Medical Image Processing

ITK Insight Toolkit

Carlos A. Vinhais

cvinhais@gmail.com

ISEP – Departamento de Física
Instituto Superior de Engenharia do Porto
Porto, Portugal
Medical Image Processing
ITK Insight Toolkit

- Introduction to ITK
- Data Representation
- Image IO
- Image Iterators
- Filtering
- Segmentation
- Registration
Introduction
ITK Insight Toolkit

Introduction

ITK
The Insight Segmentation and Registration Toolkit

- Image Input/Output (IO)
- Image Filtering
- Image Segmentation
- Image Registration
- No visualization
- No GUI
  (Graphical User Interface)
ITK Insight Toolkit
Introduction

Welcome to the National Library of Medicine Insight Segmentation and Registration Toolkit (ITK). ITK is an open-source, cross-platform system that provides developers with an extensive suite of software tools for image analysis. Developed through extreme programming methodologies, ITK employs leading-edge algorithms for registering and segmenting multidimensional data. The goals for ITK include:

- Supporting the Visible Human Project.
- Establishing a foundation for future research.
- Creating a repository of fundamental algorithms.
- Developing a platform for advanced product development.
- Support commercial application of the technology.
- Create conventions for future work.
- Grow a self-sustaining community of software users and developers.

ITK provides leading-edge segmentation and registration algorithms in two, three, and more dimensions; it is distributed as an open-source software package.

(www.itk.org)
The ITK Software Guide Book 1
Introduction and Development Guidelines
H.J. Johnson, M.M. McCormick, L. Ibanez

The ITK Software Guide Book 2
Design and Functionality
H.J. Johnson, M.M. McCormick, L. Ibanez

Freely available for download:
InsightSoftwareGuide-Book1-4.9.0.pdf
InsightSoftwareGuide-Book2-4.9.0.pdf
C++ Programming

- Install C++ Compiler/IDE
- Install CMake
- Download ITK source
- Build ITK with CMake
- Build C++ Exercises with CMake
- Run C++ Exercises
Python Programming

- Install Python(x,y) (with itk plugin selected)
- Run Spyder IDE
- Run Python Exercises
ITK Insight Toolkit
Software ITK Snap

(www.itksnap.org)
C++ Exercises

- HelloWorld.zip
- myProject.zip
Data Representation
Data Representation

Overview

- Digital Image
- Pixels and Voxels
- Sampling and Quantization
- ITK Image
- File Formats and Pixel Types
- Pixel Neighborhood
- Pixel Connectivity
- Distance Metrics
Data Representation

Digital Image
Data Representation

Pixels and Voxels

**Picture element**

**Volume element**
Data Representation
Sampling and Quantization

Spatial resolution

Pixel intensity
Data Representation
Sampling and Quantization

resolution decreases

256x256 → 128x128 → 64x64 → 32x32
Data Representation
Sampling and Quantization

Image of constant size: 472 × 374

Number of intensity levels:

\[ L = 2^k \]

\[ k = \text{no of bits/pixel} \]
Figure 4.1: Geometrical concepts associated with the ITK image.
Data Representation

ITK Image

**Pixel Coordinates**
- pixel Index: \([i, j, k]\)
- image Region
  - start \([0, 0, 0]\)
  - size \([N_x, N_y, N_z]\)
- pixel Type
  - pixel Intensity

**Physical Coordinates**
- point coord.: \((x, y, z)\)
- coord. system \(OXYZ\)
  - origin \((x_0, y_0, z_0)\)
  - spacing \((dX, dY, dZ)\)
- Image Extent
  - \((\Delta X, \Delta Y, \Delta Z)\)
Data Representation

ITK Image

Physical units: mm

Coordinate Transformation:
point \((x, y, z) \leftrightarrow \) pixel \([i, j, k]\)

\[
\begin{align*}
    x &= x_O + i \cdot dX \\
    y &= y_O + j \cdot dY \\
    z &= z_O + k \cdot dZ
\end{align*}
\]

ITK Classes:

\begin{itemize}
    \item \texttt{itk::TransformIndexToPhysicalPoint}(index, point)
    \item \texttt{itk::TransformPhysicalPointToIndex}(point, index)
\end{itemize}
Data Representation
File Formats and Pixel Types

