COMS W4115 PLT

Spring 2014

Project Report

ImageVis (IV)

Image Visualization Language

Dainis Kiusals (dvk2102)

Introduction / Motivation

The growth of visual content on the web has created many repositories of visual images available to the user. Accounts with services such as Flickr and Pinterest allow users to create repositories of visual content to share with others on the web, however presentation of the sets of images is often limited in format. The motivation behind ImageVis (IV) is to enable the user to create 2-dimensional visualizations of the image sets which provide more meaningful insight into the data and metadata associated with the pictures. With IV, the user will be able to create visualizations of the image sets in thumbnail format based on relationships between the images. The parameters to use for any of these types of plots could include the image name or size (metadata), as well as visual properties such as color or texture similarity. This may allow patterns in the metadata or visual content to be discovered, as well as the identification of clusters and outliers. The many possible combinations of parameters for the supported plot types will allow the user to create several visualizations leading to the discovery of relationships among the images.

Key Features

- Import of image files.
- Extraction of image sizing information.
- Extraction of image RGB color components for computation of color similarity between pairs of images.
- Extraction of image black & white and texture components for computation of texture similarity between pairs of images.
- Creation of distance matrices identifying color or texture distance between all pairs of images
- Create visualizations in Cartesian format.
- Selection of parameters for x- and y-axis of Cartesian plots.
- Color and texture similarity image comparison based on RGB component and texture histograms for individual images

Language Description

The ImageVis language will mostly contain Java-style statements with additional commands specific to the image feature computation, plot parameter selection and plot display. Images to be provided to the program will be in the .jpg format and the same (x, y) size, with names i01.jpg, i02.jpg,… The user will be able to define the creation of the color and texture histogram functions through intermediate BWValues and TextureValues composite data types. By default the axes for the plot will have no value (list images in order provided, but after distance matrix computation (either for color and/or texture) the axes can be selected to sort based on color or texture before plotting.

Data Types
Integer - numeric integer
Float - numeric floating point
String – for identifiers and filenames

Composite Data Types
Array - for lists of data types
Image - to hold 2D image data
BWValues - 2D array (x, y) for black & white image pixel values for image texture computations
TextureValues - 3D array (image, x, y) for texture image pixel values for image texture comparison
DistanceMatrix - 2D array of float values between all pairs of images (ex: color distances, texture distances)
RGBHistogram - histogram used for image color similarity
TextureHistogram - histogram used for image texture

Iteration
for (loop), while (loop)

Conditionals
if, else

Operators
+ - / * - for arithmetic operation on numeric data types
= - for assignment
==, != - for equality or inequality for numeric data types
>=, >, <=, < - for greater than or equal, greater than, less than or equal and less than operations
+=. -= - for addition to or subtraction from current value of numeric variable

Functions

load() – load images
plot() – plot images
setx() – set ploy x-axis parameter

sety() – set ploy y-axis parameter
rgbhistadd() – add image to RGB histogram
texhistadd() – add image to Texture histogram
computergbdist() – compute RGB distances
computetexdist() – compute texture distances

<u>Sample Program</u>
In this sample the images are the same size.
Code shows import of 20 images and creation of histograms and plot.

```
load 20;  /* load 20 images, named i01.jpg, i02.jpg,…)

for(x; 1; 20)
{
    rgbhistadd x;  /* add image to colorhistogram */
}

for(x; 1; 20)
{
    texhistadd x;  /* add image to texture histogram */
}

for(x; 1; 20)
{
    computergbdist x;  /* compute color distance for first x images */
}

for(x; 1; 20)
{
    computetexdist x;  /* compute texture distance for first x images */
}

setx 1;  /* set x-axis to color */
sety 2; /* set y-axis to texture */

plot;
```

ImageVis (IV) Language Reference Manual

Lexical Conventions

Comments

Comments in IV will be either contained between the /* and */ character sequences.

Whitespace

Whitespace comprises of one or more space, tab or newline characters.

Keywords

The following keywords are reserved for IV:

for	while	if	else
print()	int	float	string
load()	setx()	sety()	rgbhistadd()
texhistadd()	computergbdist()	computetexdist()	plot()

Identifiers

Identifiers in IV are composed of upper and lowercase letters and numbers and the '_' character. IV is a case sensitive language and identifiers need to be case sensitive as well. The first character must be a letter (either case) or '_'.

