PBImgAnalysis

P. Baillehache

January 5, 2019

Contents

Introduction

PBImgAnalysis is a C library providing structures and functions to perform various data analysis on images.

It implements the following algorithms:

- K-means clustering on the RGBA space

It uses the `PBErr`, `PBDataAnalaysis`, `GenBrush` libraries.

1 Interface

```
// ============ PBIMGANALYSIS.H ================

#ifndef PBIMGANALYSIS_H
```

```
#define PBIMGANALYSIS_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <execinfo.h>
#include <errno.h>
#include <string.h>
#include "pberr.h"
#include "pbdataanalysis.h"
#include "genbrush.h"

// ================= Define ==================

// ================= Data structure ===================

typedef struct ImgKMeansClusters {
  const GenBrush* _img;
  KMeansClusters _kmeansClusters;
} ImgKMeansClusters;

// ================ Functions declaration ====================

// Create a new ImgKMeansClusters for the image 'img' and with seed 'seed'
// and type 'type'
ImgKMeansClusters ImgKMeansClustersCreateStatic(
  const GenBrush* const img, const KMeansClustersSeed seed);

// Free the memory used by a ImgKMeansClusters
void ImgKMeansClustersFreeStatic(ImgKMeansClusters* const that);

// Get the GenBrush of the ImgKMeansClusters 'that'
#if BUILDMODE != 0
inline
#endif
const GenBrush* IKMCImg(const ImgKMeansClusters* const that);

// Set the GenBrush of the ImgKMeansClusters 'that' to 'img'
#if BUILDMODE != 0
inline
#endif
void IKMCSetImg(ImgKMeansClusters* const that, const GenBrush* const img);

// Get the KMeansClusters of the ImgKMeansClusters 'that'
#if BUILDMODE != 0
inline
#endif
const KMeansClusters* IKMCKMeansClusters(
  const ImgKMeansClusters* const that);

// Search for the 'K' clusters in the RGBA space of the image of the
// ImgKMeansClusters 'that'
void IKMCSearch(ImgKMeansClusters* const that, const int K);

// Print the ImgKMeansClusters 'that' on the stream 'stream'
void IKMCPrintln(const ImgKMeansClusters* const that,
  FILE* const stream);

// Get the index of the cluster including the 'input' pixel for the
// ImgKMeansClusters 'that'
```

```
int IKMCGetId(const ImgKMeansClusters* const that,
  const GBPixel* const input);

// Get the GBPixel equivalent to the cluster including the 'input'
// pixel for the ImgKMeansClusters 'that'
GBPixel IKMCGetPixel(const ImgKMeansClusters* const that,
  const GBPixel* const input);

// Convert the image of the ImageKMeansClusters 'that' to its clustered
// version
// IKMCSearch must have been called previously
void IKMCCluster(const ImgKMeansClusters* const that);

// ================ Polymorphism ==================

// ================ Inliner ====================

#if BUILDMODE != 0
#include "pbimganalysis-inline.c"
#endif

#endif
```