- **BMP (2D only)**
  - unsigned char
  - RGB

- **PNG (2D)**
  - unsigned char
  - unsigned short
  - RGB

- **TIFF (2D only)**
  - unsigned char
  - unsigned short
  - RGB

- **JPEG (2D only)**
  - unsigned char

- **GDCM**
  - unsigned char
  - char
  - unsigned short
  - short
  - unsigned int
  - Int
  - double
Data Representation
File Formats and Pixel Types

- **DICOM**
  - float
  - char
  - unsigned char
  - short
  - unsigned short
  - RGB (char)
  - RGB (short)

- **VTK**
  - float
  - double
  - unsigned char
  - char
  - unsigned short
  - short
  - unsigned int
  - int
  - unsigned long
  - long
  - RGB
The ** itk::Image** class can be templated over virtually any pixel type, however

- not all file formats support all data types for reading and writing. In some cases, it may be necessary to add an ** itk::CastImageFilter** to convert the output to a pixel format appropriate for the target file.

- It is important not to truncate the data by converting to a smaller type (e.g. short $\rightarrow$ char). In this case, the ** itk::RescaleIntensityImageFilter** can be used before casting.
# Data Representation

## Pixel Neighborhood

### 2D Neighborhood

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(-1, -1)</td>
<td>(0, -1)</td>
<td>(1, -1)</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>(-1, 0)</td>
<td>(0, 0)</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>(-1, 1)</td>
<td>(0, 1)</td>
<td>(1, 1)</td>
</tr>
</tbody>
</table>

radius = [1, 1]
size = [3, 3]
### 2D Neighborhood Shapes

<table>
<thead>
<tr>
<th>Pixel</th>
<th>radius</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (-3, -1)</td>
<td>[3, 1]</td>
<td>[7, 3]</td>
</tr>
<tr>
<td>1 (-2, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 (-1, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 (0, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (1, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (2, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 (3, -1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pixel</th>
<th>radius</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (-3, 0)</td>
<td>[3, 1]</td>
<td>[7, 3]</td>
</tr>
<tr>
<td>8 (-2, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 (-1, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 (0, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 (1, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 (2, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 (3, 0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pixel</th>
<th>radius</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 (-3, 1)</td>
<td>[3, 1]</td>
<td>[7, 3]</td>
</tr>
<tr>
<td>15 (-2, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 (-1, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 (0, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 (1, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 (2, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 (3, 1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pixel</th>
<th>radius</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (-1, -2)</td>
<td>[0, 2]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>1 (0, -2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 (1, -2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 (-1, -1)</td>
<td>[0, 2]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>4 (0, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (1, -1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 (-1, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 (0, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 (1, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 (-1, 1)</td>
<td>[0, 2]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>10 (0, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 (1, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 (-1, 2)</td>
<td>[0, 2]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>13 (0, 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 (1, 2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(radius = [3, 1], size = [7, 3])
Data Representation
Pixel Connectivity

2D Connectivity

4–connected

8–connected
How many CONNECTED objects?

2 OBJECTS
(4-connected)

or

1 OBJECT
(8-connected)
3D Connectivity

6–connected  10–connected  26–connected
Distance between points/pixels $p(x, y)$ and $q(s, t)$

- **Euclidian**
  $$D_e(p, q) = \sqrt{(x - s)^2 + (y - t)^2}$$

- **Cityblock**
  $$D_4(p, q) = |x - s| + |y - t|$$

- **Chessboard**
  $$D_8(p, q) = \max(|x - s|, |y - t|)$$
## Distance Metrics

<table>
<thead>
<tr>
<th>Distance Metric</th>
<th>Description</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>The Euclidean distance is the straight-line distance between two pixels.</td>
<td><img src="image1.png" alt="Image" /> <img src="distance_transform1.png" alt="Distance transform" /></td>
</tr>
<tr>
<td>City Block</td>
<td>The city block distance metric measures the path between the pixels based on a 4-connected neighborhood. Pixels whose edges touch are 1 unit apart; pixels diagonally touching are 2 units apart.</td>
<td><img src="image2.png" alt="Image" /> <img src="distance_transform2.png" alt="Distance transform" /></td>
</tr>
<tr>
<td>Chessboard</td>
<td>The chessboard distance metric measures the path between the pixels based on an 8-connected neighborhood. Pixels whose edges or corners touch are 1 unit apart.</td>
<td><img src="image3.png" alt="Image" /> <img src="distance_transform3.png" alt="Distance transform" /></td>
</tr>
</tbody>
</table>
**GetPixel()** and **SetPixel()** are very inefficient for accessing pixel data.