Constants

Numeric constants are decimal numbers which may be followed by a fractional part (after a decimal). These constants may be of type int or float (if they contain a fractional part). Examples include 3, 1.2, 135.267, 3.37 and 2.0.

String constants are declared with a String type variable or used in an expression with the string constant contained within double quotes (example: "something").

Variables

Local variables are declared within a block, and only have local scope and lifetime. Global variables are declared at the top of the program and have global scope and indefinite lifetime. Declaration is written as the type followed by the variable name or constant with assignment of a value.

Types / Objects

Numeric - numbers may be of type int for integers or float for floating point numbers. They may be defined as constants as described above.

String - string type object represent values such as image names and sizes, and may be displayed on the top of the visualization panel with the print() function.

Array - 1D arrays may be declared for lists of strings, integers or floats with '[]' braces used at declaration with the size (number of elements) contained within the braces. Individual elements are referenced by using an integer to reference the elements through a 0-based index.

RGBHistograms - histograms for each image with red, green and blue histogram components for each Image.

TEXHistograms - histograms for each image with a texture component for each image.

BWValues - 2D array of integer black & white values for each pixel of a corresponding Image, used to compute the texture values for the image.

TextureValues - 3D integer array of texture values for each pixel for multiple images, used for image pair texture comparison.

DistanceMatrix - 2D array of float values representing the color or texture distance between each pair of images in the image set. The distances between an image and itself correspond to the (0,0), (1,1),…etc. entries in the matrix.

Operators

+ addition between two numeric values or concatenation of two String objects

- subtraction between two numeric values, negation of a unary numeric variable

* multiplication of two numeric values

/ division of two numeric values

= assignment operator

+=, -= addition to, subtraction from current value

== equality operator

!= inequality operation

>=, >, <=, < Greater than or equal, greater than, less than or equal and less than operators

Punctuation

; - marks the end of a statement, separator for 'for' loop variable, initiation and final value.

() - prefix and suffix for parameter list

, - separator for parameters in a parameter list

{} – denotes the beginning and end for a block of code, defining the boundaries of local scope

Conditionals

if – the if statement is written with an expression which is either true or false (ex: if (x == 3)). After the expression either a single statement ending with a semicolon or a code block delimited by {} follows and is executed if the expression is true.

else – the else statement must follow the single statement or code block which follows an if statement. It also has a single statement or code block which follows which is executed if the corresponding if expression is false. After the expression either a single statement ending with a semicolon or a code block delimited by {} follows and is executed if the expression is true.

for – the for loop statement is written with a parameter list of 3 elements – a variable, initial value and final value, which define how many times the following statement or code block will be executed (inclusive of endpoints). After the expression either a single statement ending with a semicolon or a code block delimited by {} follows and is executed if the expression is true.

while – the while loop statement is written with a single Boolean expression which is evaluated before each iteration of the loop, followed by a single statement or code block which is executed if the expression is true (ex: while(x < 4)). After the expression either a single statement ending with a semicolon or a code block delimited by {} follows and is executed if the expression is true.

Functions

In IV, function keywords are followed by a space-delimited list of the parameters for the function (not enclosed in braces).

load() – load images, followed by an int representing the number of images to be used.
plot() – plot images
setx() – set ploy x-axis parameter, followed by either 1 or 2, for color or texture
sety() – set ploy y-axis parameter, followed by either 1 or 2, for color or texture
rgbhistadd() – add image to RGB histogram, followed by the int image number
texhistadd() – add image to Texture histogram, followed by the int image number
computergbdist() – compute RGB distances, followed by int for count of images to compare
computetexdist() – compute texture distances, followed by int for count of images to compare
print() – print a string to the title bar of the plot, followed by one string parameter

A program in IV is a sequence of statements which load the images, add the images to the histograms, computes the distances between the images, sets the axes for the plot and displays the plot.

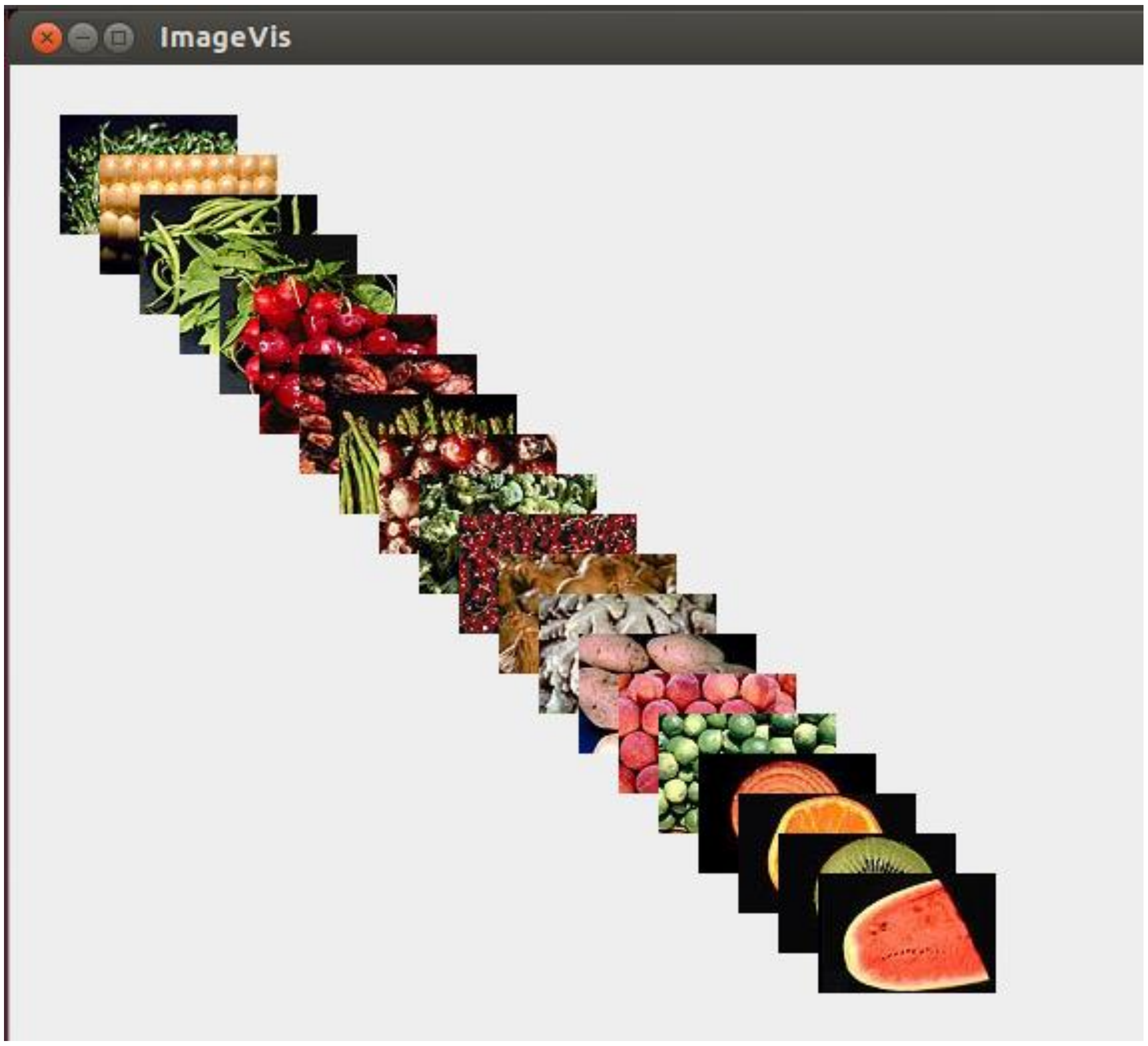
ARCHITECTURE

The main functional blocks of the IV compiler are below: A scanner (IV program file to tokens), parser (tokens to AST), translator (AST to Java code), assembler (combine Java code with the Java loading and plot display utilities to form complete Java class) and Java compiler (to form Java bytecode).

