CSC1102 Computer Graphics and Image Processing

Lecture #8 - Edge Detection(2)

Dr. Gerard Marks

School of Computing
Dublin City University

Email: gerard.marks@dcu.ie

2024/2025

Edge Detection

Algorithms for edge detection contain three steps:

Filtering: Since gradient computation based on intensity values of only two points are susceptible to noise and other vagaries in discrete computations, filtering (lowpass filtering) is commonly used to improve the performance of an edge detector with respect to noise. However, there is a trade-off between edge strength and noise reduction. More filtering to reduce noise results in a loss of edge strength.

Enhancement: In order to facilitate the detection of edges, it is essential to determine changes in intensity in the neighborhood of a point. Enhancement emphasizes pixels where there is a significant change in local intensity values and is usually performed by computing the gradient magnitude.

Edge Detection

Detection: We only want points with strong edge content. However, many points in an image have a nonzero value for the gradient, and not all of these points are edges for a particular application. Therefore, some method should be used to determine which points are edge points. Frequently, thresholding provides the criterion used for detection.

It is important to note that detection merely indicates that an edge is present near a pixel in an image, but does not necessarily provide an accurate estimate of edge location or orientation. The errors in edge detection are errors of misclassification: false edges and missing edges.

Edge Detection

The simplest 2D differentiating kernel is the **Prewitt** operator, which is obtained by convolving a 1D Gaussian derivative kernel with a 1D box filter in the orthogonal direction. This operator does not place any emphasis on the pixels that are closer to the center of the mask. These kernels can be defined as:

$$\text{Kernel for } \frac{\partial I}{\partial x} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\text{Kernel for } \frac{\partial I}{\partial y} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

The **Sobel** is one of the most commonly used edge detectors which is more robust than Prewitt. The Sobel edge enhancement filter has the advantage of providing differentiating (which gives the edge response) and smoothing (which reduces noise) concurrently. These kernels can be defined as:

$$\text{Kernel for } \frac{\partial I}{\partial x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\text{Kernel for } \frac{\partial I}{\partial y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Edge Detection

For completeness, we mention the classic **Roberts cross** operator, which is the oldest pair of gradient kernels. The Roberts kernels suffer from two drawbacks. First, they compute derivatives along the diagonal directions rather than along the x and y axes, which is more of an inconvenience than any fundamental limitation. Secondly, because they have even dimensions, the kernels are not centered. That is, they do not compute the derivative at a pixel, but rather between pixels. These kernels can be defined as:

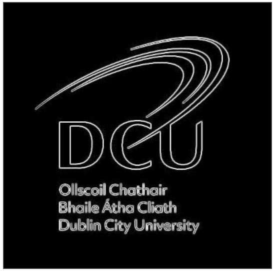
$$\text{Kernel for } \frac{\partial I}{\partial x} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\text{Kernel for } \frac{\partial I}{\partial y} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

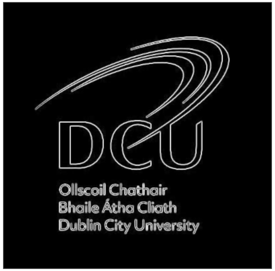
Edge Detection

Soble, Prewitt and Roberts operators are known as **Gradient Based** edge detection operators which compute the first derivative of a digital image. The images below illustrate the magnitude of the gradients:

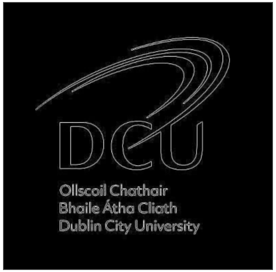
Prewitt



Sobel



Roberts



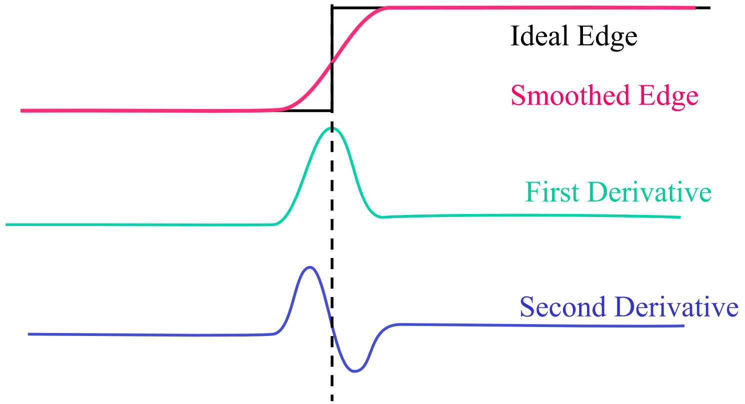
Edge Detection

The edge detectors discussed earlier computed the first derivative and, if it was above a threshold, the presence of an edge point was assumed. This results in detection of too many edge points.