These methods should only be used for debugging or for supporting interactions like querying pixel values by mouse clicking.

**Image iterators** should be used when massive access to pixel data is required.
C++ Exercises

- CreatelImages.zip

Python Exercises

1. Create2DImage.py
2. Create3DImage.py
Image IO
Overview

- Reading and Writing Images
- Region of Interest Extraction
- Slice Extraction from Volume
- DICOM Images and Header (metadata)
Image IO
ITK Computer Exercises

### C++ Exercises

- ReadRGBWritePNG.zip
- DicomReadPrintTagsWrite1.zip
- DicomReadPrintTagsWrite2.zip
- DicomSerieReadPrintTagsWrite.zip
- ImageReadImageSeriesWrite.zip

### Python Exercises

1. Read2DImageWrite.py
2. Read3DImageWrite.py
3. ExtractRegionOfInterest.py
4. ExtractSliceFromVolume.py
5. GenerateSlicesFromVolume.py
6. DicomSerieReadPrintTagsWrite.py
Image Iterators
Image Iterators
Overview

ITK Image Iterators
(non-constant, constant, with index)

- Linear
- Slice
- Region
- Neighborhood


```cpp
#include <vector>

typedef std::vector<float> VectorType;

VectorType myVector;

myVector.push_back( 4 );
myVector.push_back( 7 );
myVector.push_back( 19 );

VectorType::const_iterator itr = myVector.begin();

while( itr != myVector.end() )
{
    std::cout << *itr << std::endl;
    ++itr;
}
```
 Nested loops?

```c++
for( unsigned int z = 0; z < Nz; z++ )
{
    for( unsigned int y = 0; y < Ny; y++ )
    {
        for( unsigned int x = 0; x < Nx; x++ )
        {
            image[z][y][x] = ... // pixel access
        }
    }
}
```

How to generalize this code to 2D, 3D, 4D, ...?
Image Iterators
Region Iterators

ITK Region Iterator

Moving Iterator
- GoToBegin()
- GoToEnd()
- IsAtEnd()
- IsAtBegin()
- ++
- -
- SetPosition(index)
- GetIndex()

Image data access
- Get()
- Set(value)
Image Iterators

Region Iterators

ITK Region Iterator (const.)

```cpp
#include "itkImage.h"
#include "itkImageRegionIterator.h"

typedef itk::Image<signed short, 3> ImageType;
typedef itk::ImageRegionConstIterator<ImageType> IteratorType;

ImageType::ConstPointer image = GetConstImageSomeHow();
ImageType::RegionType region = image->GetLargestPossibleRegion();

IteratorType itr(image, region);

itr.GoToBegin();
while (!itr.IsAtEnd())
{
    std::cout << itr.Get() << std::endl;
    ++itr;
}
```
ITK Region Iterator (non-const.)

```
#include " itkImage.h"
#include " itkImageRegionIterator.h"

typedef itk::Image< signed short , 3 > ImageType;
typedef itk::ImageRegionIterator< ImageType > IteratorType;

ImageType::ConstPointer image = GetNonConstImageSomeHow();
ImageType::RegionType region = image->GetLargestPossibleRegion();

IteratorType itr( image , region );

itr.GoToBegin();
while( !itr.IsAtEnd() )
{
    itr.Set( 0 );
    ++itr;
}
```
Image Iterators

Linear Iterators

**ITK Linear Iterator (with Index)**

```cpp
#include "itkImage.h"
#include "itkImageLinearIteratorWithIndex.h"

typedef itk::Image< signed short, 3 > ImageType;
typedef itk::ImageLinearConstIteratorWithIndex< ImageType > IteratorType;

ImageType::ConstPointer image = GetConstImageSomeHow();
ImageType::RegionType region = GetImageRegionSomeHow();

IteratorType itr( image, region );

for( itr.GoToBegin(); !itr.IsAtEnd(); itr.NextLine() )
{
    while( !itr.IsAtEndOfLine() )
    {
        std::cout << itr.Get() << " , " << itr.GetIndex() << std::endl;
        // ...
        ++itr;
    }
    // ...
}
```