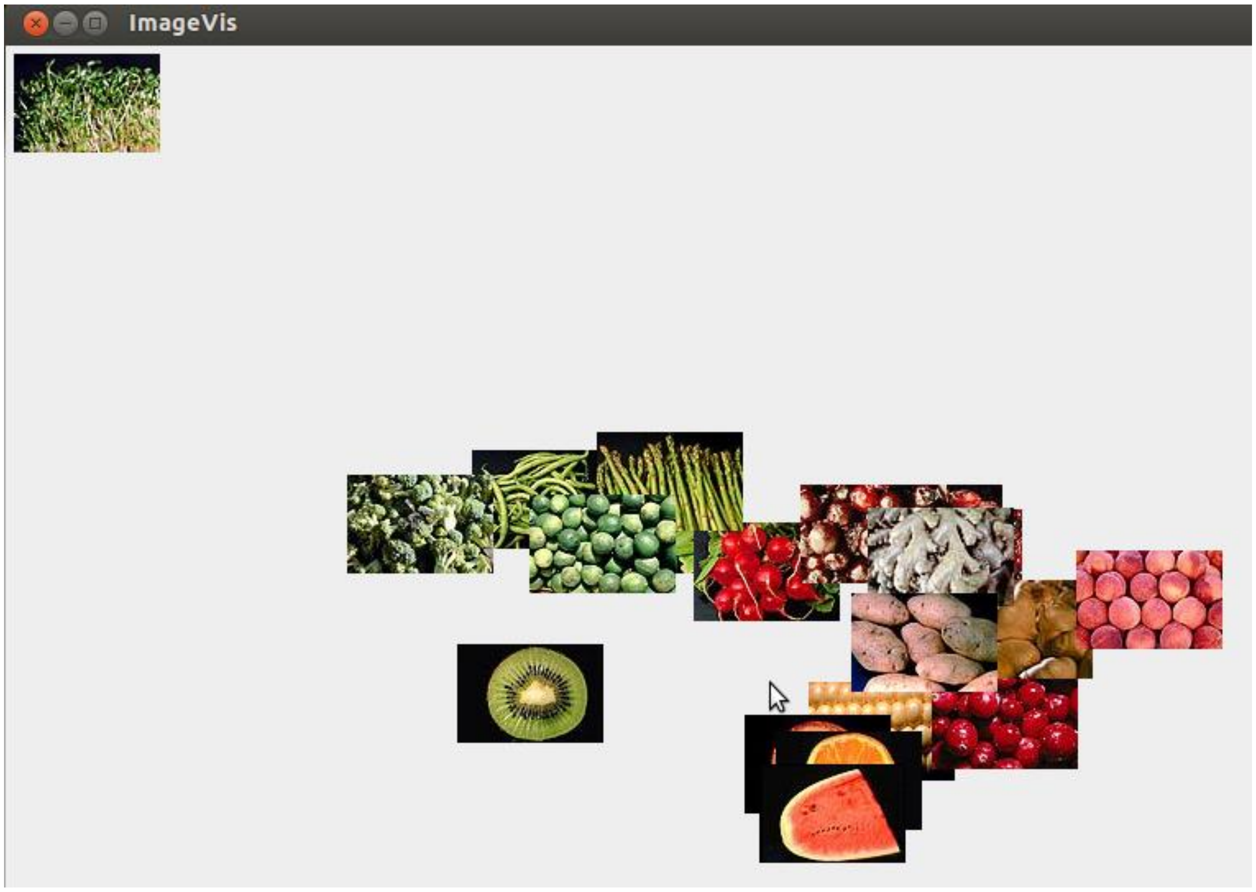
Scanner -> Parser -> Translator -> Assembler -> Java Compiler

RESULTS / TESTING

As only the scanner, parser (compiled with ocamllex and ocamlyacc) and Java utilities for file loading and plot display were implemented due to lack of time to complete the language, the testing only involved manual testing of the mentioned subsystems, which in the case of the file loading and graphics utilities was tested without insertion of the Java code generated by the AST, resulting in a default display of a diagonal line of the image icons displayed in the order in which they were loaded. If the translator were successfully implemented, the image icons would be plotted with the selected base image in the upper left corner, and the other images at a horizontal (x-axis) and vertical (y-axis) distance reflecting the color and/or texture distance between that image and the base image (assigned by setx and sety). The script iv.sh executes IV (with an input program) and then takes the IVutil.java file and combines it with the IV.code (should be generated from the AST) and compiles it with the javac compiler. The default display showing 20 images follows:



The below graphical output shows an output by the graphics utility with the substitution of the following line of code (g.drawImage(di,(int)(IMG_COUNT*40*ColorDist[0][x-1]),(int)(IMG_COUNT*40*TextureDist[0][x-1]), null);) to take into account the color and texture distance from the base image (in the upper left). This is based on inclusion of Java code which is written in the code listing as an example which should be combined with the utility code by the assembler, but which was not translated from the AST (as it should be if the full implementation was completed):



CODE

scanner.mll

```
{ open Parser }

let digit = ['0'-'9']
let digits = (digit)+
let fraction = '.'(digits)
let letter_or_space = ['A'-'Z''a'-'z''_']
let letter_or_space_or_digit = (letter_or_space) | (digit)
```

```
rule token =
 parse [' ' '\t' '\r' '\n'] { token lexbuf }
   | "/*"  { comment lexbuf }
   | '('   { LEFTPAREN }
   | ')'   { RIGHTPAREN }
   | '{'   { LEFTBRACE }
   | '}'   { RIGHTBRACE }
   | '+'   { PLUS }
   | "+="  { PLUSEQUALS }
   | "-="  { MINUSEQUALS }
   | '-'   { MINUS }
   | '*'   { TIMES }
   | '/'   { DIVIDE }
   | '>'   { GREATERTHAN }
   | ">="  { GREATERTHANEQUALS }
   | '<'   { LESSTHAN }
   | "<="  { LESSTHANEQUALS }
   | ';'   { SEMICOLON }
   | ','   { COMMA }
   | '='   { ASSIGNMENT }
   | "=="  { EQUALITY }
   | "=="  { INEQUALITY }
   | "for" { FOR }
   | "while" { WHILE }
   | "if"  { IF }
   | "else" { ELSE }
   | "const" { CONST }
   | "load" { LOAD }
   | "setx" { SETX }
   | "sety" { SETY }
   | "rgbhistadd" { RGBHISTADD }
   | "texhistadd" { TEXHISTADD }
   | "computergbdist" { COMPUTERGBDIST }
   | "computertexist" { COMPUTETEXDIST }
   | "plot" { PLOT }
   | digits as int_lit { INT_LITERAL(int_lit) }
   | (digits)(fraction)? | (digits)?(fraction) as lit { LITERAL(lit) }
   | (letter_or_space)(letter_or_space_or_digit)* as identifier { ID(identifier) }
   | _ as c { print_char c; token lexbuf }
   | eof { EOF }

  and comment = parse
  "*/" { token lexbuf }
   | _ { comment lexbuf }
```

----

parser.mly

```
%{ open Ast %}

%token FOR WHILE CONST LOAD SETX SETY LITERAL
%token GREATERTHAN LESSTHAN GREATERTHANEQUALS LESSTHANEQUALS
%token PLOT PLUSEQUALS MINUSEQUALS IF ELSE INT_LITERAL
%token LEFTBRACE RIGHTBRACE LEFTPAREN RIGHTPAREN
%token SEMICOLON COMMA EOF EQUALITY INEQUALITY
%token PLUS MINUS TIMES DIVIDE ASSIGNMENT RGBHISTADD TEXHISTADD
%token COMPUTERGBDIST COMPUTETEXDIST

%token <string> INT_LITERAL
%token <string> LITERAL
%token <string> ID

%right ASSIGNMENT PLUSEQUALS MINUSEQUALS
%left PLUS MINUS TIMES DIVIDE
%nonassoc ID

%start program
%type <Ast.program> program

%%

program:
  stmt_list { $1 }

stmt_list:
    { [] }
  | stmt stmt_list { $1 :: $2 }

stmt:
    def SEMICOLON { Def($1) }
  | LOAD expr SEMICOLON { Load($2) }
  | RGBHISTADD expr SEMICOLON { RGBHistAdd($2) }
  | TEXHISTADD expr SEMICOLON { TEXHistAdd($2) }
  | COMPUTERGBDIST expr SEMICOLON { ComputeRGBDist($2) }
  | COMPUTETEXDIST expr SEMICOLON { ComputeTEXDist($2) }
  | SETX expr SEMICOLON { SetX($2) }
  | SETY expr SEMICOLON { SetY($2) }
  | PLOT SEMICOLON { Plot() }
  | FOR LEFTPAREN ID SEMICOLON INT_LITERAL SEMICOLON INT_LITERAL RIGHTPAREN LEFTBRACE
stmt_list RIGHTBRACE
        { ForLoop($3, $5, $7, $10) }

def:
   CONST ID ASSIGNMENT expr { ConstDef($2, $4) }

expr:
   expr PLUS expr { Binop($1, Add, $3) }
  | expr MINUS expr { Binop($1, Sub, $3) }
  | expr TIMES expr { Binop($1, Mul, $3) }
  | expr DIVIDE expr { Binop($1, Div, $3) }
  | expr EQUALITY expr { Binop($1, Equal, $3) }
  | expr INEQUALITY expr { Binop($1, Neq, $3) }
  | expr LESSTHAN expr { Binop($1, Less, $3) }
  | expr LESSTHANEQUALS expr { Binop($1, Leq, $3) }
  | expr GREATERTHAN expr { Binop($1, Greater, $3) }
  | expr GREATERTHANEQUALS expr { Binop($1, Geq, $3) }
  | expr ASSIGNMENT expr { Assign($1, $3) }
  | ID { Id($1) }
  | LITERAL { Lit($1) }
  | INT_LITERAL { Lit($1) }
  | LEFTPAREN expr RIGHTPAREN { $2 }
```

IVutil.java

```
import java.io.*;

import javax.imageio.*;

import java.awt.*;

import java.awt.image.*;

import javax.swing.*;



public class IV {

public static void main (String[] args) {

String fileName = null;

File input = null;



int IMG_COUNT = 20;  // Needs to be translated from AST
```

```
// Initialize internal data stuctures and load images

int[][][][] RGBhistograms = new int[IMG_COUNT][13][13][13];

int[][] TEXhistograms = new int[IMG_COUNT][4081];
float[][] ColorDist = new float[IMG_COUNT][IMG_COUNT];
float[][] TextureDist = new float[IMG_COUNT][IMG_COUNT];

int imageWidth = 0;

int imageHeight = 0;

for(int w=0;w<IMG_COUNT;w++)

for(int x=0;x<13;x++)

for(int y=0;y<13;y++)

for(int z=0;z<13;z++)

RGBhistograms[w][x][y][z] = 0;

for(int x=0;x<IMG_COUNT;x++)

for(int y=0;y<4081;y++)

TEXhistograms[x][y] = 0;

for(int i=1;i<=IMG_COUNT;i++)

{

if(i<10)

{

fileName = "i0" + i + ".jpg";

}

else

{

fileName = "i" + i + ".jpg";
```

```
}

input = new File(fileName);

try {

BufferedImage image = ImageIO.read(input);

imageWidth = image.getWidth();

imageHeight = image.getHeight();

int[][] BWValues = new int[imageWidth][imageHeight];

int[][][] TextureValues = new int[IMG_COUNT][imageWidth][imageHeight];

System.out.println("Image " + (i) + " width, height are: " + imageWidth + ", " + imageHeight);



// THE BELOW CODE IS AN EXAMPLE OF WHAT SHOULD BE GENERATED BY THE TRANSLATOR
// FOR A USER-DEFINED IMPLEMENTATION OF THE HISTOGRAM AND DISTANCE IMPLEMENTATIONS
// AND COMBINED WITH THE LOADING CODE ABOVE AND GRAPHICS CODE BELOW BY THE ASSEMBLER
// (IN THE CURRENT PROJECT STATE THIS IS ALL ENCOMPASSED BY ADD***HIST AND COMPUTE***DIST)
//
*************************************************************************************
*******


int xPos = 0;

int yPos = 0;

for(xPos=0;xPos < imageWidth; xPos++)

for(yPos=0;yPos < imageHeight; yPos++)

{

int pixelRGB = image.getRGB(xPos, yPos);

Color pixelColor = new Color(pixelRGB);

int pixelRed = pixelColor.getRed();

int pixelGreen = pixelColor.getGreen();
int pixelBlue = pixelColor.getBlue();

RGBhistograms[(i-1)][(pixelRed/20)][(pixelGreen/20)][(pixelBlue/20)] += 1;


for(int x=0;x<3;x++)

for(int y=0;y<3;y++)

for(int z=0;z<3;z++)

{

RGBhistograms[(i-1)][(x)][(y)][(z)] = 0;

}


BWValues[xPos][yPos] = ((pixelColor.getRed() + pixelColor.getGreen() + pixelColor.getBlue()) / 3);

}


int getxPos = 0;

int getyPos = 0;

for(xPos=0;xPos < imageWidth; xPos++)

for(yPos=0;yPos < imageHeight; yPos++)

{

getxPos = xPos;

getyPos = yPos;


TextureValues[(i-1)][xPos][yPos] = (8 * BWValues[xPos][yPos]);


if(xPos < (imageWidth-1))
```

```
getxPos = xPos + 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

getxPos = xPos;

if(xPos < (imageWidth-1))

getxPos = xPos + 1;

if(yPos < (imageHeight-1))

getyPos = yPos + 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

getxPos = xPos;

getyPos = yPos;

if(yPos < (imageHeight-1))

getyPos = yPos + 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

getyPos = yPos;

if(xPos > 0)

getxPos = xPos - 1;

if(yPos < (imageHeight-1))

getyPos = yPos + 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

getxPos = xPos;

getyPos = yPos;

if(xPos > 0)

getxPos = xPos - 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];
```

```
getxPos = xPos;

if(xPos > 0)

getxPos = xPos - 1;

if(yPos > 0)

getyPos = yPos - 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

getxPos = xPos;

getyPos = yPos;

if(yPos > 0)

getyPos = yPos - 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

getyPos = yPos;

if(xPos < (imageWidth-1))

getxPos = xPos + 1;

if(yPos > 0)

getyPos = yPos - 1;

TextureValues[(i-1)][xPos][yPos] -= BWValues[getxPos][getyPos];

}


for(xPos=0;xPos < imageWidth; xPos++)

for(yPos=0;yPos < imageHeight; yPos++)

{

TEXhistograms[(i-1)][(TextureValues[(i-1)][xPos][yPos] + 2040)] += 1;

}
```

```
} catch (IOException ioe) {

System.exit(1);

}

}




for(int i=0;i<IMG_COUNT;i++)

for(int j=0;j<IMG_COUNT;j++)

ColorDist[i][j] = (float)0.0;

for(int i=0;i<IMG_COUNT;i++)

for(int j=(i);j<IMG_COUNT;j++)

{

for(int x=0;x<13;x++)

for(int y=0;y<13;y++)

for(int z=0;z<13;z++)

{

ColorDist[i][j] += Math.abs(RGBhistograms[i][x][y][z] - RGBhistograms[j][x][y][z]);

}

ColorDist[i][j] = ColorDist[i][j] / (float)(2.0 * imageWidth * imageHeight);
// System.out.println(ColorDist[i][j]);

}




for(int i=0;i<IMG_COUNT;i++)
for(int j=0;j<IMG_COUNT;j++)

TextureDist[i][j] = (float)0.0;

for(int i=0;i<IMG_COUNT;i++)

for(int j=(i);j<IMG_COUNT;j++)

{

for(int x=0;x<4081;x++)

{

TextureDist[i][j] += Math.abs(TEXhistograms[i][x] - TEXhistograms[j][x]);

}

TextureDist[i][j] = TextureDist[i][j] / (float)(2.0 * imageWidth * imageHeight);
// System.out.println(TextureDist[i][j]);

}


// ******************************************************************************


// Graphics Utility



try {

JFrame fd = new JFrame("ImageVis");

fd.setLayout(new FlowLayout());

BufferedImage img=new BufferedImage(1000,1000,BufferedImage.TYPE_INT_ARGB);

Graphics g=img.getGraphics();

g.setColor(Color.blue);

for(int x=1;x<=IMG_COUNT;x++)

{
```

```
String Dpic = "i";

if(x<10)

{

Dpic += "0";

}

Dpic = Dpic + x + ".jpg";

System.out.println(Dpic);

File dinput = new File(Dpic);

BufferedImage di = ImageIO.read(dinput);

g.drawImage(di,x*20,x*20, null);
// Change x*20 for default to function of
// the distance matrices to create different plots with color and/or texture
// difference values for each display icon.
// g.drawImage(di,(int)(IMG_COUNT*40*ColorDist[0][x-1]),(int)(IMG_COUNT*40*TextureDist[0][x-1]),
null);

}



g.dispose();

JLabel label=new JLabel(new ImageIcon(img));

fd.add(label);

fd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

fd.setLocation(0,0);

fd.setSize(200,200);

fd.pack();

fd.setVisible(true);

} catch (Exception e) {
```

```
System.out.println("Error trying to display graph.");

System.exit(1);

}

}

}
```

----

iv.sh

```
#!/bin/bash

# ./IV < $1

cat IVutil.java > IV.java
# cat iv.code >> IV.java

javac IV.java
```

The above script in the full implementation would operate on multiple files made up of the broken-up contents of the loading and graphics utility, so that the image count (load statement), histogram and distance computation code would be inserted.