ITK Meets OpenCV
A New Open Source Software Resource for CV

Insight Software Consortium

June 2011

Luis Ibáñez  Patrick Reynolds  Gabe Hart  Matt Leotta  Amitha Perera
1. Virtual Machines Preparation
2. ITK Overview
3. Software Environment
4. ITK Introduction
5. OpenCV Introduction
6. ITK OpenCV Bridge
7. ITK Video Filters
What this Tutorial is about

- Introduce the Insight Toolkit (ITK) to Computer Vision
- Hands-on code examples as a starting point for using ITK
- Preview of the upcoming video extensions to ITK
- Raffle of ASUS Eee Pad Transformer
Who is Kitware

SOFTWARE PROCESS

SUPERCOMPUTING VISUALIZATION

MEDICAL IMAGING

DATA MANAGEMENT

COMPUTER VISION
Program

- Virtual Machines Preparation (10min)
- ITK Overview (30min)
- Short Break (10min)
- Software Environment (10min)
- ITK Introduction (30min)
- Break (30min)
- OpenCV Introduction (30min)
- OpenCV+ITK (30min)
- Short Break (10min)
- ITK Video Filters (40min)
- Wrap-up (10min)
Virtual Machines Preparation
Virtual Machines Preparation

- Get USB Memory Stick
- Install VirtualBox from it
- Import the VirtualMachine file
- Boot the Virtual Machine
- Log in
- Get familiar with directories
USB Memory Stick Content
Directories and Files

- **VirtualBoxInstallers**
  - VirtualBox-4.0.8-71778-OSX.dmg (Mac)
  - VirtualBox-4.0.8-71778-Win.exe (Windows)
  - (Ubuntu Linux)
    - virtualbox-4.0.8-71778 Ubuntu lucid amd64.deb
    - virtualbox-4.0.8-71778 Ubuntu lucid i386.deb
    - virtualbox-4.0.8-71778 Ubuntu maverick amd64.deb
    - virtualbox-4.0.8-71778 Ubuntu maverick i386.deb
    - virtualbox-4.0.8-71778 Ubuntu natty amd64.deb
    - virtualbox-4.0.8-71778 Ubuntu natty i386.deb
    - ...

- **VirtualMachine**
  - ”859d59f9-ed19-4aa2-9cd6-a852ba47cdac.vmdk”
  - ”OpenCV-ITK.ovf”
Install VirtualBox

- Select the installer for your platform
- Run it
You can also install VirtualBox by doing:

- `sudo apt-get install virtualbox-ose-qt`
Importing the Virtual Machine

- Run VirtualBox
- In “File” Menu select “Import Appliance”
- Provide the filename in the USB stick ”VirtualMachine/OpenCV-ITK.ovf”
- A progress bar will appear, and when it finishes you should see:

![Image of VirtualBox interface]
Booting the Virtual Machine

- Click on the “OpenCV-ITK” icon on the left, to select it.
- Click on the Green Arrow at the top “Show”.
- The VM will start to boot and you will see the warning:

![Warning Message]

- Click “OK”
Booting the Virtual Machine

- The boot sequence should continue and you should see:
Your Virtual Machine is Ready!
Additional Copies

- The same source trees are available in the USB key
- Outside of the VirtualBox image
ITK Overview
What is ITK?

- C++ Library
- Open Source (Apache 2.0 License)
- Generic Programming / C++ Templates
- Image Processing
- Image Segmentation
- Image Registration
What ITK is not?

- No Visualization
- No GUI
Funding

- Funded (mostly) by the US National Library of Medicine
- With contributions from
  - National Institute of Dental and Craniofacial Research
  - National Science Foundation
  - National Eye Institute
  - National Institute of Neurological Disorders and Stroke
  - National Institute of Mental Health
  - National Institute on Deafness and Other Communication Disorders
  - National Cancer Institute
History

- Started in 2000
- Developed by Companies and Universities
  - GE Corporate Research
  - Insightful
  - Kitware
  - UNC Chapel Hill
  - University of Pennsylvania
  - University of Utah
Recent contributions by a larger community

- Harvard - Brigham and Women’s Hospital
- University of Iowa
- Georgetown University
- INRA - France
- German Cancer Research Center
- ...and many others...
Project Profile

- Total lines of code: 1,886,674
- Active developers: 56 (past 12 months)
- Total developers: 146 (full history of the project)
- Top 2% largest open source teams in the world
- Estimated cost of development: $34.5 M

\(^1\text{Ohloh: http://www.ohloh.net/p/itk}\)
Funding

- First 10 years of development = $15M
- Refactoring of ITKv4 in 2011 = $5M
- Yearly maintenance supported by NLM
Community Size

Users Mailing List
- 2,071 Subscribers
- About 400 emails/month

Developers Mailing List
- 475 Subscribers
- About 350 emails/month

4,800 Downloads/month
- from 116 countries
What ITK has to offer

- Image Processing
- Segmentation
- Registration
Anisotropic Diffusion
Mathematical Morphology
Anti-Aliasing
Resampling
Distance Maps
Bias Correction
Diffusion Tensor Imaging
Hessians, Laplacian, Gaussians
Hough, Canny
Image Segmentation

- Level Sets
- Region Growing
- Watersheds
- Label Voting
- Connected Components
- Label Image Processing
- Deformable Models
- Cellular Models
- Statistical Classification
  - K-Means
  - Gaussian Mixture Models
  - Markov Random Fields
Image Registration

- **Generic Framework**
  - Image Metrics
  - Interpolators
  - Transforms
  - Optimizers

- **Deformable Registration**
  - BSplines (Free-form)
  - Demons
  - Diffeomorphic Demons
  - PDE Solving

- **Multi-Resolution**
- **Multi-Modality**
The Insight Journal
New Algorithms Every Week...