(Carlos A. Vinhais)

ITK Insight Toolkit

March, 2017 43 / 133
Image Iterators
Slice Iterators

ITK Slice Iterator (with Index)

```cpp
#include "itkImage.h"
#include "itkImageSliceIteratorWithIndex.h"

typedef itk::Image< signed short, 3 > ImageType;
typedef itk::ImageSliceConstIteratorWithIndex < ImageType > IteratorType;

ImageType::ConstPointer image = GetConstImageSomeHow();
ImageType::RegionType region = GetImageRegionSomeHow();

IteratorType itr( image, region );

for( itr.GoToBegin(); !itr.IsAtEnd(); itr.NextSlice() )
{
    for( ; !itr.IsAtEndOfSlice(); itr.NextLine() )
    {
        for( ; !itr.IsAtEndOfLine(); ++itr )
        {
            std::cout << itr.Get() << ", " << itr.GetIndex() << std::endl;
            // ...
        }
        // ...
    }
    // ...
}
```

(Carlos A. Vinhais)
ITK Insight Toolkit
March, 2017 44 / 133
Image Iterators

Neighborhood Iterators

ITK Neighborhood Iterator

Neighborhood
- GetRadius()
- Size()
- GetNeighborhood()
- GetIndex()

Image data access
- GetPixel( offset )
- GetCenterPixel()
- GetNext( dir )
- GetPrevious( dir )
- SetPixel( i, value )
- SetCenterPixel( value )
- SetNext( dir, value )
- SetPrevious( dir, value )
ITK Neighborhood Iterator

```cpp
#include "itkImage.h"
#include "itkConstNeighborhoodIterator.h"

typedef itk::Image< signed short , 3 > ImageType;
typedef itk::ConstNeighborhoodIterator< ImageType > IteratorType;

ImageType::ConstPointer image = GetConstImageSomeHow();
ImageType::RegionType region = GetImageRegionSomeHow();

IteratorType itr( image, region );

itr.GoToBegin();

while( !itr.IsAtEnd() )
{
    for ( int i = 0; i < itr.Size(); i++ )
    {
        value = itr.GetElement( i );
        // ...
    }
    // ...
    ++itr;
}
```
MIP - Maximum Intensity Projection

leon25.vtk (3D image, slice 87)
axial plane OXY

MIP along OZ axis (2D)
(projection direction = 2)
MIP - Maximum Intensity Projection

leon25.vtk (3D image, slice 256) coronal plane OXZ

MIP along OY axis (2D) (projection direction = 1)
MIP - Maximum Intensity Projection

Leon25.vtk (3D, slice 256)
sagittal plane OYZ

MIP along OX axis (2D)
(projection direction = 0)
C++ Exercises

- MIP.zip
Filtering
Filtering
Overview

- Thresholding
- Intensity Mapping
- Edges and Derivatives
- Neighborhood Filters
- Mathematical Morphology
- Logical and Arithmetic Operations
- Smoothing
Filtering
Thresholding

Image Histogram
Binary Thresholding

- The thresholding operation is used to change or identify pixel values based on specifying one or more threshold values.

- This filter transforms the input image into a binary image.

\[
I_1(x, y) = \begin{cases} 
\text{insideValue}, & \text{if } T_{\text{lower}} < I_0(x, y) < T_{\text{upper}} \\
\text{outsideValue}, & \text{otherwise}
\end{cases}
\]
Filtering

Thresholding

Binary Threshold

cthead1.png (UC, 2D)

\[ T_{lower} = 100 \]
\[ T_{upper} = 255 \]

\[ T_{lower} = 200 \]
\[ T_{upper} = 255 \]
General Thresholding
(Threshold Below)

\[ I_1(x, y) = \begin{cases} 
\text{outsideValue}, & \text{if } T_{below} < I_0(x, y) \\
I_0(x, y), & \text{otherwise}
\end{cases} \]
General Thresholding
(Threshold Above)

\[
I_1(x, y) = \begin{cases} 
\text{outsideValue}, & \text{if } I_0(x, y) > T_{above} \\
I_0(x, y), & \text{otherwise}
\end{cases}
\]
General Thresholding
(Threshold Outside)

\[ I_1(x, y) = \begin{cases} I_0(x, y), & \text{if } T_{\text{lower}} < I_0(x, y) < T_{\text{upper}} \\ \text{outsideValue}, & \text{otherwise} \end{cases} \]
Filtering
Thresholding

General Threshold

cthead1.png (UC, 2D)

ThresholdBelow
\( T_{lower} = 100 \)

ThresholdOutside
\( T_{lower} = 100, \ T_{upper} = 200 \)
Intensity Mapping

- Used to pixel type-conversions required in the data pipeline.