A better approach would be to find only the points that have local maxima in gradient values and consider them edge points.

This means that at edge points, there will be a peak in the first derivative and, equivalently, there will be a zero crossing (i.e. where the value changes from negative to positive and vice-versa) in the second derivative. Thus, edge points may be detected by finding the zero crossings of the second derivative of the image intensity.

Edge Detection



Edge Detection

The second category of edge detection methods are known as **Gaussian Based** operator which compute the second derivative of a digital image. **Laplacian of Gaussian (LoG)** and **Canny** edge detector are a few examples of this category.

The second derivative of a smoothed step edge is a function that crosses zero at the location of the edge.

The Laplacian is the two-dimensional equivalent of the second derivative. The formula for the Laplacian of a 2D image $I(x, y)$ is:

$$\nabla^2 I(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Edge Detection

Based on the equation in the previous slide, the Laplacian of an image is the sum of the second derivative along the two axes.

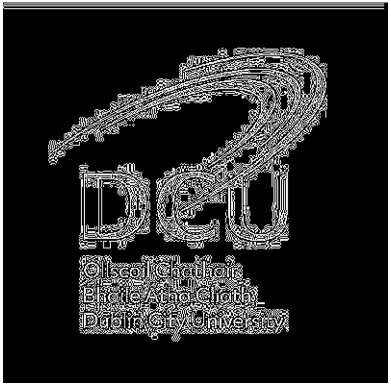
For a 2D image, the discrete Laplacian can be given as convolution with the following kernels:

$$\text{Kernel for } \nabla^2 I(x, y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

This kernel is approximating a second derivative measurement on the image.

Edge Detection

Edge points detected by finding the zero crossings of the second derivative of the image intensity are very sensitive to noise.



Edge Detection

To reduce the effects of noise, we can smooth the image with a Gaussian, then compute the Laplacian. To do this, the **Laplacian of Gaussian (LoG)**, combines Gaussian filtering with the Laplacian for edge detection.

Edge Detection

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result. Doing things this way has two advantages:

- Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations.
- The LoG kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image.

Edge Detection

The LoG kernel can be calculated:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right) exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

This equation contains Gaussian smoothing and second derivative computation, there is no need to conduct them separately, as long as an image is convolved with LoG.

Edge Detection

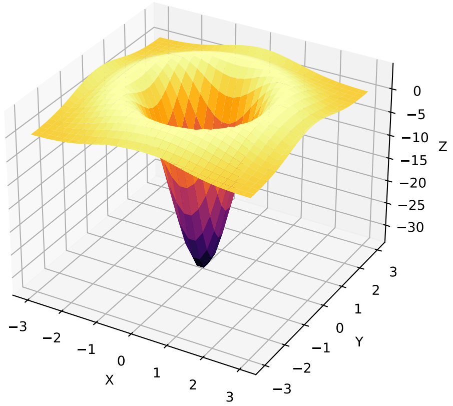
Ollscoil Chathair Bhaile Átha Cliath Dublin City University

The fundamental characteristics of the LoG edge detector are:

1. The smoothing filter is a Gaussian.
2. The enhancement step is the second derivative (Laplacian in two dimensions).
3. The detection criterion is the presence of a zero crossing in the second derivative with a corresponding large peak in the first derivative.
4. The edge location can be estimated with subpixel resolution using linear interpolation.

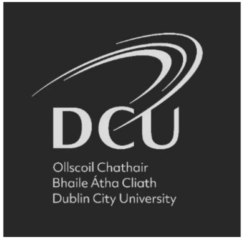
Edge Detection

The LoG function is also known as the inverted "Mexican hat" operator, because of its shape. It is rotationally symmetric and is a center-surround filter because it consists of a central core of negative values surrounded by an annular ring of positive values.