- 2,129 Registered Users
- 472 Publications
- 834 Open Reviews
- Open Access Journal
- Reproducible Research (RR)
Software Environment
How to take the mouse out of the Virtual Machine

- Hit the **RIGHT CTRL** Key
How to Open a Terminal - Icon in Upper Left Corner
To type your command line instructions
How to Open a Terminal - Icon in Upper Left Corner

To type your command line instructions
How to Navigate Directories
Double Click in Folder Icons in the Desktop
Walk through the directories

- Find source code of exercises
  ```
  cd ~/src/ITK-OpenCV-Bridge-Tutorial/Exercises
  pwd
  ls
  nautilus .
  ```

- Find binary build of exercises
  ```
  cd ~/bin/ITK-OpenCV-Bridge-Tutorial/Exercises
  pwd
  ls
  ```
How to View Images

- Go to the directory
- Invoke "eye of gnome" eog application
  ```
  cd ~/data
  eog mandrill.png
  ```
- Hit ESC key to quit the application
How to View Videos

- Go to the directory
- Invoke "VideoLAN" vlc application
  ```
  cd ~/data
  vlc Walk1.mpg
  
  ```
- Hit CTRL-Q to quit the application
- or use the menus
ITK Introduction
You will typically do:
- Include headers
- Pick pixel type
- Pick image dimension
- Instantiate image type
- Instantiate filter type
- Create filters
- Connect pipeline
- Run pipeline
Basic Filtering - Median Filter

Include headers

```cpp
19 #include <itkImageFileReader.h>
20 #include <itkImageFileWriter.h>
21 #include <itkMedianImageFilter.h>
```

- Read images from files
- Write images from files
- Apply a Median filter in an image
Declare pixel types and image dimension

```cpp
typedef unsigned char InputPixelType;
typedef unsigned char OutputPixelType;
const unsigned int Dimension = 2;
```

Declare input and output image types

```cpp
typedef itk::Image< InputPixelType, Dimension > InputImageType;
typedef itk::Image< OutputPixelType, Dimension > OutputImageType;
```
### Declare the types for reader and writer

```cpp
40 typedef itk::ImageFileReader< InputImageType > ReaderType;
41 typedef itk::ImageFileWriter< OutputImageType > WriterType;
```

### Instantiate the reader and writer objects (source and sink)

```cpp
43 ReaderType::Pointer reader = ReaderType::New();
44 WriterType::Pointer writer = WriterType::New();
```

### Set input and output filenames

```cpp
46 reader->SetFileName( argv[1] );
47 writer->SetFileName( argv[2] );
```
Declare the Median filter type

49  typedef itk::MedianImageFilter<InputImageType, OutputImageType> FilterType;

Create the filter

51  FilterType::Pointer filter = FilterType::New();
Define the Median kernel radius (Manhattan Radius)

```cpp
55  indexRadius[0] = atoi(argv[3]);  // radius along x
56  indexRadius[1] = atoi(argv[4]);  // radius along y
57  filter->SetRadius(indexRadius);
```

Connect the pipeline

```cpp
60  filter->SetInput(reader->GetOutput());
61  writer->SetInput(filter->GetOutput());
```
• Trigger the pipeline execution by calling Update().

```cpp
try {
    writer->Update();
}
catch ( itk::ExceptionObject & excp ) {
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
}
```

• ITK uses C++ exceptions for error management
• Exceptions are typically thrown during Update() calls
• Applications must catch the exceptions and solve them
How to Configure and Build

cmake-gui

- Create a binary directory
- Configure the code with CMake
- Build (compile and link an executable)
- Run it in example image
How to Configure and Build
cmake-gui

- Create a binary directory
  
  ```bash
  cd ~/bin
  mkdir itkexercise1
  cd itkexercise1
  ```
Run “cmake-gui”
How to Configure and Build

cmake-gui

- Set “Source Directory” (where the source code is)
- Set “Binary Directory” (where to build the executable)
- Click on “Configure”
You will get an error message

Because the project needs ITK
and we have not provided ITK_DIR yet
Provide the path to ITK in the ITK_DIR variable

/home/tutorial/bin/ITKVideo/Release
How to Configure and Build cmake-gui

- Click on “Configure”
- Click on “Generate”
In the command line do:

cd /home/tutorial/bin/itkexercise1
make
How to Run

/home/tutorial/bin/itkexercise1

- While in the binary directory:
  /home/tutorial/bin/itkexercise1

- In the command line type:

  ./BasicImageFilteringITK ~/data/mandrillgray.png ./mandrillgrayMedian.png 3 3
How to View the Result

Image viewing application “eye of gnome”: eog

- In the command line type:
  