- It is up to the user to anticipate the pixel type-conversions required in the data pipeline.

- In medical imaging applications it is usually not desirable to use a general pixel type, since this may result in the loss of valuable information.
Linear Intensity Mapping
(ITK Filters)

- itk::CastImageFilter
- itk::RescaleIntensityImageFilter
- itk::ShiftScaleImageFilter
- itk::NormalizeImageFilter
Non Linear Mapping
(sigmoid filter)

- the **sigmoid filter** maps a specific range of intensity values into a new intensity range, by making a *very smooth* and *continuous transition* in the borders of the range.

- ITK Implementation: the filter includes 4 *parameters* that can be tuned to select the input and output intensity ranges.

\[ I' = (Max - Min) \cdot \frac{1}{1 + e^{-\frac{(I - Min)}{\alpha}}} + Min \]
Sigmoid Filter

\((alpha\ \text{parameter})\)
Sigmoid Filter
(*beta* parameter)
Filtering

Intensity Mapping

Sigmoid

cthead1.png
(UC, 2D)

$\alpha = 30$
$\beta = 128$

$\alpha = -30$
$\beta = 128$
Edges

Model of an ideal digital edge

Model of a ramp digital edge

Gray-level profile of a horizontal line through the image

Gray-level profile of a horizontal line through the image
Edges (cont.)
Canny Edge Detection

- Widely used for edge detection

- Optimal solution satisfying the constraints of:
  - edge sensitivity
  - edge localization
  - noise robustness

- This filter operates on image of pixel type float.
Filtering
Edges and Derivatives

Canny Edge Detection

cthead1.png (UC, 2D)

CannyEdge $\sigma = 1$

CannyEdge $\sigma = 10$
Filtering
Edges and Derivatives

Image Gradient
(First Order Derivatives)

\[ \nabla f(x, y) = \left[ \frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right] = [G_x, G_y] \]

- Magnitude and direction
- Approximations

\[
G = \sqrt{G_x^2 + G_y^2}
\]

\[
\theta = \tan^{-1} \frac{G_y}{G_x}
\]

\[
G \approx |G_x| + |G_y|
\]

\[
G \approx \max(|G_x|, |G_y|)
\]
Filtering
Edges and Derivatives

**Image Gradient**
(Edge Detection Kernels)

- **Roberts**
  \[ G_x \approx \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad G_y \approx \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \]

- **Prewitt**
  \[ G_x \approx \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y \approx \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \]

- **Sobel**
  \[ G_x \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y \approx \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \]
ITK Gradient Magnitude

ITK computes the magnitude of the image gradient at each pixel location using a simple finite differences approach:

\[ G_x \approx f(x+1, y) - f(x-1, y) \]
\[ G_y \approx f(x, y+1) - f(x, y-1) \]

Equivalent to convolving of the image with masks . . .

...then computing the gradient magnitude as:

\[ G = \sqrt{G_x^2 + G_y^2} \]
Filtering
Edges and Derivatives

ITK Gradient Magnitude

cthead1.png
(UC, 2D)

gradient magnitude
Filtering
Edges and Derivatives

Laplacian Filters
(Second Order Derivatives)

- Laplacian \( L(x, y) \) of an image with pixel intensity values \( I(x, y) \)

\[
L(x, y) = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2}
\]

- Discrete convolution kernels that can approximate the second derivatives:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>8</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
Neighborhood Filters

Examples

- **Mean** Filter (local operator)

![Mean Filter Diagram]

- **Median** Filter (order filter)

