# EECS 4422/5323 Computer Vision

## Feature Detection Lecture 3

Calden Wloka

25 September, 2019

# Announcments

- Bring your Python questions to office hours or labs
  - Recommendation for your own machine: Anaconda
  - Installing OpenCV through `conda`: link
  - Note that `matplotlib` works differently in Jupyter than in standard Python, requires `matplotlib.pyplot.show()` to display figure windows

# Outline

- Linking Edge Elements
- Parametric Fitting
- Contour Voting
- Gestalt Principles

# Review of Edge Elements

Last class we started discussing edges in more detail.

- For many applications, it is better to have a localized edge line rather than a gradient map

# Review of Edge Elements

Last class we started discussing edges in more detail.

- For many applications, it is better to have a localized edge line rather than a gradient map
- One way to better localize edges is through a Laplacian

# Review of Edge Elements

Last class we started discussing edges in more detail.

- For many applications, it is better to have a localized edge line rather than a gradient map
- One way to better localize edges is through a Laplacian
- The individual localized points responding to the Laplacian are *edge elements*

# Review of Edge Elements

Last class we started discussing edges in more detail.

- For many applications, it is better to have a localized edge line rather than a gradient map
- One way to better localize edges is through a Laplacian
- The individual localized points responding to the Laplacian are *edge elements*
- Let's demo the behaviour of a Laplacian filter...

# The Canny Edge Detector

A classic and still popular approach to edge detection was formalized by Canny (1986), which is now known as the Canny edge detector.

The Canny detector is based on many of the principles which we have already discussed, but also includes slightly more sophisticated localization techniques.

1. Smooth the image

# The Canny Edge Detector

A classic and still popular approach to edge detection was formalized by Canny (1986), which is now known as the Canny edge detector.

The Canny detector is based on many of the principles which we have already discussed, but also includes slightly more sophisticated localization techniques.

1. Smooth the image
2. Calculate the gradient of the image

# The Canny Edge Detector

A classic and still popular approach to edge detection was formalized by Canny (1986), which is now known as the Canny edge detector.

The Canny detector is based on many of the principles which we have already discussed, but also includes slightly more sophisticated localization techniques.

1. Smooth the image
2. Calculate the gradient of the image
    - Can be combined with the previous step

# The Canny Edge Detector

A classic and still popular approach to edge detection was formalized by Canny (1986), which is now known as the Canny edge detector.

The Canny detector is based on many of the principles which we have already discussed, but also includes slightly more sophisticated localization techniques.

1. Smooth the image
2. Calculate the gradient of the image
    - Can be combined with the previous step
    - Typically calculated as a combination of vertical, horizontal, and diagonal filters

# The Canny Edge Detector

A classic and still popular approach to edge detection was formalized by Canny (1986), which is now known as the Canny edge detector.

The Canny detector is based on many of the principles which we have already discussed, but also includes slightly more sophisticated localization techniques.

1. Smooth the image
2. Calculate the gradient of the image
    - Can be combined with the previous step
    - Typically calculated as a combination of vertical, horizontal, and diagonal filters
3. Apply non-maximum suppression along the gradient

# The Canny Edge Detector

A classic and still popular approach to edge detection was formalized by Canny (1986), which is now known as the Canny edge detector.

The Canny detector is based on many of the principles which we have already discussed, but also includes slightly more sophisticated localization techniques.

1. Smooth the image
2. Calculate the gradient of the image
   - Can be combined with the previous step
   - Typically calculated as a combination of vertical, horizontal, and diagonal filters
3. Apply non-maximum suppression along the gradient
4. Double-threshold - employ hysteresis when eliminating weak edges

# Hysteresis

*Hysteresis* is the dependence of a system's state on its past. It is applied in this case to the retention of edges with strength below the first higher threshold but above the second threshold (*weak edges*).

# Hysteresis

*Hysteresis* is the dependence of a system's state on its past. It is applied in this case to the retention of edges with strength below the first higher threshold but above the second threshold (*weak edges*).

Weak edges are retained if they share a neighbourhood with a strong edge, but are eliminated if they do not.

# Hysteresis

*Hysteresis* is the dependence of a system's state on its past. It is applied in this case to the retention of edges with strength below the first higher threshold but above the second threshold (*weak edges*).

Weak edges are retained if they share a neighbourhood with a strong edge, but are eliminated if they do not.

