Using the Julia Image Processing Toolkit

This toolkit works on Julia version 0.4.5. Included are a few test files to show you how it works. All it requires is the installation of Julia. There are currently no dependencies except for **imageIO.jl** which uses Gadfly to generate histogram plots. (You can always comment this out if you don't want to use it - you can generate images using ImageJ).

Here is the code from `testIO.jl`

```
1   include("imageHIST.jl")
2   using imageHIST
3   include("imageIO.jl")
4   using imageIO

5   println("Image filename to read? (.txt)")
6   fnameI= chomp(readline(STDIN))

7   img = imageIO.imreadGray(fnameI)
8   dx,dy = size(img)
9   imE = imageHIST.bihistEQ(img)

10  println("Image filename to write? (.txt)")
11  fnameO = chomp(readline(STDIN))

12  imageIO.imwriteGray(fnameO, imE)
```

The code in lines 1-4 incorporates the two modules used in the file. The code on line 5 prompts the user for the file name, which is input on line 6. The code on line 7 uses the function **`imreadGray()`** from the module **`imageIO`** to read the text file (e.g. mandalay.txt). The image is stored in the array **`img`**. The image is then passed to the function **`bihistEQ()`** from the module **`imageHIST`**. The final three lines 10-12 write the specified image to file (as a text image).

Image I/O

Image input is provided for the toolkit in two modules: **`imageIO`**, and **`PGMimages`**.

Module: **`imageIO`**

The module **`imageIO`** deals with ASCII images, whcih are composed of the 8-bit values stored as rows and columns. There are four functions in this module.

`imreadGray()` - read a text image file
`imwriteGray()` - write a text image file
`getIMGhist()` - generate a histogram with a default bin size of 256
`plotIMGhist()` - plot the histogram using Gadfly

The function `imreadGray()` uses the function `readdlm()` to read in the 2D array of values. Similarly, the function `imwriteGray()` uses the function `writelm()` to write the 2D array of values into an ASCII file.

Module: **`PGMimages`**

The module **`PGMimages`** reads and writes both grayscale and colour images in the PGM format. There are five functions in this module.

`readPGMHeader()` - Read the PGM header
`readPGM()` - Read a PGM image (grayscale)
`writePGM()` - Write a PGM image (grayscale)
`readPGMc()` - Read a PGM image (colour)
`writePGMc()` - Write a PGM image (colour)

Both the grayscale (P5) and the colour (P6) images are input/output as binary files.

Here is some code to read in a colour PGM:

```
print("image filename? ")
fname = chomp(readline(STDIN))

img, nr, nc = PGMimages.readPGMc(fname)
```

Here the function `readPGMc()` returns the colour image as well as the dimensions of the image.

Here is the code from **`testPGM.jl`**:

```
include("PGMimages.jl")
using PGMimages

print("image filename? ")
fnameIN = chomp(readline(STDIN))

#img, nr, nc = PGMimages.readPGM(fnameIN)
img, nr, nc = PGMimages.readPGMc(fnameIN)

print("image filename? ")
fnameOUT = chomp(readline(STDIN))

#PGMimages.writePGM(fnameOUT,img,nr,nc,255,"P5")
PGMimages.writePGMc(fnameOUT,img,nr,nc,255,"P6")
```

This file is used to test the I/O of both grayscale and colour PGM images (relevant sections are commented out using the # delimiter).

NOTE

You can create ASCII image files using ImageJ:

```
File -> Save As -> Text Image...
```

Similarly, colour PGM's can be created in ImageJ:

```
File -> Save As -> PGM...
```