![Median Filter Diagram]
Filtering
Neighborhood Filters

Mean vs Median

cthead1.png
(UC, 2D)

mean
radius = [1,1]

median
radius = [1,1]
Filtering
Neighborhood Filters

**Mean vs Median**

cthead1.png (UC, 2D)

mean
radius = [5,5]

median
radius = [5,5]
Mathematical Morphology

- Morphological Filters
  (Binary and Grayscale Image Filters)
  - Dilation
  - Erosion
- Closing
- Opening
Binary Dilation (pipeline)
Filtering
Mathematical Morphology

Binary Dilation

binary
(UC, 2D)

dilation
s.e. radius = 1
Binary Closing
(pipeline)
Filtering
Mathematical Morphology

Binary Closing

binary
(UC, 2D)
dilation
s.e. radius = 1
closing
s.e. radius = 1
Logical and Arithmetic Operations

- Logical Operations:
  - And, Not, ...

- Arithmetic Operations:
  - Add, Subtract, ...

- Basic Algorithms:
  - Boundary Extraction
  - Morphological Gradient
Boundary Extraction
(pipeline)
Boundary Extraction

binary (UC, 2D)

boundary
s.e. radius = 1

boundary
s.e. radius = 5
Morphological Gradient
(pipeline)
Morphological Gradient

cthead1.png (UC, 2D)
morphological gradient s.e. radius = 1
morphological gradient s.e. radius = 5
Gradient Magnitude vs Morphological Gradient

cthead1.png (UC, 2D)  gradient magnitude (Sobel)  morphological gradient s.e. radius = 1
Filtering
ITK Computer Exercises

C++ Exercises

- BinaryThreshold.zip
- GeneralThreshold.zip
- Sigmoid.zip
- Rescale.zip
- CannyEdge.zip
- GradientMagnitude.zip
- Laplacian.zip
- Mean.zip
- Median.zip

- BinaryDilate.zip
- BinaryClose.zip
- BoundaryExtraction.zip
- IJ_MorphologicalGradient.zip
- IJ_LogicalOperationExtra.zip
- SmoothingRecursiveGaussian.zip
- GradientAnisotropicDiffusion.zip
- CurvatureFlow.zip
- Bilateral.zip
- DerivativeRecursiveGaussian.zip

(Carlos A. Vinhais)
Python Exercises

1. BinaryThreshold.py
2. Threshold.py
3. RescaleIntensity.py
4. Sigmoid.py
5. GradientMagnitude.py
6. MeanMedian.py
7. BinaryDilate.py
8. BinaryClose.py
9. BoundaryExtraction.py
10. MorphologicalGradient.py
Segmentation
Segmentation
Overview

- Optimal Thresholding
- Seeded Region Growing
- Connected Component Analysis
Optimal Thresholding

Automated selection of *optimal* threshold value(s), under some criteria, for separating objects from background by histogram analysis.
Optimal Thresholding

- Local maxima of the histogram correspond to objects of interest in the image scene
- Threshold values are best placed at local minima of the histogram to separate such objects
Iterative Thresholding
(ISODATA Clustering)

- iterative unsupervised clustering method
- distances between the cluster centers and the members of the cluster are minimized
- fixed number of clusters to be detected must be initially provided

\[ T^{(i+1)} = \frac{\bar{H}_1^{(i)} + \bar{H}_2^{(i)}}{2} \]
Segmentation

Optimal Thresholding

Otsu Thresholding

*Finds a threshold that classifies the image into two clusters such that we minimize the area under the histogram for one cluster that lies on the other cluster's side of the threshold*

- minimize the within class variance
- maximize the between class variance of foreground and background pixels

\[
T^* = \arg \min_T \left\{ \sigma_b^2 (T) \right\}
\]

\[
\sigma_b^2 (T) = n_1(T) n_2(T) \left[ \bar{H}_2(T) - \bar{H}_1(T) \right]^2
\]
Otsu Thresholding

cthead1.png (UC, 2D)  Otsu Threshold: $T = 84$ (rescaled)  NOT filter (rescaled)
Multiple Otsu Thresholding

cthead1.png (UC, 2D)

Otsu Thresholds:
\[ T_0 = 73, \ T_1 = 188 \]
Robust Thresholding

- Fast and noise robust automatic threshold selection method based on gradients in the image

- The basic idea is to choose the threshold at the intensity where the gradient are the highest

\[
T = \frac{\sum_{p \in D} G(p) \cdot I(p)}{\sum_{p \in D} G(p)}
\]