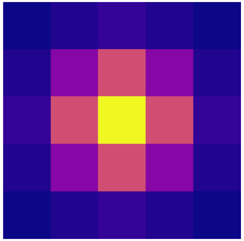


Edge Detection

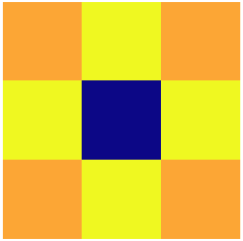
Input Image



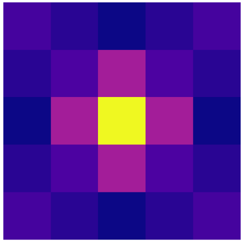
Gaussian kernel with $\sigma = 3$



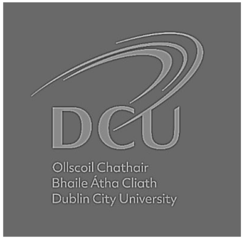
Laplacian kernel



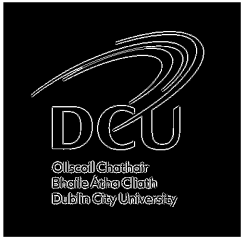
LoG kernel



Convolved image with LoG



Zero Crossing



Edge Detection

LoG has a few limitations:

- Very sensitive to noise.
- The localization error may be severe at curved edges.
- It generates noisy responses that do not correspond to edges, so-called "false edges".

Edge Detection

Canny edge detection is a multi-stage algorithm that detects wide range of edges:

1. Smooth image by filtering with a Gaussian.
2. Computing the gradient magnitude and orientation (using kernels like Sobel).
3. Perform non-maximal suppression to identify candidate edges.
4. Trace edge chains using Hysteresis thresholding.

Edge Detection

We are already familiar with the first two steps. The magnitude of image array $I(x, y)$ will have large values where the image gradient is large, but this is not sufficient to identify the edges, since the problem of finding locations in the image array where there is rapid change has merely been transformed into the problem of finding locations in the magnitude array that are local maxima.

Edge Detection

To identify edges, the broad ridges in the magnitude array must be thinned so that only the magnitudes at the points of greatest local change remain. This process is called **Non-maximum Suppression**, which in this case results in thinned edges.

Non-maximum is simply an edge thinning process. After computing our gradient magnitude representation, the edges themselves are still quite noisy and blurred, but in reality there should only be one edge response for a given region, not a whole clump of pixels reporting themselves as edges.

Edge Detection

Non-maximum suppression steps:

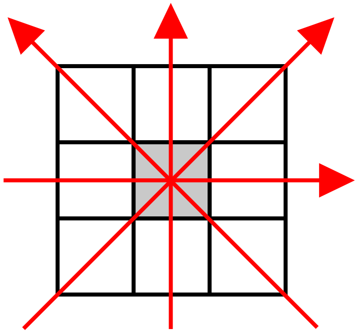
1. Compare the current pixel to the 3×3 neighborhood surrounding it.
2. Determine in which direction the orientation is pointing:
   - If it's pointing towards the north or south, then examine the north and south magnitude.
   - If the orientation is pointing towards the east or west, then examine the east and west pixels
3. If the center pixel magnitude is *greater than* both the pixels it is being compared to, then preserve the magnitude; otherwise, set the center pixel to 0.

Edge Detection

Some implementations of the Canny edge detector round the value of θ to either 0°, 45°, 90°, or 135°, and then use the rounded angle to compare not only the north, south, east, and west pixels, but also the corner top-left, top-right, bottom-right, and bottom-left pixels as well.

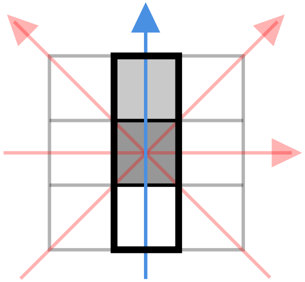
Edge Detection

A pixel can have total 4 directions for the gradient (shown below) since there are total 8 neighboring pixels.



Edge Detection

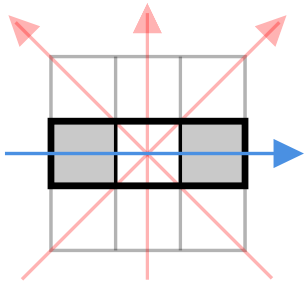
In the following example, the direction of gradient for the center pixel is $90°$.



Hence we will compare the magnitude of the gradient with both the pixel above $(90°)$ and below $(270°)$ it. In this example, white represents higher value (255) and black represents lower value (0). It can be seen that the bottom pixel has higher value than the one at the center. Hence the center pixel will be set to 0.