Let's look at a Canny demo.

# Contours

Even with more strongly localized edges, we are still essentially left with a map of disparate edge elements. To begin reasoning over them or characterizing their properties, we need to form relationships between the edge elements to form larger cohesive structures.

# Contours

Even with more strongly localized edges, we are still essentially left with a map of disparate edge elements. To begin reasoning over them or characterizing their properties, we need to form relationships between the edge elements to form larger cohesive structures.

The most general structure formed by stringing together edge elements is often referred to as a *contour*.

# Contours

Even with more strongly localized edges, we are still essentially left with a map of disparate edge elements. To begin reasoning over them or characterizing their properties, we need to form relationships between the edge elements to form larger cohesive structures.

The most general structure formed by stringing together edge elements is often referred to as a *contour*.

Contours are formed by taking edge elements and grouping them, usually through connected components with added rules. We can see an example with OpenCV's `findContours()` function.

# Reasoning with Contours

Once we have a set of contours, we can begin to work directly with them to reason over the content of the image or infer useful information, for example:

- Compute bounding boxes over objects with closed contours

# Reasoning with Contours

Once we have a set of contours, we can begin to work directly with them to reason over the content of the image or infer useful information, for example:

- Compute bounding boxes over objects with closed contours
- Encode additional logical steps to manipulate contours based on whole contour attributes, rather than kernel neighbourhoods

# Reasoning with Contours

Once we have a set of contours, we can begin to work directly with them to reason over the content of the image or infer useful information, for example:

- Compute bounding boxes over objects with closed contours
- Encode additional logical steps to manipulate contours based on whole contour attributes, rather than kernel neighbourhoods
  - Link contour segments even across gaps

# Reasoning with Contours

Once we have a set of contours, we can begin to work directly with them to reason over the content of the image or infer useful information, for example:

- Compute bounding boxes over objects with closed contours
- Encode additional logical steps to manipulate contours based on whole contour attributes, rather than kernel neighbourhoods
  - Link contour segments even across gaps
  - Eliminate isolated or topologically unsatisfying contour segments

# Parametric Segments

Often it is preferable to take another abstraction step over contour segments and fit a parametric curve rather than model it explicitly as a set of linked edge elements.

- More compact representations

# Parametric Segments

Often it is preferable to take another abstraction step over contour segments and fit a parametric curve rather than model it explicitly as a set of linked edge elements.

- More compact representations
- Allows straightforward applications of algebraic operations (*e.g.* calculate intersections)

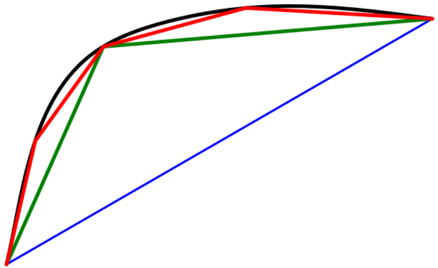
# Parametric Segments

Often it is preferable to take another abstraction step over contour segments and fit a parametric curve rather than model it explicitly as a set of linked edge elements.

- More compact representations
- Allows straightforward applications of algebraic operations (*e.g.* calculate intersections)
- Can sometimes eliminate small quantities of noise

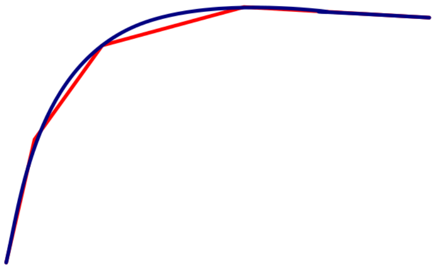
# Piecewise Linear Fitting

Approximating a curve with an increasing number of small linear segments (each increase is done by finding points on the curve furthest from the current approximation).



# Splines

Instead of piece-wise linear approximations, we can instead utilize piecewise polynomial fitting (splines).



# Linking Contour Segments

As we saw with our example penguin, often we must deal with missing "chunks" of an edge, and it would be beneficial to link together contour segments which follow the same overall curve even if they have a gap in the edge image.

# Linking Contour Segments

As we saw with our example penguin, often we must deal with missing "chunks" of an edge, and it would be beneficial to link together contour segments which follow the same overall curve even if they have a gap in the edge image.

This is a common enough problem that a number of standard techniques have been developed to approach this problem.