I = input image
G = gradient of I
Robust Thresholding

- same method with a power of the gradient gives better results with noisy images, by giving a higher weight to the high gradients.

\[ T = \frac{\sum_{p \in D} G(p)^m \cdot I(p)}{\sum_{p \in D} G(p)^m} \]

- m greater than 1, typically 2.
Robust Thresholding


Seeded Region Growing

- Neighborhood Iterators
- Connected Threshold
- Isolated Connected
- Confidence Connected
Segmentation
Seeded Region Growing

Neighborhood Iterators

Pixel Connectivity

Example (NeighborhoodIterators6)
Limitations of histogram based segmentation:

- the image histogram does not allow to recover spatial information, but only pixel intensities or gray-levels
- Image segmentation based on the histogram does not explore the fact that neighbor pixels can have similar intensity values

Alternative:
- image partition in sub-regions $R_1, R_2, \ldots, R_N$
Image Partition

Conditions:

\[ \bigcup_{i=1}^{N} R_i = R \]

\( R_i \) is connected, \( \forall i \)

\( R_i \cap R_j = \emptyset, \quad \forall i, j, \quad i \neq j \)

\( P(R_i) = \text{true}, \quad \forall i \)

\( P(R_i \cup R_j) = \text{false}, \quad \forall i, j, \quad i \neq j \)

\( P(R_i) = \) property of the pixels that belong to region \( R_i \) must satisfy:
Seeded Region Growing

Algorithm

- From the image, choose or find initial points (pixels) called seeds, that satisfy some criteria:
  
  e.g. \( P \geq T \) (local maximum)

- Join to an existing region, the neighbor pixels that are connected to that region, if they satisfy some criteria:
  
  e.g. \( P \geq \text{mean}(R) - 2 \)

- Stop the algorithm if the grow of the existing region is no more possible

- Consider the remaining points as new regions, coincident with the connected regions
Segmentation

Seeded Region Growing

Seeded Region Growing

Example
Connected Threshold

A simple similarity criterion for including connected pixels in the region is based on their gray level:

- "pixels having gray level \( I(x,y,z) \) in the same interval receive the same class label"

- \( T_L \) and \( T_U \) are the lower and upper thresholds that define that interval

- The region growing algorithm includes those pixels connected to the seed that satisfy:

\[
T_L \leq I(x,y,z) \leq T_U
\]
Segmentation
Seeded Region Growing

Connected Threshold
Segmentation
Seeded Region Growing

Connected Threshold
Confidence Connected

- Criteria based on statistics of the current region may be used.

- In the simplest case, the algorithm first computes the mean $m$ and standard deviation $\sigma$ of values for all the pixels currently included in the region.

- By providing a factor $f$ to define a range around the mean, neighbor pixels whose values fall inside the range are accepted and included in the region:

\[ I(P) \in [m - f\sigma, m + f\sigma] \]
Confidence Connected
Confidence Connected
Isolated Connected

Two seeds and a lower threshold $T_L$ are provided. The algorithm can be applied to grow a region connected to the first seed and not connected to the second one.
Segmentation
Seeded Region Growing

Isolated Connected
Segementation

Seeded Region Growing

Seeded Region Growing

- Image segmentation using SRG should be performed on smoothed version of the image.

- Like thresholding, region growing is sensitive to noise and is not often used alone but within a set of image processing operations.
Segmentation
Connected Component Analysis

Connected Component

Seeded Region Growing

Methods:
- \textbf{SetSeed}( \text{index0} )
- \textbf{AddSeed}( \text{index1} )
Segmentation

Connected Component Analysis

Region Labeling

**input** image (binary)

**labeled** image (gray)
Region Labeling

1º scan

if $P = 255$:

- if all visited neighbors of $P$ are 0, then $P = \text{new label (1, 2, 3, ...)}$
- if only one visited neighbor of $P$ is not 0, then $P = \text{this neighbor}$
- if exist more than one visited neighbor of $P$ not 0, then:
  - if all neighbors not 0 are equal, then $P = \text{this value}$
  - if neighbors are different, then:
    - $P = \text{one of them}$
    - Construct a table of equivalence

2º scan

- Choose a unique value for each equivalence
- Change all equivalent values to the chosen value
Segmentation
Connected Component Analysis

Region Labeling

Table (equivalence)