Edge Detection

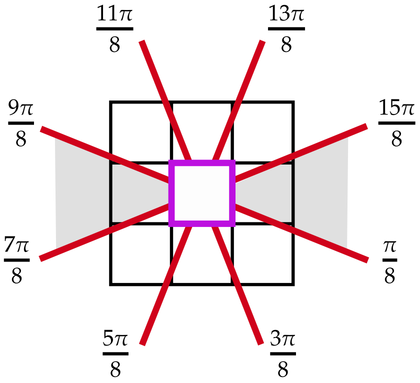
In another example, the gradient direction of the center pixel is $0°$. So we will compare the magnitude of gradient of the right $(0°)$ and left $(180°)$ pixel with it.



Clearly the magnitude of gradient of the center pixel is higher than the other two. Therefore, we update the output pixel value by the magnitude of gradient of the center pixel without any change.

Edge Detection

In general, a range of degrees are considered to select a neighbor. For example, if the direction of the center pixel is between $\frac{15\pi}{8}$ and $\frac{\pi}{8}$ and also between $\frac{7\pi}{8}$ and $\frac{9\pi}{8}$ then, the middle pixel should be compared with the left and right neighbor pixels.



Edge Detection

The typical procedure used to reduce the number of false edge fragments in the non-maximum suppressed gradient magnitude is to apply a threshold. All values below the threshold are changed to zero.

The result of applying a threshold to the non-maximum suppressed magnitude is an array of the edges detected in the image $I[x, y]$.

There will still be some false edges because the threshold τ was too low (false positives), and portions of actual contours may be missing (false negatives) due to softening of the edge contrast by shadows or because the threshold τ was too high.

Edge Detection

A more effective thresholding scheme uses two thresholds. The double thresholding algorithm takes the non-maximum suppressed image, and applies two thresholds τ_1 and τ_2, with $\tau_2 \approx 2\tau_1$, to produce two thresholded edge images $T_1[x, y]$ and $T_2[x, y]$.

Since image T_2 was formed with a higher threshold, it will contain fewer false edges; but T_2 may have gaps in the contours (too many false negatives).

The double thresholding algorithm links the edges in T_2 into contours. When it reaches the end of a contour, the algorithm looks in T_1 at the locations of the 8-neighbors for edges that can be linked to the contour.

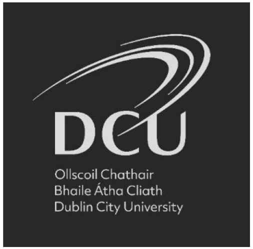
Edge Detection

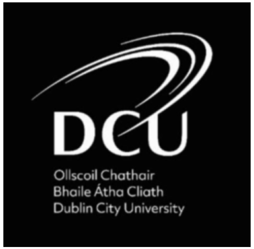
The algorithm continues to gather edges from T_1 until the gap has been bridged to an edge in T_2. The algorithm performs edge linking as a by-product of thresholding and resolves some of the problems with choosing a threshold.

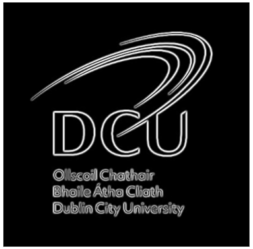
To simplify, **Hysteresis thresholding** can be defined with these conditions:

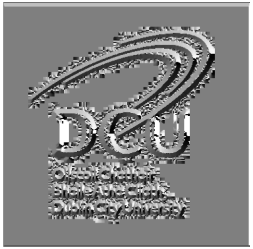
- Any edges with intensity greater than τ_2 are the definite edges.
- Any edges with intensity less than τ_1 are considered non-edges.
- The edges between τ_2 and τ_1 thresholds are classified as edges only if they are connected to a definite edge otherwise discarded.

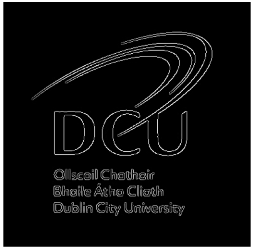
Edge Detection

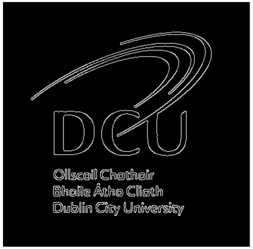

Input Image


Convolved image with Gaussian $\sigma = 1$


Gradient Magnitude using Sobel


Gradient Direction


Non-maximum suppression


Hysteresis thresholding

Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

```
End Lecture #8;
```