# Linking Contour Segments

As we saw with our example penguin, often we must deal with missing "chunks" of an edge, and it would be beneficial to link together contour segments which follow the same overall curve even if they have a gap in the edge image.

This is a common enough problem that a number of standard techniques have been developed to approach this problem.

- Random sample consensus (RANSAC)

# Linking Contour Segments

As we saw with our example penguin, often we must deal with missing "chunks" of an edge, and it would be beneficial to link together contour segments which follow the same overall curve even if they have a gap in the edge image.

This is a common enough problem that a number of standard techniques have been developed to approach this problem.

- Random sample consensus (RANSAC)
- Hough Transform

# The Challenge of Outliers

When trying to link up disparate contour segments, we often face the challenge of spurious segments (either because they more properly belong to another grouping, or because they are an edge element based on noise).

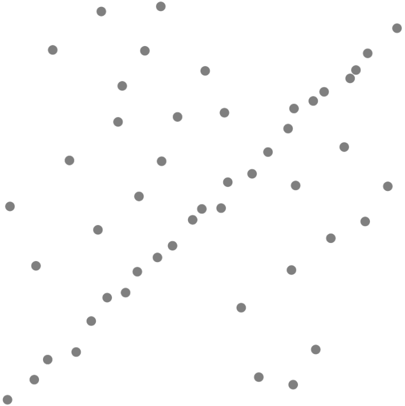


Image source: Wikipedia

# Random Sampling

RANSAC runs a number of trials, each time sampling a minimal set of data to fit a parametric model (such as a line).

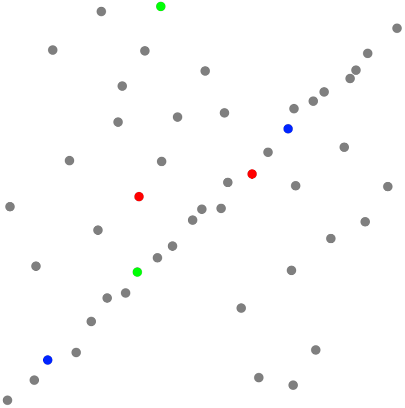


Image source: Modified from Wikipedia

# Sample Voting

For each random sample, the fitted model is tested to see how much of the data it explains.

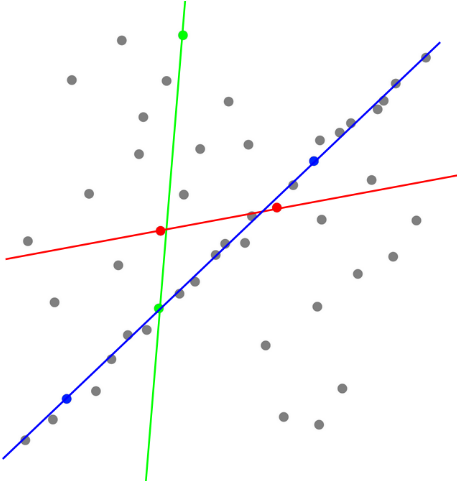


Image source: Modified from Wikipedia

# Fit Selection

The fit which best explains the majority of data is accepted. Only data which is consistent with this fit (within an allowable error threshold) is considered part of the final fit; all other data is considered an outlier.

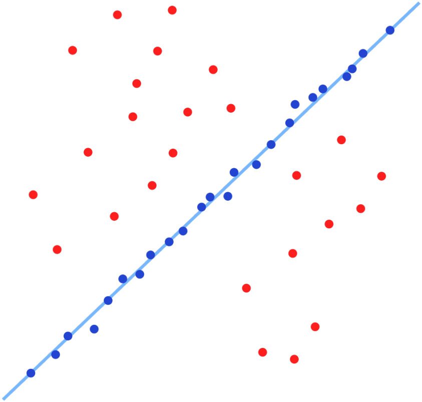


Image source:Wikipedia

# Fair Voting

One drawback to RANSAC is that it is stochastic, and only guaranteed to provide a good solution with a certain probability based on the number of random sample steps taken. A deterministic alternative is the *Hough transform*.

# Fair Voting

One drawback to RANSAC is that it is stochastic, and only guaranteed to provide a good solution with a certain probability based on the number of random sample steps taken. A deterministic alternative is the *Hough transform*.

