CMSC 165

Digital Image Processing

Image processing is the analysis, interpretation and manipulation of signals in the form of *images* or with *images* as input. An example of which is a person looking at the road to check if it is safe to cross. The person is processing the *image* of the road in his mind.

An **image** is something, an object or a thought, that bears a similar appearance with another object.

Digital image processing is using computer algorithms to do *image processing* on *digital images*.

A **digital image** is a representation of a two-dimensional *image* stored in a computer. Digital images are usually stored as two dimensional arrays of *pixels*.

Pixel or picture element is the smallest piece of information in a digital image and is usually distinguishable by its *color*. The pixel's color usually has three (*red, green, and blue*) or four (*cyan, magenta, yellow, and black*) components. The number of colors a pixel could represent depends on the number of bits it has or its **bbp**. (bits per pixel) The number of colors is equal to 2 raised to the number of bits a pixel has. ($\mathbf{2^{bpp}}$)

ImageLab Syntax

//refer to d_gray.cpp

Images

Imagelab uses two classes in storing and handling digital images: the `Image<T>` class and the `RGBImage` class. The **Image<*T*>** would create a two-dimensional array of the type specified in ***T***.

```
Image<int> IntImg(4,5);
```

In the statement given above, you declare a 4 x 5 image which uses integers to represent pixels. It is the same as declaring a two-dimensional array of the type *int* like in this statement.

```
int IntImg[4][5];
```

The **RGBImage** class, on the other hand, is just like the `Image<T>` class with *int* as its type but with methods made to access the pixels using the RGB color format.

```
RGBImage img;
img.resize(4,5);
```

An RGBImage's dimensions are specified using the resize() method (shown above).

Reading an Image

To be able to analyze, interpret, and manipulate digital images stored in the computer, you must load it in an *RGBImage* variable using the `readJpeg` method. The **`readJpeg`** method has 2 parameters: first is the variable name of the image variable you want the image to be stored to, and the second is the address of the image. The address of the image is inputted as a string (or as a char[]) and could be relative or absolute.

```
RGBImage inputColorImage;

readJpeg(inputColorImage , "images/kristen.jpg" );
```

In the code above, kristen.jpg is read and stored as an RGBImage (*inputColorImage*).

Writing an Image

After processing the image, you could view the contents of the image variable by saving it into the computer by using the `writeJpeg` method. The **`writeJpeg`** method has 3 parameters: first is the variable name of the image variable (must be an *RGBImage*) that contains the image to be saved in the computer, the second is the address the image you will be saving (string or char[]), and the third is the quality of the image (int). The quality of the image would range from 1 (worst) to 100 (best) and would determine the quality and the size of the image saved to the computer.

```
writeJpeg( outputColorImage, "images/output/gray.jpg", 90 );
```

In the code above, `outputColorImage` is saved to the computer as gray.jpg with 90% quality.

Pixel Operations

Accessing the pixels of an image in an image variable is just like accessing an element of a 2D array. A pixel's value can also be changed or acquired.

```
Image<unsigned char> grayImage;

…

// change a pixel value
grayImage(50,75)  = 210;    // pixel location (50,75) now has a value of
210

// get the value of a pixel
unsigned char value2;
value2 = grayImage(34,150);
```

To set the color of a pixel for RGB Images, the COLOR_RGB function is used. This function “mixes” red, green, and blue values to give a unique color value. While the RED(), GREEN(), and BLUE() functions are used to get the pixel’s component values.

```
//outputColorImage is an RGB Image
//COLOR_RGB([0-255], [0-255], [0-255])

outputColorImage(45,89) = COLOR_RGB(255, 255, 255);


//RED(img(x,y)), GREEN(img(x,y)), or BLUE(img(x,y))

int RedValue = RED(outputColorImage(45,89));
```

Other methods

Getting an image’s height and width (in pixels).

```
int ImgHeight = img.height();
int ImgWidth = img.width();
```

How to set the dimensions of an initialized image.

```
//img.resize(width, height);

Image<unsigned char> grayImage;
// set the image's dimensions to  320 pixels wide  and  240 pixels high
grayImage.resize( 320,240 );  // allocate memory for the image
```

How to set all pixels to a value.

```
img.setAll(COLOR_RGB(0,0,0));
// ^ this is equivalent to img.setAll(0);
```

//end: prepared by Jeremiah Pascual, Juan Miguel Bawagan, Maverick Crisostomo, and Dr. Vladimir Y. Mariano