  ```
  eog ~/data/mandrillgray.jpg &
  eog ./mandrillgrayMedian.png &
  ```
Result of Median Filter
Excercise 1
Replace the filter with another one

- Select a Filter from the Doxygen documentation (e.g. MeanImageFilter)
- Replace the MedianImageFilter with the selected filter
- Recompile
- Rerun
### Collaboration diagram for Module ITK-Smoothing:

#### Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>itk::BinomialBlurImageFilter&lt; TInputImage, TOutputImage &gt;</code></td>
<td>Performs a separable blur on each dimension of an image. More...</td>
</tr>
<tr>
<td><code>itk::DiscreteGaussianImageFilter&lt; TInputImage, TOutputImage &gt;</code></td>
<td>Blurs an image by separable convolution with discrete gaussian kernels. This filter performs Gaussian blurring by separable convolution of an image and a discrete Gaussian operator (kernel). More...</td>
</tr>
<tr>
<td><code>itk::MeanImageFilter&lt; TInputImage, TOutputImage &gt;</code></td>
<td>Applies an averaging filter to an image. More...</td>
</tr>
<tr>
<td><code>itk::MedianImageFilter&lt; TInputImage, TOutputImage &gt;</code></td>
<td>Applies a median filter to an image. More...</td>
</tr>
<tr>
<td><code>itk::RecursiveGaussianImageFilter&lt; TInputImage, TOutputImage &gt;</code></td>
<td>Base class for computing IIR convolution with an approximation of a Gaussian kernel. More...</td>
</tr>
<tr>
<td><code>itk::SmoothingRecursiveGaussianImageFilter&lt; TInputImage, TOutputImage &gt;</code></td>
<td>Computes the smoothing of an image by convolution with the Gaussian kernels implemented as IIR filters. More...</td>
</tr>
</tbody>
</table>

#### Detailed Description

This module includes the most common image smoothing filters. For example, Gaussian and Median filters. You may also find it interesting to look at the AnisotropicSmoothing group of filters.

**Dependencies:**
- Module ITK-ImageFilterBase
- Module ITK-ImageFunction
Excercise 1

First we replace the Header file:

```cpp
21 #include <itkMeanImageFilter.h>
```

Then we replace the Filter instantiation:

```cpp
49 typedef itk::MeanImageFilter<InputImageType, OutputImageType> FilterType;
```
In the command line do:

```
cd /home/tutorial/bin/itkexercise1
make
```
How to Run
/home/tutorial/bin/itkexercise1

- While in the binary directory:
  /home/tutorial/bin/itkexercise1
- In the command line type:
  ./BasicImageFilteringITK
      
    ~/data/mandrillgray.png
    ./mandrillgrayMean.png
    3 3
How to View the Result

Image viewing application “eye of gnome”: `eog`

In the command line type:

```
eog ~/data/mandrillgray.jpg &
eog ./mandrillgrayMean.png &
```
Result of Mean Filter
Find All Other Exercises

- Go to the binary directory
  
  ```
  cd ~/bin/ITK-OpenCV-Bridge-Tutorial/Exercises
  ```
Some filters expect specific pixel types
Canny Edge detection is an example
Here we Cast the image before Canny
Then we Cast/Rescale it after Canny
Let’s start with the relevant headers:

```cpp
#include <itkCastImageFilter.h>
#include <itkCannyEdgeDetectionImageFilter.h>
#include <itkRescaleIntensityImageFilter.h>
```

We then declare the relevant pixel types:

```cpp
typedef unsigned char InputPixelType;
typedef float RealPixelType;
typedef unsigned char OutputPixelType;
```

Then we declare the relevant image types:

```cpp
typedef itk::Image< InputPixelType, 2 > InputImageType;
typedef itk::Image< RealPixelType, 2 > RealImageType;
typedef itk::Image< OutputPixelType, 2 > OutputImageType;
```
We declare the Casting filter and instantiate it:

```cpp
typedef itk::CastImageFilter<
    InputImageType, RealImageType > CastFilterType;

CastFilterType::Pointer caster = CastFilterType::New();
```

We declare and instantiate the Canny filter:

```cpp
typedef itk::CannyEdgeDetectionImageFilter<
    RealImageType, RealImageType > FilterType;

FilterType::Pointer canny = FilterType::New();
```

and do the same for the RescaleIntensity filter:

```cpp
typedef itk::RescaleIntensityImageFilter<
    RealImageType, OutputImageType > RescaleFilterType;

RescaleFilterType::Pointer rescaler = RescaleFilterType::New();
```
We connect the pipeline:

70
    caster->SetInput( reader->GetOutput() );
71
    canny->SetInput( caster->GetOutput() );
72
    rescaler->SetInput( canny->GetOutput() );
73
    writer->SetInput( rescaler->GetOutput() );

Set the parameters of the Canny Edge detection filter:

76
    canny->SetVariance( atof( argv[3] ) );
77
    canny->SetLowerThreshold( atof( argv[4] ) );
78
    canny->SetUpperThreshold( atof( argv[5] ) );
- Trigger the execution of the pipeline:

```cpp
try {
    writer->Update();
} catch ( itk::ExceptionObject & excp ) {
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
}
```

- Note that the pipeline only runs when we call `Update()`
- That’s the point where we should catch exceptions
How to Run

/home/tutorial/bin/itkexercise1

- While in the binary directory:
  /home/tutorial/bin/itkexercise1

- In the command line type:
  
  ./BasicImageFilteringITKAnswer2  
  ~/data/mandrillgray.png  
  ./mandrillgrayCanny.png  
  6 1 8

- 6 = Variance for Gaussian
- 1 = Lower Threshold
- 8 = Upper Threshold
How to View the Result
Image viewing application “eye of gnome”: eog

- In the command line type:
  
  ```bash
  eog ~/data/mandrillgray.jpg &
  eog ./mandrillgrayCanny.png &
  ```
Result of Canny Filter
OpenCV Introduction
What is OpenCV? OpenCV is . . .