In the Hough Transform, edge elements vote for the lines which they provide evidence for, and these votes are tallied in an *accumulator space*.

# Fair Voting

One drawback to RANSAC is that it is stochastic, and only guaranteed to provide a good solution with a certain probability based on the number of random sample steps taken. A deterministic alternative is the *Hough transform*.

In the Hough Transform, edge elements vote for the lines which they provide evidence for, and these votes are tallied in an *accumulator space*.

If gradient information is retained to assign an orientation to a given edge element, then voting can be restricted to elements in accumulator space which are consistent with a given orientation. Without orientation, edge elements typically vote for all possible oriented lines passing through them, leading to a significant increase in computational requirements.

# Accumulator Space

Typically, the accumulator space is a large array corresponding to parameter values characterizing instances of the target shape.

- For lines, this is typically implemented in $\rho$-$\theta$ coordinates

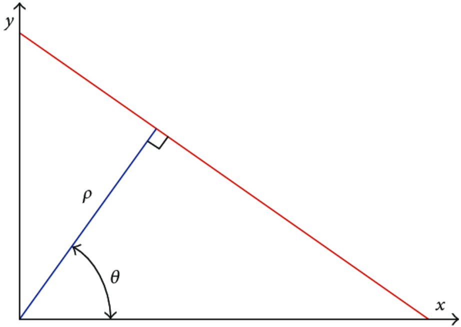



Image source: Original source unknown

# Accumulator Space

Typically, the accumulator space is a large array corresponding to parameter values characterizing instances of the target shape.

- For lines, this is typically implemented in $\rho$-$\theta$ coordinates
- Accumulator space can be very large for pixel-wise accuracy

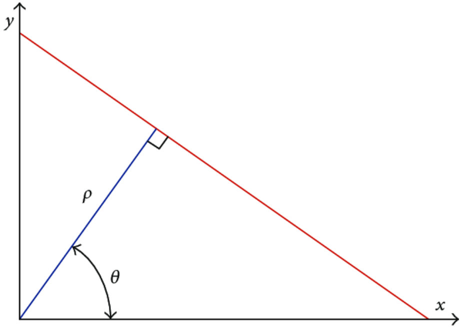


Image source: Original source unknown

# Accumulator Space

Typically, the accumulator space is a large array corresponding to parameter values characterizing instances of the target shape.

- For lines, this is typically implemented in $\rho$-$\theta$ coordinates
- Accumulator space can be very large for pixel-wise accuracy
- Often accumulator space is implemented over a coarse spatial scale, and then fit is refined

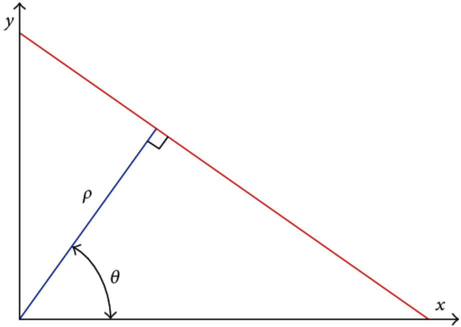



Image source: Original source unknown

# Hough Transforms for All

Although originally developed for line detection, Hough transforms can be readily adapted to arbitrary parametric shapes, and are particularly popular for circles and ellipses.

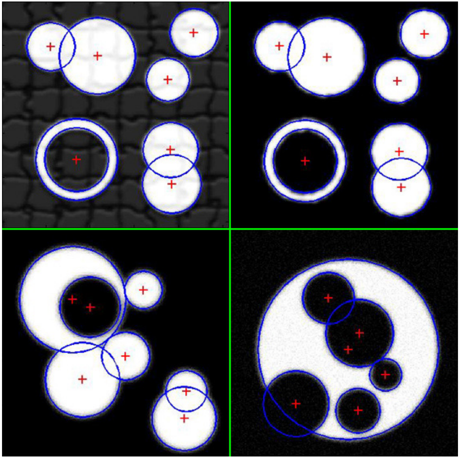


Image source: Peng, 2010

# Hough Transforms for All

Although originally developed for line detection, Hough transforms can be readily adapted to arbitrary parametric shapes, and are particularly popular for circles and ellipses.