2 ≡ 1
4 ≡ 5 ≡ 3
Region Labeling

**input image**
(binary)

**labeled image**
(gray)
Region Labeling
(ITK Implementation)

```cpp
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkConnectedComponentImageFilter.h"
#include "itkRelabelComponentImageFilter.h"

// Define image type and dimension ...
// Instantiate the reader ...
// Create the reader ...
```
Region Labeling

typedef itk::ConnectedComponentImageFilter<
    InputImageType,
    OutputImageType > ConnectedComponentFilterType;

ConnectedComponentFilterType::Pointer
    labeller = ConnectedComponentFilterType::New();

labeller -> SetInput ( reader -> GetOutput () );
labeller -> Update();

Object Relabeling

typedef itk::RelabelComponentImageFilter<
    OutputImageType,
    OutputImageType > RelabelComponentType;

RelabelComponentType ::Pointer
    relabeller = RelabelComponentType ::New();

relabeller -> SetInput( labeller -> GetOutput() );
relabeller -> Update();
Object Statistics
(1 object)

```cpp
// ...

unsigned long nObjects =
    relabeller -> GetNumberOfObjects();

std::cout << "Number of objects: "
    << nObjects
    << std::endl;

unsigned long size0fObject1 =
    relabeller -> GetSizeOfObjectsInPixels()[0];

std::cout << "Size of object 1: "
    << size0fObject1
    << " pixels" << std::endl;

// ...
```
Object Statistics
(All objects)

```cpp
// ...

const std::vector<unsigned long> size_pix =
    relabeller -> GetSizeOfObjectsInPixels();

const std::vector<float> size_phys =
    relabeller -> GetSizeOfObjectsInPhysicalUnits();

std::vector<unsigned long>::const_iterator it1;
std::vector<float>::const_iterator it2;

int i;
for (i = 0, it1 = size_pix.begin(), it2 = size_phys.begin();
    it1 != size_pix.end();
    ++i, ++it1, ++it2 )
    std::cout << "Size of object " << i+1 << ": "
        << (*it1) << " pixels, "
        << (*it2) << " physical units" << std::endl;

// ...
```
Segmentation
Connected Component Analysis

Connected Components

cthead1.png (UC, 2D)

binary threshold ($T_L = 150, T_U = 255$)

connected components (relabeled objects: 23)
Watersheds Concepts

- The strategy of watershed segmentation is to treat an image \( I \) as a **height function**

- The image \( f \) is often not the original input data, but is derived from that data through some filtering, feature extraction, or fusion of feature maps from different sources

- Assume that higher values of \( I \) (or \( -I \)) indicate the presence of boundaries in the original data.
Watersheds Concepts

- **Watershed segmentation** classifies pixels into regions using gradient descent on image features and analysis of weak points along region boundaries.

- Imagine *water raining* onto a landscape topology and flowing with gravity to collect in low basins. The size of those basins will grow with increasing amounts of precipitation until they spill into one another, causing small basins to merge together into larger basins.

- Regions (**catchment basins**) are formed by using local geometric structure to associate points in the image domain with local extrema in some feature measurement such as curvature or gradient magnitude.
Gradient descent associates regions with local minima of $I$ using the watersheds of the graph of $I$. 
Segmentation

Watersheds

Watersheds

- The care with which the data is preprocessed will greatly affect the quality of the result;

- Typically, the best results are obtained by preprocessing the original image with an edge-preserving filter, such as anisotropic diffusion or bilateral image filters;

- The height function used as input should be created such that higher positive values correspond to object boundaries;

- A suitable height function for many applications can be generated as the gradient magnitude of the image to be segmented.
Segmentation
Watersheds

Watershed

cthead1.png (UC, 2D)

watershed 2.0 5 0.01 0.05 1

watershed 2.0 5 0.0 0.05 1
Segmentation
ITK Computer Exercises

C++ Exercises

- OtsuThreshold.zip
- IJ_RobustSelection.zip
- ConnectedThreshold.zip
- ConfidenceConnected.zip
- IsolatedConnected.zip
- ConnectedComponent.zip
- WatershedSegmentation1.zip

Python Exercises

1. OtsuThreshold.py
2. MultipleOtsuThreshold.py
3. ConnectedThreshold.py
4. ConnectedComponent.py
End.