- an open source computer vision library.
- written in C, but has C++ and Python APIs.
- released under a BSD license.
- supported and guided by Willow Garage.
- found at http://opencv.willowgarage.com/wiki/.

We will be using OpenCV 2.2 (from subversion).

The new C++ interface will be used when possible.

We assume you have some prior experience with OpenCV

. . . but we will cover the basics in case you don’t.
We need to include the OpenCV headers

19 include <opencv2/imgproc/imgproc.hpp>
20 include <opencv2/highgui/highgui.hpp>

and standard library headers for cout and string

22 include <iostream>
23 include <string>
```cpp
int main ( int argc, char **argv )
{
    if ( argc < 2 )
    {
        std::cout << "Usage: " " argv[0] " input_image output_image" " std::endl;
        return -1;
    }

    cv::Mat inputImage = cv::imread( argv[1] );
    cv::Mat resultImage;
    cv::medianBlur( inputImage, resultImage, 9 );

    if( argc < 3 )
    {
        std::string windowName = "Exercise 1: Basic Filtering in OpenCV";
        cv::namedWindow( windowName, CV_WINDOW_FREERATIO);
        cv::resizeWindow( windowName.c_str(), resultImage.cols, resultImage.rows+50 );
        cv::imshow( windowName, resultImage );
        cv::waitKey();
    }
    else
    {
        cv::imwrite( argv[2], resultImage );
    }

    return 0;
}
```
If no arguments, print usage and exit

Note: argv[0] contains the executable name

```cxx
if ( argc < 2 )
{
    std::cout << "Usage: " << argv[0] << " input_image output_image" << std::endl;
    return -1;
}
```

Load an image from the specified file into a matrix object

```cxx
cv::Mat inputImage = cv::imread( argv[1] );
```

Create a matrix for the output and apply a median filter

```cxx
cv::Mat resultImage;
cv::medianBlur( inputImage, resultImage, 9 );
```

The last argument represents the size of the filter, 9 × 9 in this case.
If no output file is specified, use HighGUI to display the resulting image.

```cpp
if ( argc < 3 )
{
  std::string windowName = "Exercise 1: Basic Filtering in OpenCV";
  cv::namedWindow( windowName, CV_WINDOW_FREERATIO );
  cv::ResizeWindow( windowName.c_str(), resultImage.cols, resultImage.rows+50 );
  cv::imshow( windowName, resultImage );
  cv::waitKey();
}
```
Create the Display Window
OpenCVIntroduction/exercise1/BasicFilteringOpenCV.cxx

- A title string is used to identify the GUI window.
  
  40 std::string windowName = "Exercise 1: Basic Filtering in OpenCV";

- Create a named window.
  - CV_WINDOW_FREERATIO is needed for display using OpenGL.

  41 cv::namedWindow( windowName, CV_WINDOW_FREERATIO );

- Resize the window to match the image size.
  - cvResizeWindow() is a C function with no C++ API equivalent.
  - It is not in the cv namespace.
  - It requires a const char* name, provided by std::string::c_str().
  - The +50 in height to account for the height of tool/status bars.

  42 cvResizeWindow( windowName.c_str(), resultImage.cols, resultImage.rows+50 );
• Show the image in the previously created window.

```cpp
43 cv::imshow( windowName, resultImage );
```

• Wait for the user to press a key, then continue.

```cpp
44 cv::waitKey();
```
If an output file is specified, write the image to the specified file

```cpp
else
{
    cv::imwrite( argv[2], resultImage );
}
```
Exercise 1
OpenCVIntroduction/exercise1/BasicFilteringOpenCV.cxx

- Replace the median filter with a Canny edge detector

```cpp
35 cv::Mat resultImage;
36 cv::medianBlur(inputImage, resultImage, 9);
```

- Hint: The OpenCV Canny function has this signature

```cpp
void Canny(const Mat& image, Mat& edges, double threshold1, double threshold2);
```

- Canny requires grayscale image input, but our images are color

- Hint: Use this function to convert a color space

```cpp
void cvtColor(const Mat& src, Mat& dst, int code);
```

- Use CV_BGR2GRAY for code to convert BGR color to gray.
Exercise 1: Answer
OpenCVIntroduction/exercise1/BasicFilteringOpenCVAnswer.cxx

```
34 cv::Mat inputImage = cv::imread( argv[1] );
35 cv::Mat grayImage;
36 cv::Mat resultImage;
37 cv::cvtColor(inputImage, grayImage, CV_BGR2GRAY);
38 cv::Canny( grayImage, resultImage, 128, 255 );
```

- Run `./BasicFilteringOpenCV ~/data/mandrill.png`

![Median Filter](Image)

![Canny Edges](Image)
Video filtering is similar to image filtering, except
- use a VideoCapture object to read a video file.
- loop over each frame in the video and process each one.
- display or encode each output frame within the loop.
The first step is to open the video file and parse the headers.

```c++
88 cv::VideoCapture vidCap( argv[1] );
89 if ( !vidCap.isOpened() )
90 {
91     std::cerr << "Unable to open video file: " << argv[1] << std::endl;
92     return -1;
93 }
```

If unable to parse the headers then terminate the program.
Same command line usage as before for selecting between displaying or saving results.