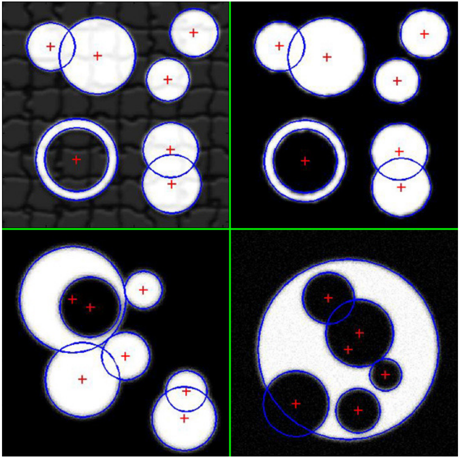


Image source: Peng, 2010

What changes about the accumulator space for a circle transform instead of a line?

# Hough Transforms Example

Go to Hough Transform example.

# Useful Tool for Constrained Applications

If you know what sorts of curves and lines you should find in an image, these tools can be very straightforward to implement and execute while still achieving satisfactory results. Examples of applications for which these would be well suited include, especially for situations with minimal training data:

- Scanned form analysis

# Useful Tool for Constrained Applications

If you know what sorts of curves and lines you should find in an image, these tools can be very straightforward to implement and execute while still achieving satisfactory results. Examples of applications for which these would be well suited include, especially for situations with minimal training data:

- Scanned form analysis
- Counting repetitive objects

# Useful Tool for Constrained Applications

If you know what sorts of curves and lines you should find in an image, these tools can be very straightforward to implement and execute while still achieving satisfactory results. Examples of applications for which these would be well suited include, especially for situations with minimal training data:

- Scanned form analysis
- Counting repetitive objects
- Detecting reliable structures (field lines in sports, windows or building contours)

# Overarching Principles

The heuristics and rules we have covered so far for how to select and group contours and fit lines and curves to those contours are drawn from a set of psychological principles known as *Gestalt Principles* (or *Gestalt Laws* or *principles of grouping*).

These general rules do not fit all possible situations in visual perception, but they do give us a nice starting point when reasoning over lines, curves, and contours (or even other features, like patches).

# Proximity

- Items which are close by tend to be interpreted as forming a group

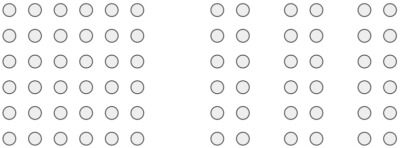


Image source:Wikipedia

# Proximity

- Items which are close by tend to be interpreted as forming a group
- Items can be grouped by spatial proximity even if highly different in appearance (some caveats - see *similarity*)

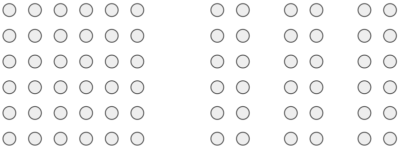


Image source:Wikipedia

# Proximity

- Items which are close by tend to be interpreted as forming a group
- Items can be grouped by spatial proximity even if highly different in appearance (some caveats - see *similarity*)
- Grouped elements can create the illusion of shapes, even when not touching

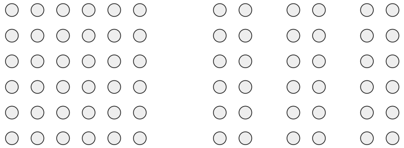


Image source:Wikipedia

# Similarity

- Items which are similar in appearance are more likely to be grouped together than objects which are different in appearance

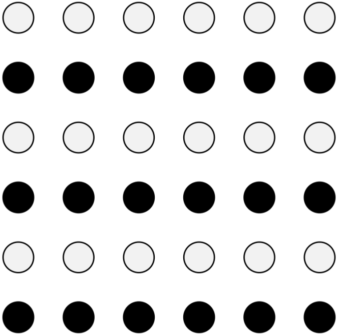


Image source:Wikipedia

# Similarity

- Items which are similar in appearance are more likely to be grouped together than objects which are different in appearance
- Can work in tandem or be in conflict with *proximity*

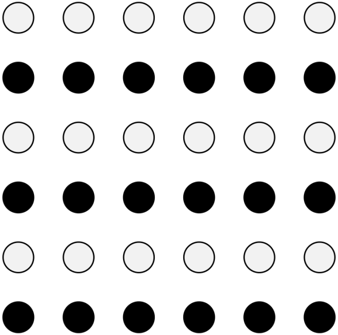


Image source:Wikipedia

# Similarity

- Items which are similar in appearance are more likely to be grouped together than objects which are different in appearance
- Can work in tandem or be in conflict with *proximity*
- Can operate along disparate feature channels (*e.g.* colour, texture, shape)