2 Code

2.1 pbimganalysis.c

```
// ============ PBIMGANALYSIS.C ================

// ================ Include ================

#include "pbimganalysis.h"
#if BUILDMODE == 0
#include "pbimganalysis-inline.c"
#endif

// ================ Define ==================

// ================ Functions declaration ====================

// ================ Functions implementation ====================

// Create a new ImgKMeansClusters for the image 'img' and with seed 'seed'
// and type 'type'
ImgKMeansClusters ImgKMeansClustersCreateStatic(
  const GenBrush* const img, const KMeansClustersSeed seed) {
#if BUILDMODE == 0
  if (img == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'img' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Declare the new ImgKMeansClusters
  ImgKMeansClusters that;
  // Set properties
  that._img = img;
  that._kmeansClusters = KMeansClustersCreateStatic(seed);
```

```
  // Return the new ImgKMeansClusters
  return that;
}

// Free the memory used by a ImgKMeansClusters
void ImgKMeansClustersFreeStatic(ImgKMeansClusters* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Free the memory used by the KMeansClusters
  KMeansClustersFreeStatic((KMeansClusters*)IKMCKMeansClusters(that));
}

// Search for the 'K' clusters in the RGBA space of the image of the
// ImgKMeansClusters 'that'
void IKMCSearch(ImgKMeansClusters* const that, const int K) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (K < 1) {
    PBImgAnalysisErr->_type = PBErrTypeInvalidArg;
    sprintf(PBImgAnalysisErr->_msg, "'K' is invalid (%d>0)", K);
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Create a set to memorize the rgb values of the image
  GSetVecFloat input = GSetVecFloatCreateStatic();
  // Get the array of pixels
  const GBPixel* pixels = GBSurfaceFinalPixels(GBSurf(IKMCImg(that)));
  // Get the number of pixels
  long nbPix = (long)GBSurfaceArea(GBSurf(IKMCImg(that)));
  // Loop on pixels
  for (long iPix = nbPix; iPix--;) {
    // Convert the pixel values to float and add them to the
    // input set
    VecFloat* pix = VecFloatCreate(4);
    for (int i = 4; i--;)
      VecSet(pix, i, (float)(pixels[iPix]._rgba[i]));
    GSetAppend(&input, pix);
  }
  // Search the clusters
  KMeansClustersSearch((KMeansClusters*)IKMCKMeansClusters(that),
    &input, K);
  // Free the memory used by the input
  while (GSetNbElem(&input) > 0) {
    VecFloat* v = GSetPop(&input);
    VecFree(&v);
  }
}

// Print the ImgKMeansClusters 'that' on the stream 'stream'
void IKMCPrintln(const ImgKMeansClusters* const that,
  FILE* const stream) {
#if BUILDMODE == 0
  if (that == NULL) {
```

```
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (stream == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'stream' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Print the KMeansClusters of 'that'
  KMeansClustersPrintln(IKMCKMeansClusters(that), stream);
}

// Get the index of the cluster including the 'input' pixel for the
// ImgKMeansClusters 'that'
int IKMCGetId(const ImgKMeansClusters* const that,
  const GBPixel* const input) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (input == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'input' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Convert the pixel values to float
  VecFloat* pix = VecFloatCreate(4);
  for (int i = 4; i--;)
    VecSet(pix, i, (float)(input->_rgba[i]));
  // Get the index of the cluster for this pixel
  int id = KMeansClustersGetId(IKMCKMeansClusters(that), pix);
  // Free memory
  VecFree(&pix);
  // Return the id
  return id;
}

// Get the GBPixel equivalent to the cluster including the 'input'
// pixel for the ImgKMeansClusters 'that'
GBPixel IKMCGetPixel(const ImgKMeansClusters* const that,
  const GBPixel* const input) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
  if (input == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'input' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Declare the result pixel
  GBPixel pix;
  // Get the id of the cluster for the input pixel
  int id = IKMCGetId(that, input);
```

```
  // Get the 'id'-th cluster's center
  const VecFloat* center =
    KMeansClustersCenter(IKMCKMeansClusters(that), id);
  // Update the returned pixel values and ensure the converted value
  // from float to char is valid
  for (int i = 4; i--;) {
    float v = VecGet(center, i);
    if (v < 0.0)
      v = 0.0;
    else if (v > 255.0)
      v = 255.0;
    pix._rgba[i] = (unsigned char)v;
  }
  // Return the result pixel
  return pix;
}

// Convert the image of the ImageKMeansClusters 'that' to its clustered
// version
// IKMCSearch must have been called previously
void IKMCCluster(const ImgKMeansClusters* const that) {
#if BUILDMODE == 0
  if (that == NULL) {
    PBImgAnalysisErr->_type = PBErrTypeNullPointer;
    sprintf(PBImgAnalysisErr->_msg, "'that' is null");
    PBErrCatch(PBImgAnalysisErr);
  }
#endif
  // Get the array of pixels
  GBPixel* pixels = GBSurfaceFinalPixels(GBSurf(IKMCImg(that)));
  // Get the number of pixels
  long nbPix = (long)GBSurfaceArea(GBSurf(IKMCImg(that)));
  // Loop on pixels
  for (long iPix = nbPix; iPix--;) {
    // Get the clustered pixel for this pixel
    GBPixel clustered = IKMCGetPixel(that, pixels + iPix);
    // Replace the original pixel
    pixels[iPix] = clustered;
  }
}
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)
```

```
# Rules to make the executable
repo=pbimganalysis
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$
```