The processing code has been encapsulated into functions.

If only input video is specified, then display the resulting video.

```cpp
if (argc < 3)
{
    processAndDisplayVideo( vidCap );
}
```

If a second file is specified, then save the resulting video to the file.

```cpp
else
{
    processAndSaveVideo( vidCap, argv[2] );
}
```
This function processes the video and displays the results.

```cpp
void processAndDisplayVideo(cv::VideoCapture& vidCap)
{
  Use the get() member function to access video properties:
  frame rate, width, and height.

  double frameRate = vidCap.get(CV_CAP_PROP_FPS);
  int width = vidCap.get(CV_CAP_PROP_FRAME_WIDTH);
  int height = vidCap.get(CV_CAP_PROP_FRAME_HEIGHT);

  Setup the HighGUI window as in the previous example.

  std::string windowName = "Exercise 2: Basic Video Filtering in OpenCV";
  cv::namedWindow(windowName, CV_WINDOW_FREERATIO);
  cv::resizeWindow( windowName.c_str(), width, height+50 );
}
Number of milliseconds to wait before drawing the next frame.

```cpp
unsigned delay = 1000 / frameRate;
```

Loop until no more frames can be read from the video capture object.

```cpp
cv::Mat frame;
while ( vidCap.read(frame) )
{
    cv::Mat outputFrame = processFrame(frame);
    cv::imshow( windowName, outputFrame );
    if ( cv::waitKey(delay) >= 0 )
    {
        break;
    }
}
```

Call `processFrame()` on each frame and display the result.

`waitkey(delay)` returns positive value indicating the key pressed or returns -1 after `delay` ms have passed.
This function processes the video and saves the results.

```cpp
void processAndSaveVideo(cv::VideoCapture& vidCap, const std::string& filename) {
    double frameRate = vidCap.get(CV_CAP_PROP_FPS);
    int width = vidCap.get(CV_CAP_PROP_FRAME_WIDTH);
    int height = vidCap.get(CV_CAP_PROP_FRAME_HEIGHT);
    int fourcc = CV_FOURCC('D', 'I', 'V', 'X');
    cv::VideoWriter vidWrite(filename, fourcc, frameRate,
                             cvSize(width, height));
}
```

Get the video properties as before.

Specify that the output video will be encoded in DIVX format.

Create a video writer object with the specified filename.
Loop until no more frames can be read from the video capture object.

```cpp
71 cv::Mat frame;
72 while( vidCap.read(frame) )
73 {
74   cv::Mat outputFrame = processFrame( frame );
75   vidWrite << outputFrame;
76 }
```

Call `processFrame()` on each frame.

Write the resulting image to the output video file with the insertion operator (`<<`).
Exercise 2
OpenCVIntroduction/exercise2/BasicVideoFilteringOpenCV.cxx

Implement `processFrame()` by adding the Canny edge detector.

```cpp
27 cv::Mat processFrame( const cv::Mat& inputImage )
28 {
29     // ADD FRAME PROCESSING CODE HERE
30     return inputImage;
31 }
```

Saving to a video requires color images, but Canny produces grayscale images.

Hint: Use `cvtColor()` again to convert back to color.

Use code `CV_GRAY2BGR` for code to convert gray to BGR.
Exercise 2: Answer

OpenCVIntroduction/exercise2/BasicVideoFilteringOpenCVAnswer.cxx

```cpp
cv::Mat processFrame( const cv::Mat& inputImage )
{
    cv::Mat grayImage, edgeImage, resultImage;
    cv::cvtColor( inputImage, grayImage, CV_BGR2GRAY );
    cv::Canny( grayImage, edgeImage, 128, 255 );
    cv::cvtColor( edgeImage, resultImage, CV_GRAY2BGR);
    return resultImage;
}
```

- Run `./BasicVideoFilteringOpenCV ~/data/Walk1.mpg`

HighGUI Display

Saved Video (Opened in vlc)
ITK OpenCV Bridge
Introduction

- ITK Module for working with other libraries
- Moving frame and/or video data between OpenCV and ITK
- Bring biomedical and computer vision folks together
- https://github.com/itkvideo/ITK
Funding Source

- Contract from the National Library of Medicine (HHSN276201000579P)
OpenCV users and ITK users should both be comfortable
Image to image utility functions
cv::Source to itk::VideoStream
(Later) Focus on performance
Basic Image Filtering (Revisited)
We include ITK and OpenCV headers (like before):

```cpp
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <itkImage.h>
#include <itkMedianImageFilter.h>
```

We also need to include the bridge header:

```cpp
#include <itkOpenCVImageBridge.h>
```
The basic layout of this file is the same as the OpenCV Examples:

```cpp
int main ( int argc , char **argv )
{
    if ( argc < 2 )
    {
        std::cout << "Usage: " << argv[0] << " input_image output_image" << std::endl;
        return -1;
    }
    cv::Mat inputImage = cv::imread( argv[1] );

    if (argc < 3)
    {
        std::string windowName = "Exercise 1: Basic Filtering in OpenCV & ITK";
        cv::namedWindow( windowName, CV_WINDOW_FREERATIO );
        cv::ResizeWindow( windowName.c_str(), resultImage.cols, resultImage.rows+50 );
        cv::Mat scaled;
        resultImage.convertTo( scaled, CV_8UC1 );
        cv::imshow( windowName, scaled );
        cv::waitKey();
    } else
    {
        cv::imwrite( argv[2], resultImage );
    }
    return 0;
}
```
The type definitions should also be familiar from the ITK Material:

```cpp
typedef unsigned char OutputPixelType;
const unsigned int Dimension = 2;
typedef itk::Image<InputPixelType, Dimension> InputImageType;
typedef itk::Image<OutputPixelType, Dimension> OutputImageType;
typedef itk::OpenCVImageBridge BridgeType;
typedef itk::MedianImageFilter<InputImageType, OutputImageType, FilterType> FilterType;
```

However, notice the bridge class. It contains the conversion function between OpenCV and ITK.

```cpp
typedef itk::OpenCVImageBridge BridgeType;
```
We call our conversion function to go from a cv::Mat to an itk::Image

BridgeType::CVMatToITKImage<InputImageType>(inputImage);
The median filtering is normal ITK code, but we do not connect our output to a writer.

```cpp
FilterType::Pointer filter = FilterType::New();
InputImageType::SizeType neighborhoodRadius;
neighborhoodRadius[0] = 9;
neighborhoodRadius[1] = 9;
filter->SetRadius(neighborhoodRadius);
filter->SetInput(itkImage);
try {
    filter->Update();
} catch (itk::ExceptionObject &excp) {
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
}
```

Instead, we set it to our conversion function.

```cpp
cv::Mat resultImage =
BridgeType::ITKImageToCVMat<OutputImageType>(filter->GetOutput());
```
Run the example with the following command

```
./BasicFilteringITKOpenCVBridge
~/data/mandrillgray.png
~/mandrillgrayMedian.png
```
Viewing the Results

ITKOpenCVBridge/exercise1/BasicFilteringITKOpenCVBridge.cxx

- Running the example the same way as before, we see a nicely median-filtered image.

- Now, the fun part. Let’s modify our example to use Curvature Flow, an anisotropic diffusion filter built into ITK.
Exercise 1
ITKOpenCVBridge/exercise1/BasicFilteringITKOpenCVBridge.cxx

- **Hint 1**: Curvature Flow requires `float` as the output pixel type.
- **Hint 2**: Curvature Flow does not take a radius parameter. It's salient functions are:

```cpp
50  filter -> SetTimeStep( 0.5 );
51  filter -> SetNumberOfIterations( 20 );
```
Exercise 1: Answer
ITKOpenCVBridge/exercise1/BasicFilteringITKOpenCVBridgeAnswer.cxx

- First, change the included filter

```cpp
#include <itkCurvatureFlowImageFilter.h>
```

- The type definitions also need to change to reflect the new filter and output image types.

```cpp
typedef unsigned char InputPixelType;
typedef float OutputPixelType;
const unsigned int Dimension = 2;
typedef itk::Image< InputPixelType , Dimension > InputImageType;
typedef itk::Image< OutputPixelType , Dimension > OutputImageType;
typedef itk::OpenCVImageBridge BridgeType;
typedef itk::CurvatureFlowImageFilter< InputImageType , OutputImageType > FilterType;
```
The semantics of calling the filter also has to change:

```
50    filter->SetTimeStep( 0.5 );
51    filter->SetNumberOfIterations( 20 );
```

That's it! Now you're using a “better” blurring scheme.
Exercise 1: Answer

ITKOpenCVBridge/exercise1/BasicFilteringITKOpenCVBridgeAnswer.cxx

The final pipeline should look like this:

```cpp
35 cv::Mat inputImage = cv::imread( argv[1] );

46 InputImageType::Pointer itkImage = 
  BridgeType::CVMatToITKImage< InputImageType >( inputImage );

49 FilterType::Pointer filter = FilterType::New();
filter->SetTimeStep( 0.5 );
filter->SetNumberOfIterations( 20 );
filter->SetInput( itkImage );
try {
  filter->Update();
} catch ( itk::ExceptionObject & excp ) {
  std::cerr << excp << std::endl;
  return EXIT_FAILURE;
}
```
Run the answer with the following command

```bash
./BasicFilteringITKOpenCVBridgeAnswer
~/data/mandrillgray.png
./mandrillgrayCurvatureFlow.png
```
Exercise 1: Answer
ITKOpenCVBridge/exercise1/BasicFilteringITKOpenCVBridgeAnswer.cxx

The results should look like this:

- Original
- Median Filter
- Curvature Flow
Basic Video Filtering (Revisited)
The basic layout of this file is the same as the OpenCV Video Examples:

- There are three functions as before.