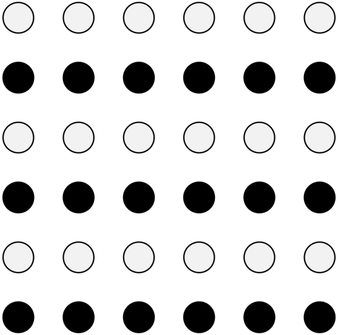


Image source:Wikipedia

# Closure

- Despite gaps, we tend to group edges into complete, closed contours

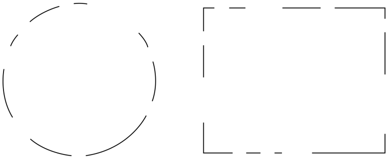


Image source:Wikipedia

# Closure

- Despite gaps, we tend to group edges into complete, closed contours
- Heavily informed by *border ownership*, which will be discussed later

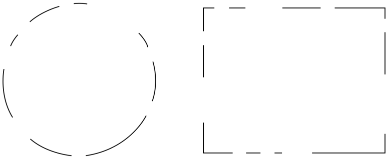


Image source:Wikipedia

# Closure

- Despite gaps, we tend to group edges into complete, closed contours
- Heavily informed by *border ownership*, which will be discussed later
- Influenced by surrounding context, semantic knowledge, strength of existing edges, and size of the gaps

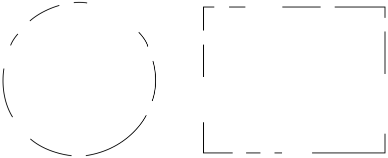


Image source:Wikipedia

# Continuation

- Perceptual preference is given to items being uninterrupted, cohesive objects

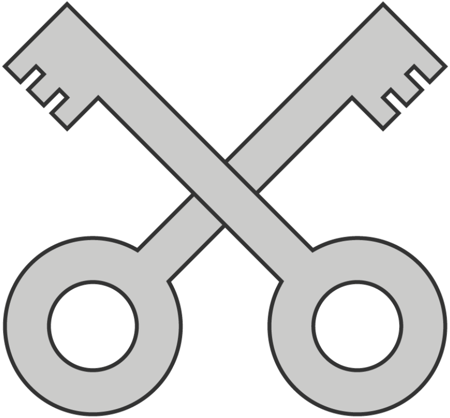


Image source:Wikipedia

# Continuation

- Perceptual preference is given to items being uninterrupted, cohesive objects
- Curves and lines tend to follow an established direction, rather than sharp or abrupt changes

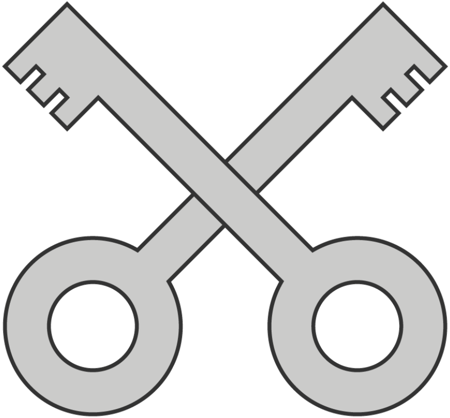


Image source:Wikipedia

# Continuation

- Perceptual preference is given to items being uninterrupted, cohesive objects
- Curves and lines tend to follow an established direction, rather than sharp or abrupt changes
- When abrupt changes do occur, they tend to indicate occlusions or boundaries

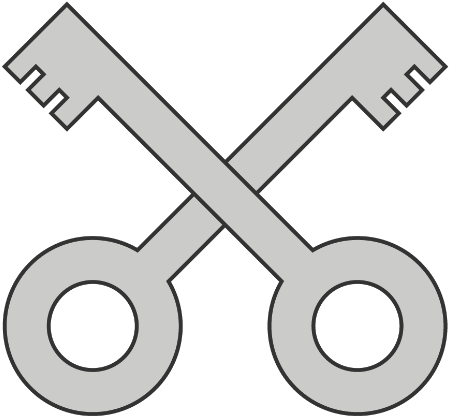


Image source:Wikipedia