4 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>
#include "pbimganalysis.h"

void UnitTestImgKMeansClusters() {
  srandom(1);
  for (int K = 2; K <= 10; ++K) {
    char* fileName = "./imgkmeanscluster01.tga";
    GenBrush* img = GBCreateFromFile(fileName);
    ImgKMeansClusters clusters = ImgKMeansClustersCreateStatic(
      img, KMeansClustersSeed_Forgy);
    IKMCSearch(&clusters, K);
    printf("%s K=%d:\n", fileName, K);
    IKMCPrintln(&clusters, stdout);
    IKMCCluster(&clusters);
    char fileNameOut[50] = {'\0'};
    sprintf(fileNameOut, "./imgkmeanscluster01-%02d.tga", K);
    GBSetFileName(img, fileNameOut);
    GBRender(img);
    GBFree(&img);
    ImgKMeansClustersFreeStatic(&clusters);
  }
  for (int K = 2; K <= 10; ++K) {
    char* fileName = "./imgkmeanscluster02.tga";
    GenBrush* img = GBCreateFromFile(fileName);
    ImgKMeansClusters clusters = ImgKMeansClustersCreateStatic(
      img, KMeansClustersSeed_Forgy);
    IKMCSearch(&clusters, K);
    printf("%s K=%d:\n", fileName, K);
    IKMCPrintln(&clusters, stdout);
    IKMCCluster(&clusters);
    char fileNameOut[50] = {'\0'};
    sprintf(fileNameOut, "./imgkmeanscluster02-%02d.tga", K);
    GBSetFileName(img, fileNameOut);
    GBRender(img);
    GBFree(&img);
    ImgKMeansClustersFreeStatic(&clusters);
  }
  printf("UnitTestImgKMeansClusters OK\n");
```