```cpp
28 cv::Mat processFrame( const cv::Mat& inputImage )
```

```cpp
63 void processAndDisplayVideo( cv::VideoCapture& vidCap )
```

```cpp
89 void processAndSaveVideo( cv::VideoCapture& vidCap, const std::string& filename )
```

- We only need to modify processFrame to incorporate ITK.
The median filtering is again normal ITK code.

```cpp
void processFrame(const cv::Mat& inputImage)
{
    FilterType::Pointer filter = FilterType::New();
    InputImageType::SizeType neighborhoodRadius;
    neighborhoodRadius[0] = 9;
    neighborhoodRadius[1] = 9;
    filter->SetRadius(neighborhoodRadius);

    InputImageType::Pointer itkFrame =
        BridgeType::CVMatToITKImage<
            InputImageType>
            (inputImage);

    filter->SetInput(itkFrame);
    try
    {
        filter->Update();
    }
    catch(itk::ExceptionObject & excp)
    {
        std::cerr << excp << std::endl;
    }

    return BridgeType::ITKImageToCVMat<
        OutputImageType>(filter->GetOutput(),
        true);
}
```
Run the example with the following command

```bash
./BasicVideoFilteringITKOpenCVBridge
~/data/Walk1.mpg
./Walk1Median.mpg
```
Let's modify this example to use the same canny edge detection from the ITK examples

Hint 1: You should use the following function instead of ITK's cast image filter for producing output

```cpp
frameOut.convertTo( frameOut, CV_8U );
```

Hint 2: You should only have to modify processFrame
You should now have some extra typedefs:

```cpp
// Exercise 2: Answer
ITKOpenCVBridge/exercise2/BasicVideoFilteringITKOpenCVBridgeAnswer.cxx

typedef itk::CastImageFilter< InputImageType, RealImageType > CastFilterType;
typedef itk::CannyEdgeDetectionImageFilter< RealImageType, RealImageType > FilterType;
typedef itk::RescaleIntensityImageFilter< RealImageType, OutputImageType > RescaleFilterType;
```
Your new ITK pipeline will look like this:

```cpp
CastFilterType::Pointer caster = CastFilterType::New();
FilterType::Pointer canny = FilterType::New();
RescaleFilterType::Pointer rescaler = RescaleFilterType::New();

InputImageType::Pointer itkFrame =
  itk::OpenCVImageBridge::CVMatToITKImage<InputImageType>(inputImage);
caster->SetInput(itkFrame);
canny->SetInput(caster->GetOutput());
rescaler->SetInput(canny->GetOutput());

canny->SetVariance(6);
canny->SetLowerThreshold(1);
canny->SetUpperThreshold(8);

try{
  rescaler->Update();
} catch(itk::ExceptionObject & excp){
  std::cerr << excp << std::endl;
}
```
With the following code for output purposes:

```cpp
69 cv::Mat frameOut =
70   BridgeType::ITKImageToCVMat< OutputImageType >(rescaler->GetOutput(), true);
71
72 frameOut.convertTo( frameOut, CV_8U );
73
74 return frameOut;
```
Run the answer with the following command:

```bash
./BasicVideoFilteringITKOpenCVBridgeAnswer
~/data/Walk1.mpg
~/data/Walk1Edges.mpg
```
ITK Video Filters
Support video processing natively in ITK
Standard framework for multi-frame filters
Use ITK’s library of image filters in video context
Video Filtering - Median Filter

ITKVideoPipeline/exercise1/ITKVideoSingleFrameFilters.cxx

- Video data structure
  ```cpp
  #include <itkVideoStream.h>
  ```

- Video file reader and writer
- Use OpenCV for video IO
  ```cpp
  #include <itkVideoFileReader.h>
  #include <itkVideoFileWriter.h>
  #include <itkOpenCVVideoIOFactory.h>
  ```

- Median image filter
- Image filter → video filter wrapper
  ```cpp
  #include <itkMedianImageFilter.h>
  #include <itkImageFilterToVideoFilterWrapper.h>
  ```
Types for data structures

```
const unsigned int Dimension = 2;
typedef unsigned char PixelType;
typedef itk::Image< PixelType, Dimension > FrameType;
typedef itk::VideoStream< FrameType > VideoType;
```

Types reader and writer

```
typedef itk::VideoFileReader< VideoType > ReaderType;
typedef itk::VideoFileWriter< VideoType > WriterType;
```

Types for video median filter

Use image filter type to define video filter type

```
typedef itk::MedianImageFilter< FrameType, FrameType > ImageFilterType;
typedef itk::ImageFilterToVideoFilterWrapper< ImageFilterType > VideoFilterType;
```

(Insight Software Consortium)
Create reader and writer

```
47 ReaderType::Pointer reader = ReaderType::New();
48 WriterType::Pointer writer = WriterType::New();
```

Create image filter and video median filters

```
49 ImageFilterType::Pointer imageFilter = ImageFilterType::New();
50 VideoFilterType::Pointer videoFilter = VideoFilterType::New();
```
Set up reader and writer

Tell ITK that we’re using OpenCV for IO

```cpp
tk::ObjectFactoryBase::RegisterFactory( itk::OpenCVVideoIOFactory::New() );
reader->SetFileName( argv[1] );
writer->SetFileName( argv[2] );
```

Set up radius for (image) median filter

Set video filter wrapper to use image median filter internally

```cpp
FrameType::SizeType neighborhoodRadius;
neighborhoodRadius[0] = 10;
neighborhoodRadius[1] = 10;
imageFilter->SetRadius( neighborhoodRadius );
videoFilter->SetImageFilter( imageFilter );
```
Connect the pipeline

reader → videoFilter → writer

```
62 videoFilter->SetInput( reader->GetOutput() );
63 writer->SetInput( videoFilter->GetOutput() );
```

Try calling Update() to process the entire video

```
try
{
    writer->Update();
}
catch ( itk::ExceptionObject & excp )
{
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
}
```
Exercise 1
Replace median filter with curvature flow filter

- **Hint 1:** Curvature Flow requires `float` as the pixel type for output.
- **Hint 2:** OpenCV uses ffmpeg which requires `unsigned char` as the pixel type for writing.
- **Hint 3:** Curvature Flow does not take a radius parameter. Its salient functions are:

```cpp
70  imageFilter->SetTimeStep( 0.5 );
71  imageFilter->SetNumberOfIterations( 20 );
```

- **Hint 4:** To use an image filter in a video pipeline use `ImageFilterToVideoFilterWrapper`
Excercise 1: Answer
Replace median filter with curvature flow filter

- Include curvature and cast image filters

```cpp
22 #include <itkCurvatureFlowImageFilter.h>
23 #include <itkCastImageFilter.h>
```

- Re-define types using separate IO and Real pixel types

```cpp
37 const unsigned int Dimension = 2;
38 typedef unsigned char IOPixelType;
39 typedef float RealPixelType;
40 typedef itk::Image< IOPixelType, Dimension > IOFrameType;
41 typedef itk::Image< RealPixelType, Dimension > RealFrameType;
42 typedef itk::VideoStream< IOFrameType > IOVideoType;
43 typedef itk::VideoStream< RealFrameType > RealVideoType;
```
Excercise 1: Answer

Replace median filter with curvature flow filter

- Define cast filter to convert from Real to IO pixel type

```cpp
typedef itk::CastImageFilter< RealFrameType, IOFrameType > CastImageFilterType;
typedef itk::ImageFilterToVideoFilterWrapper< CastImageFilterType > CastVideoFilterType;
```

- Replace median filter with curvature flow filter

```cpp
typedef itk::CurvatureFlowImageFilter< IOFrameType, RealFrameType > ImageFilterType;
typedef itk::ImageFilterToVideoFilterWrapper< ImageFilterType > VideoFilterType;
```

- Use Real pixel type for output
Excercise 1: Answer

Replace median filter with curvature flow filter

- Set up video cast filter

```csharp
68 videoCaster->SetImageFilter(imageCaster);
```

- Set up curvature flow filter

```csharp
70 imageFilter->SetTimeStep(0.5);
71 imageFilter->SetNumberOfIterations(20);
72 videoFilter->SetImageFilter(imageFilter);
```

- Connect the pipeline

reader → curvature flow → output caster → writer

```csharp
74 videoFilter->SetInput(reader->GetOutput());
75 videoCaster->SetInput(videoFilter->GetOutput());
76 writer->SetInput(videoCaster->GetOutput());
```
Run the example with the following command

```bash
./exercise1/ITKVideoSingleFrameFilters ~/data/Walk1_short.avi ./Walk1_short_median.avi
```
Excercise 2
Compute frame differences after diffusion

- Start with solution to Excercise 1 (ITKVideoMultiFrameFilters.cxx)
- Hint 1: Use FrameDifferenceVideoFilter
- Hint 2: Difference filter’s important parameter is:

```cpp
73  frameDifferenceFilter->SetFrameOffset(1);
```

- This sets the spacing of the frames to be differenced. Setting it to 1 means adjacent frames will be used.
Excercise 2: Answer
Compute frame differences after diffusion

- Include frame difference filter

```cpp
#include <itkFrameDifferenceVideoFilter.h>
```

- Define type for frame difference filter

- Since this is a native video filter, no wrapper is necessary

```cpp
typedef itk::FrameDifferenceVideoFilter< IOVideoType, IOVideoType > FrameDifferenceFilterType;
```

- Create frame difference filter

```cpp
FrameDifferenceFilterType::Pointer frameDifferenceFilter = FrameDifferenceFilterType::New();
```
Excercise 2: Answer
Compute frame differences after diffusion

- Set the frame offset to 1
  
  ```
  frameDifferenceFilter->SetFrameOffset(1);
  ```

- Connect the pipeline
- reader → curvature flow → output caster → frame differ → writer

  ```
  videoFilter->SetInput(reader->GetOutput());
  videoCaster->SetInput(videoFilter->GetOutput());
  frameDifferenceFilter->SetInput(videoCaster->GetOutput());
  writer->SetInput(frameDifferenceFilter->GetOutput());
  ```
Run the example with the following command

```bash
./exercise2/ITKVideoMultiFrameFiltersAnswer ~/data/Walk1_short.avi ./Walk1_short_diff.avi
```

Extra Credit: Add a threshold to the difference output to suppress background noise

- ITKVideoPipeline/exercise1/ITKVideoMultiFrameFiltersAnswer2.cxx