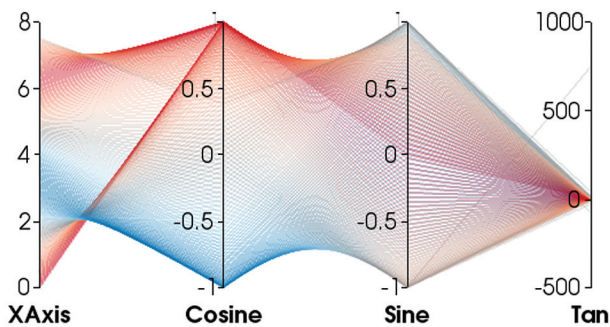
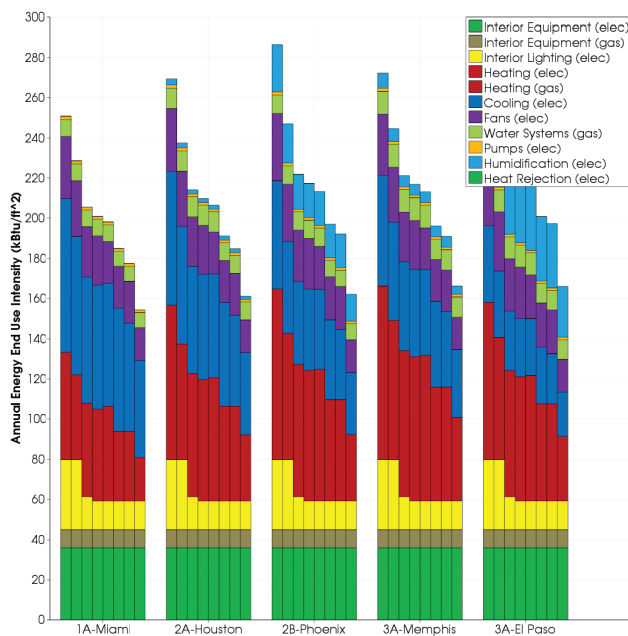


MEDICAL IMAGE ANALYSIS WITH ITK ON APPLE IOS



Parallel coordinates plot

VTK now also provides modern CMake Exports. Inclusion of the `VTKTargets.cmake` file allows easy access to all VTK targets within your own projects that build against VTK. The `UseVTK.cmake` file has also been updated to automatically include `VTKTargets.cmake`.



Stacked bar chart displaying energy consumption

CDASH 1.8

CDash, the open-source, web-based software testing server has had another major release since February 2010. CDash aggregates, analyzes and displays the results of software testing processes submitted from clients around the world, conveying the state of a software system to allow continuous improvements in its quality. This new release fixes more than 20 bugs and adds new features including:

- Support for asynchronous submissions
- Support for CDash@Home: central management of builds
- Automatic removal of builds based on group type
- Better code stability and improved coverage
- Better submission checking with CTest
- Better report of submission errors

Additional information about CDash 1.8 is available at www.cdash.org.

Need a quick way to get started with CDash? We can host your project for free at <http://my.cdash.org>.

Mobile computing devices are becoming increasingly important and prevalent. Apple's mobile devices, the iPod touch, iPhone, and iPad, are notable examples. At the same time, ITK is becoming increasingly popular; implementing a large number of basic and advanced image processing, segmentation, and registration algorithms; and benefiting from a tremendous growth through open-source code contributed by leading research groups and individuals from around the world. The time is right to seriously explore the marriage of these two exciting parallel developments: ITK for medical image analysis and iOS for mobile computing devices.

ITK ON IOS

In the 2010 July-December issue of the Insight Journal [1] we were happy to share with the scientific community the detailed steps needed for building ITK on iOS and creating a basic ITK app on the iOS, which demonstrated successful basic integration. However, our long-term goal has been to open up mobile computing devices to the whole of ITK. We wanted to make it much simpler for an iOS app developer to take advantage of the full ITK functionality. Since images are at the core of such development, the first obstacle we needed to overcome was the lack of a well-defined interface in ITK for the iOS for handling images. Our next mission was clear: we needed to create an ITK Image IO Interface with Apple iOS. On September 13th, 2010, we were excited to release exactly that to the community [2].