```
}

void UnitTestAll() {
  UnitTestImgKMeansClusters();
}

int main(void) {
  UnitTestAll();
  return 0;
}
```

5 Unit tests output

```
./imgkmeanscluster01.tga K=2:
<218.785,141.315,71.795,255.859>
<83.260,118.971,136.942,255.724>
./imgkmeanscluster01.tga K=3:
<73.485,111.447,129.915,255.691>
<233.899,180.949,118.323,255.714>
<200.456,110.663,43.155,255.750>
./imgkmeanscluster01.tga K=4:
<197.770,107.498,40.956,255.728>
<55.685,96.031,114.051,255.563>
<135.950,162.180,179.006,255.298>
<242.126,176.390,101.600,255.701>
./imgkmeanscluster01.tga K=5:
<242.154,176.248,101.317,255.701>
<197.719,107.446,40.924,255.727>
<156.517,174.756,188.655,255.000>
<48.431,83.318,98.715,255.328>
<82.360,128.366,150.550,255.247>
./imgkmeanscluster01.tga K=6:
<178.208,193.387,204.038,255.000>
<197.688,107.394,40.877,255.727>
<46.782,77.709,91.393,255.125>
<242.133,176.098,101.091,255.700>
<122.669,150.250,168.122,255.000>
<65.438,115.939,138.623,255.183>
./imgkmeanscluster01.tga K=7:
<213.324,124.205,51.800,255.500>
<247.538,198.273,128.487,255.275>
<238.727,162.259,84.560,255.480>
<79.776,126.748,149.114,255.238>
<180.573,89.307,29.318,255.423>
<152.006,171.491,186.306,255.000>
<48.016,82.606,97.879,255.306>
./imgkmeanscluster01.tga K=8:
<247.737,203.079,135.230,255.038>
<226.602,142.039,65.317,255.355>
<150.912,170.677,185.675,255.000>
<79.102,126.310,148.706,255.236>
<243.062,172.282,95.573,255.366>
<173.659,83.591,25.808,255.138>
<47.846,82.409,97.678,255.299>
<202.843,110.966,42.828,255.418>
./imgkmeanscluster01.tga K=9:
<65.062,115.580,138.255,255.181>
<225.044,139.790,63.420,255.341>
```

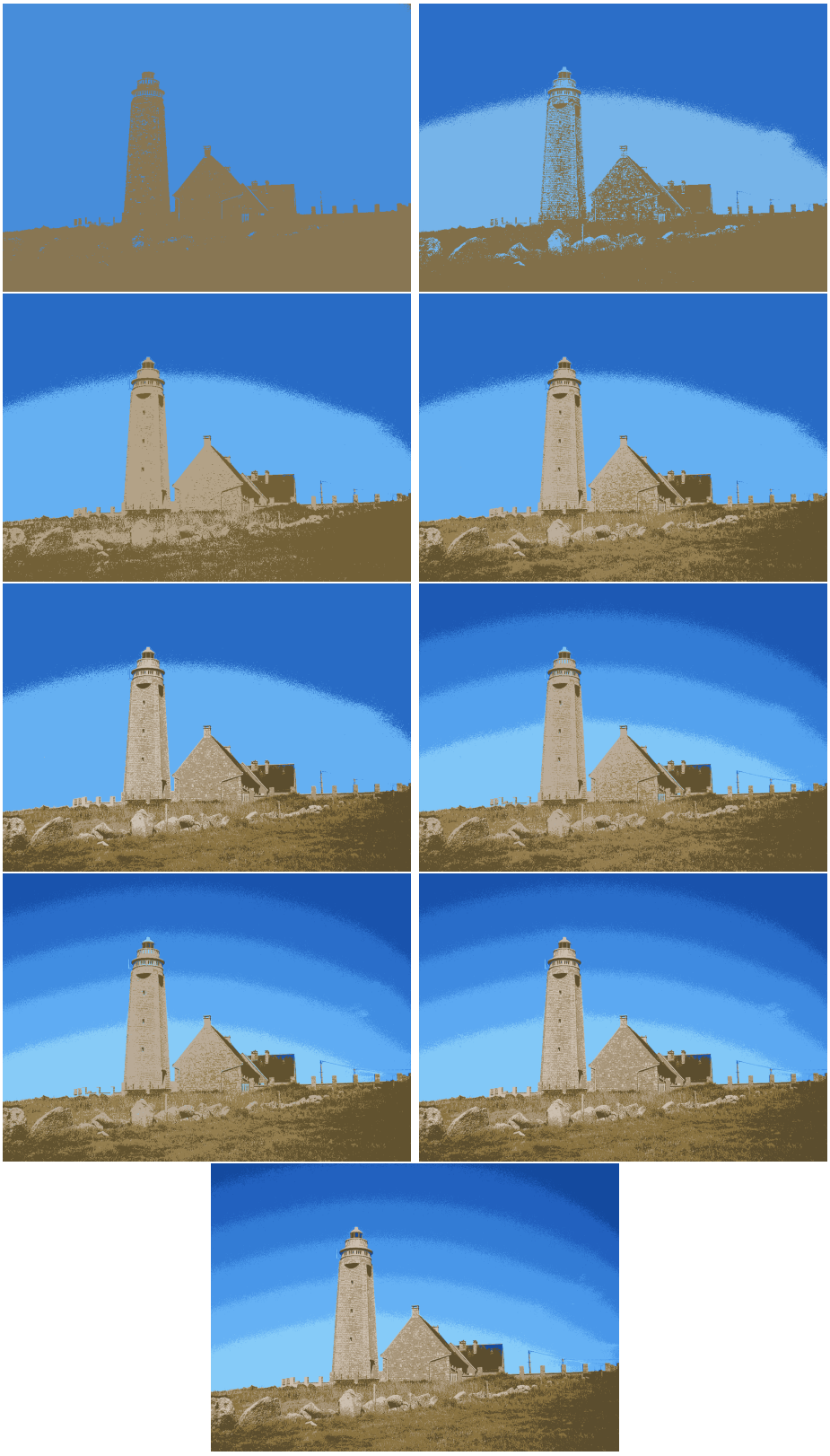
<175.437,191.932,204.291,255.000>
<242.088,169.556,92.422,255.357>
<121.658,149.652,167.620,255.000>
<172.694,82.879,25.351,255.084>
<46.408,77.498,91.273,255.114>
<247.833,200.123,130.008,255.141>
<201.641,109.470,41.866,255.408>
./imgkmeanscluster01.tga K=10:
<175.297,191.864,204.310,255.000>
<233.180,151.540,73.453,255.268>
<213.213,124.019,51.510,255.306>
<248.106,203.788,134.633,255.000>
<46.112,77.616,91.583,255.113>
<159.641,74.786,20.461,255.000>
<121.697,149.667,167.629,255.000>
<65.169,115.672,138.348,255.179>
<244.535,177.064,101.349,255.231>
<191.134,97.676,34.591,255.333>
