The purpose of this program is to search for similar images in an image database using histograms. We first begin with the image loading phase in which we essentially build the histograms to represent each image in the database, db, before the user interacts with the program. This stage occurs offline so that all the histograms for the database are created only once which allows the program to run smoothly. The algorithm for this stage is as follows: once the program begins execution before the user interface appears, a compressed image (JPEG) is selected from the database and is decompressed to RGB format, then a color code is calculated for each pixel and stored in a histogram. That processed is repeated for each image stored in the database, as shown in the diagram below.


Description generated with very high confidence

The implantation for the histogram is shown in the structure provided below:

All of the database histograms are stored in the following line of code: std::vector<histogram> db\_histograms;.

// starting at bit s, get f bits to the right of s shifted all the way to the //right,

// while discarding bits outside of the range [s, s-f)

// ex: val = 1001101, bits\_range(val,4,2) -> 00000011

#define BIT\_RANGE(val, s, f) (((val) &= (1 << (s)) - 1) >> ((s) - (f)))

struct histogram

{

void set(int bits)

{

//allocates memory for max possible number of bits for RGB hence one //shifted 24

//does this once

if (!values)

values = (int\*)malloc(sizeof(int) \* (1 << 24));

// only set the amount of bits needed to zero, hence 1 << (bits \* 3)

//just clears the necessary histogram space

memset(values, 0, sizeof(int) \* (1 << (bits \* 3)));

uint8\* pixels = image.pixels;

int len = image.width \* image.height \* 4;

for(int i=0; i < len; i += 4) {

int r = BIT\_RANGE(pixels[ i ], 8, bits) << (bits \* 2);

int g = BIT\_RANGE(pixels[i+1], 8, bits) << bits;

int b = BIT\_RANGE(pixels[i+2], 8, bits);

values[r | g | b]++;

}

}

int\* values = NULL;

int difference = 0;

Image image;

};

Once all the histograms are stored offline, the user interface window is displayed and the user selects an image, which we identify as the test image. A histogram is then calculated for the test image which is then compared to the histograms that represent the database images. The comparison is accomplished by taking the difference between the test images’ histogram, , and the histogram of the current image from the database, and then taking the absolute value of the difference. The difference between the two histograms can be expressed by the following expression:

This comparison process is repeated for all the images stored in the database. Once all the images from the database have been compared to the test image, the most similar images are displayed on the user interface. The algorithm for the main body of the program is displayed below.A screenshot of a cell phone

Description generated with very high confidenceThe following function measure is used to calculate the difference between the test histogram and the database histograms.

//sets up the database and opens all the images at once

//returns a sorted vector from most similar (0 or close to 0 differnce)

// to least similar

std::vector<histogram> measure(histogram& mh, int bits)

{

static std::vector<std::string> fnames = dirlist("db", FILES\_ONLY);

static std::vector<histogram> db(fnames.size());

static int init = 0;

// Create texture for all the images in 'db' directory

//does this for all images once

if (!init) {

for(int i=0; i < db.size(); i++) {

db[i].image.pixels = stbi\_load(("db/"+fnames[i]).c\_str(), &db[i].image.width, &db[i].image.height, &db[i].image.n, RGBA);

texture\_image(&db[i].image);

}

init = 1;

}

// Get the histogram difference, and return sorted vector

mh.set(bits);

for(int i=0; i < db.size(); i++) {

db[i].set(bits);

db[i].difference = hist\_dif(db[i].values, mh.values, bits);

}

std::sort(db.begin(), db.end(), [](const histogram& a, const histogram& b) -> bool { return a.difference < b.difference;});

return db;

}

A screenshot of a computer

Description generated with very high confidence

A screenshot of a computer screen

Description generated with very high confidence

A screen shot of a computer

Description generated with high confidence

A screenshot of a computer screen

Description generated with very high confidence