cite_note-micro.magnet.fsu.edu-0)

From Wikipedia.org

By subtracting a blurred version of an image (blurred using a Gaussian kernel), you can achieve a greyscale edge detection image. The Gaussian kernel is quite a complex kernel, and is often large (5x5, 6x6, or even larger), making it a slightly more complex and slower algorithm, that is very good at eliminating noise, though not as effective as Canny Edge detection. It is therefore often used for high noise images, however is not used as much as other edge detection algorithms on most images; Because of this, and the complexity of Gaussian kernels, I have chosen not to implement this algorithm.

The images show that though good at not detecting noise, difference of Gaussians also loses some detail

Images from Wikipedia.org, licensed under the Creative Commons Attribution-ShareAlike 3.0 License; author: SadaraX



|  |  |
| --- | --- |
| ***Advantages*** | ***Disadvantages*** |
| Not Very Sensitive to noise (though slightly more sensitive than Canny edge detection) | Poor Detail |
|  | Can take more time than most methods to process images due to kernel complexity, but faster than Canny edge detection |
|  | Greyscale only  Not widely used and not cohesive with the AS course |
|  |  |