ITK IMAGE IO FOR IOS

While in other operating systems the images are specified by a path to the image file itself (e.g. `@"/lib/itk/sampleimages/mri.jpg"`), the iOS software development kit (SDK) uses the `UIImage` object which contains all the image pixel data and metadata. In iOS, the file paths themselves are abstracted even from the programmer and only the file data can be manipulated.

Both image reading and writing is done through Apple's built-in interface. The interface either loads an image from the photo album library and returns its data as a `UIImage`, or saves the data in a `UIImage` object back in a file in the album library. Hence, any interface that deals with image IO must be able to probe and manipulate `UIImage` instances. Based on the `UIImage` class, we have developed `itkiOSImageIO`, the necessary ITK class that provides the interface with the repository of images stored on iOS devices.

ITK Image IO Methods and Interface

The main behaviors our class had to implement were:

- Reading image metadata (dimension, color scheme, tropicity, etc.)
- Reading image pixel data
- Writing image metadata
- Writing image pixel data

As a result, we implemented several critical methods in `itkiOSImageIO`:

The virtual `void ReadImageInformation()` is a method that reads the properties (dimensions, color scheme, tropicity, etc.) of the current image in question.

The virtual void `Read(void* buffer)` is a method that reads the current image data (the actual value at each pixel).

The virtual void `WriteImageInformation()` is a method that writes the image information to the current image. Note this does not write the pixel data yet.

The virtual void `Write(const void* buffer)` is a method that writes the image data into the image file. This writes the actual pixel data.

Figure 1 demonstrates the data flow offered by the `image IO` class:

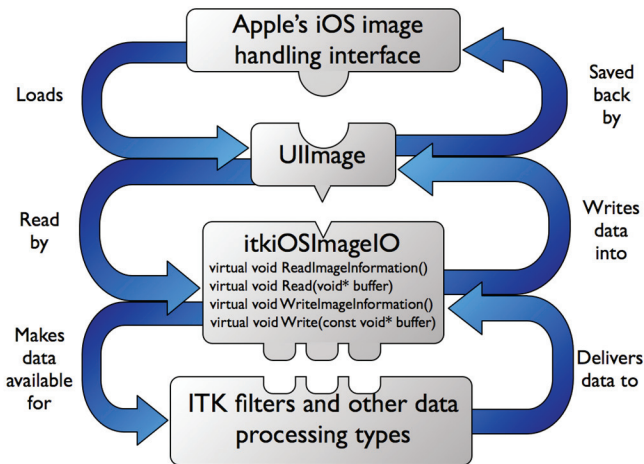


Figure 1: The role of `itkIOSImageIO` as the mediator between Apple's iOS and ITK.

IMPLEMENTING READING METHODS

`ReadImageInformation()` is the first nontrivial method to implement. The necessary image properties that need to be read via this method are the dimensions of the image (width and height), the number of bits used to store each color component value, as well as the interpretation of these bits, e.g. 8 bits per pixel for a 256-level grayscale image or 32 bits per pixel for Red, Green, Blue, and Alpha transparency (RGBA) values of color images.

Fortunately, these necessary components are accessible via Objective-C queries to the `CGImage` object, which can be obtained from the `UIImage` object via the method call `[theUIImagePointer CGImage]`. This call returns a reference to a `CGImage` object which contains all the meta data of the image. This metadata can be queried directly through methods implemented in the `CGImage` class, resulting in most calls such as image dimensions and number of bits per pixel being trivial.

The method virtual void `Read(void* buffer)` needs the data buffer in the argument to be filled with the pixel data. Unfortunately, the image data is not easily accessible to the programmer. There is no known way to directly extract the image data from the `UIImage` object or from the `CGImage` object. Instead, we used a code snippet posted publicly online [3]. In essence, this code creates a new image context in Objective-C, then redraws the image in the new context with the data buffer as a pointer to the context data. The following steps expose the details.

First, obtain the metadata information required to create the context. This includes the number of bits per component, the number of bits per row, the color space information, endianness, the existence of an alpha (transparency) channel or not, and the width and height in pixels of the image.

Second, create the context with the command:

```
CGContextRef theContext = CGContextCreate(
    (buffer, width, height, bitsPerComponent,
     bytesPerRow, colorSpace, theBitmapInfo)
```

Notice that the first argument, `buffer`, is the data buffer that needs to point to the data.

Third, draw the image of interest into the context set up using the command:

```
CGContextDrawImage(theContext,
    CGRectMake(0, 0, width, height),
    theCGImageRef)
```

This draws the image specified by the `CGImageRef` into the `Context`, with the data buffer pointing to the data of the newly drawn image.

Fourth, release the context, since we are only interested in the data pointed to by the variable 'buffer'.

IMPLEMENTING WRITING METHODS

To create an image file in ITK, the metadata is written as the first few blocks of memory using the `WriteImageInformation` method, followed by writing the image pixel data using the `Write` method. However, on the iOS, the `UIImage` class should be used to write the image, which in turn requires access to not only the image pixel data, but also the metadata. Therefore, the `Write(const void* buffer)` method is responsible for organizing all the pertinent image data (pixel data and metadata) and saving it into the appropriate place on the iOS. The following steps highlight the image writing procedure on the iOS.

First, all image metadata is collected. Second, the image is created using the command:

```
CGImageRef theImageRef =
    CGImageCreate(width, height,
                 bitsPerComponent,
                 bitsPerPixel,
                 bytesPerRow,
                 colorSpace,
                 bitmapInfo,
                 theDataProvider, nil,
                 shouldInterpolate,
                 theIntent)
```

This creates an image with all the required properties. The last three arguments in this command are not critical, but further information can be found at Apple's Developer website [4].

Third, a new `UIImage` object is created using the `CGImage` from the previous step using the command:

```
UIImage* outputImage =
    [UIImage imageWithCGImage:(theImageRef)]
```

Fourth, the image is saved into the iOS image library using the command:

```
UIImageWriteToSavedPhotosAlbum(outputImage,
    NULL, NULL, NULL)
```

Again, the last three arguments are not critical, but more information about this method can be found on Apple's Developer website.

EXAMPLE "HELLO IOS IMAGES"

Here, we show how we used the new `itkIOSImageIO` class to read an image and make use of existing ITK classes to filter the image, followed by using the `itkIOSImageIO` class

to write the resulting image to the iOS image library. For brevity purposes, we will only show one set of experiments which uses an RGB image, but further results can be found in the online publication [2].

In order to demonstrate the effectiveness of the `itkiOSImageIO` class when dealing with color images, we have chosen the color image in Figure 2(a) from the public domain Visible Human Project provided by the US National Library of Medicine. We converted it into a grayscale image using ITK's `itk::RGBToLuminanceImageFilter`, and performed binary filtering on it using `itk::BinaryThresholdImageFilter`. When reading and writing the image, the code ran error-free and the results can be seen in Figure 2(b) and 2(c).

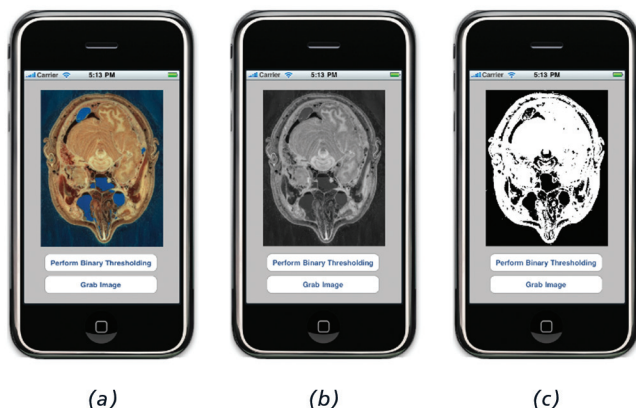


Figure 2: Reading and writing a color image using `itkiOSImageIO`.
(a) the original image, (b) its grayscale converted copy and
(c) its filtered copy.

FUTURE FOR MOBILE MEDICAL IMAGE ANALYSIS

This work has been motivated by the importance of medical image analysis in health applications, which is increasingly being performed using the ITK library, and by the ubiquitous mobile devices, in particular Apple's iOS devices, with their increasingly faster processing, larger storage, and exquisite multi-touch displays. We developed `itkiOSImageIO`, the necessary ITK class to interface with images on iOS devices. However, many important steps still need to be taken to make mobile medical image analysis a popular reality.

One of the important extensions to our work would be to facilitate reading, writing, and processing of 3D or higher dimensional medical images using ITK on iOS devices. There exist several iOS apps that work with and visualize 3D medical images. These 3D images are handled by the app itself and not via the iOS photo library. The most suitable approaches for working with high dimensional images using ITK on the iOS remain to be seen.

The second foreseeable development is supporting C++ libraries other than ITK, mainly the Visualization ToolKit (VTK), ITK's visualization homologue. In the application presented here, we used the `UIImageView` object to display the images before and after processing. VTK, on the other hand, offers much greater image visualization capabilities and works especially well with 3D images and spatial objects produced by ITK.

Another direction for further development is related to ITK's pluggable object factories. This factory mechanism allows the ITK `ImageFileReader` and `ImageFileWriter` functions to

determine at run-time the file format (typically based on the file extensions) and invoke the proper image IO code accordingly. Implementing this mechanism for the iOS, given the iOS constraints on obtaining image information, is left for future work.

For medical images in particular, the physical dimensions of each pixel are quite important. Unfortunately, when reading an image, iOS assumes only isotropic pixels without any physical dimensions assigned to it. Nevertheless, the programmer may set the pixel size of the image using `itkiOSImageIOInstance->SetSpacing(0,xSpacing)` for the horizontal pixel spacing and `itkiOSImageIOInstance->SetSpacing(1,ySpacing)` for the vertical spacing. Further exploration of working with physical units of pixel resolutions, image offsets, direction cosines, etc., on the iOS remains an important future goal.

Although very popular, the iOS is not the only mobile OS. We hope our work will spark parallel efforts that will bring ITK to other mobile platforms such as Google's Android. With mobile operating systems running ITK, it would be interesting to profile the processing speed and memory usage of ITK on these devices and explore the possibility of using several of those as thin clients for server based image analysis and processing.

Finally, it is important to have serious discussions with doctors to better appreciate the clinical applications that would benefit most from medical image analysis on mobile devices and work together on developing apps targeting specific tasks.

REFERENCES

- [1] Boris Shabash, Ghassan Hamarneh, Zhi Feng Huang, and Luis Ibanez. ITK on the iOS. *Insight Journal*, July-December:1-9, 2010
- [2] Boris Shabash, Ghassan Hamarneh, Zhi Feng Huang, and Luis Ibanez. ITK Image IO Interface with Apple iOS. *Insight Journal*, July-December:1-16, 2010
- [3] <http://stackoverflow.com/questions/448125/>
- [4] <http://developer.apple.com>



Zhi Feng Huang received his B.S in Computing Science from Simon Fraser University in 2009. He is a Master's student in the SFU Vision and Media Laboratory performing research on computer vision and machine learning and is funded by the NSERC of Canada through the CGS Master scholarship.



Boris Shabash is a Master's student at the School of Computing Science at Simon Fraser University and a Bachelor of Health Sciences graduate from the University of Calgary. His current research focus is RNA structure visualization and human computer interaction in the field of visual bio-informatics.



Ghassan Hamarneh is an Associate Professor at the School of Computing Science, Simon Fraser University, Canada and is the co-director and founder of their Medical Image Analysis Lab. He obtained his doctoral and master's degrees from Chalmers University of Technology in Sweden, and his bachelor's degree from the University of Jordan. His primary research interest is in medical image analysis.