./imgkmeanscluster02.tga K=2:
<237.517,235.022,233.022,255.698>
<56.371,88.628,75.141,255.852>
./imgkmeanscluster02.tga K=3:
<31.742,76.652,63.372,255.804>
<242.511,240.932,240.034,255.671>
<139.125,131.443,117.259,255.479>
./imgkmeanscluster02.tga K=4:
<18.173,65.878,52.538,255.684>
<61.922,95.780,83.279,255.574>
<150.483,139.868,124.434,255.347>
<242.846,241.342,240.511,255.669>
./imgkmeanscluster02.tga K=5:
<16.885,57.281,45.650,255.548>
<155.742,144.211,128.456,255.253>
<31.841,94.236,78.392,255.555>
<243.009,241.538,240.739,255.667>
<94.153,94.089,83.847,255.205>
./imgkmeanscluster02.tga K=6:
<20.766,76.181,60.754,255.515>
<94.980,93.841,83.367,255.184>
<243.010,241.540,240.741,255.667>
<155.756,144.232,128.484,255.252>
<38.265,101.927,86.619,255.250>
<15.657,44.509,36.153,255.087>
./imgkmeanscluster02.tga K=7:
<95.217,93.678,83.051,255.174>
<155.722,144.207,128.465,255.253>
<43.522,107.580,93.446,255.000>
<27.080,87.516,71.380,255.324>
<243.008,241.538,240.739,255.667>
<17.337,66.813,52.523,255.342>
<15.519,37.754,31.387,255.000>
./imgkmeanscluster02.tga K=8:
<141.704,131.956,116.347,255.115>
<15.810,44.562,36.260,255.092>
<250.559,250.630,250.800,255.510>
<178.996,166.510,152.336,255.000>
<20.063,76.492,60.910,255.508>
<38.037,102.321,86.810,255.236>
<89.550,89.000,79.011,255.078>
<227.283,222.642,219.840,255.000>
./imgkmeanscluster02.tga K=9:
<84.111,82.566,72.399,255.000>

```
<249.838,249.824,249.962,255.537>
<25.888,88.247,71.501,255.305>
<15.695,37.768,31.474,255.000>
<119.227,116.811,105.267,255.000>
<224.224,218.807,215.244,255.000>
<161.105,147.826,131.029,255.000>
<16.572,66.914,52.599,255.330>
<43.725,107.872,93.702,255.000>
./imgkmeanscluster02.tga K=10:
<110.457,110.296,100.194,255.000>
<15.718,37.704,31.438,255.000>
<188.209,175.883,163.171,255.000>
<250.855,250.978,251.154,255.498>
<25.601,88.224,71.362,255.300>
<16.413,66.840,52.499,255.325>
<228.623,224.362,221.878,255.000>
<151.488,139.203,122.271,255.000>
<43.391,107.739,93.451,255.000>
<81.554,80.068,69.901,255.000>
UnitTestImgKMeansClusters OK
```

5.1 K-Means clustering on RGBA space

imgkmeanscluster01.tga:

clustering for K equals 2 to 10:



imgkmeanscluster02.tga:

clustering for K equals 2 to 10: