[read image from .raw file]

#include “PIC.h”

Pic a(fileNameIn, sizey, sizex, sizez);

[save pic as .raw]

#include “PIC.h”

PIC a(fileNameIn, sizey, sizex, sizez) ;

a.output(fileNameOut)

[save image using Shannon-Fano/Huffman/Adaptive Huffman coding]

#include “PIC.h”

Pic a(fileNameIn, sizey, sizex, sizez);

string treeFileName = "tree.csv";

a.encodeFano(fileNameOut, treeFileName); // using Shanno-Fano coding

a.encodeHuff(fileNameOut, treeFileName); // using Huffman coding

a.encodeAdaptHuff(fileNameOut); // using Adaptive Huffman

* Note:

1. Shannon-Fano and Huffman coding will generate the trees file that are required for decoding

[decode file back to image using Shannon-Fano/Huffman/Adaptive Huffman coding]

#include “PIC.h”

Pic a;

a = Pic::decodeFano(fileNameIn, treeFileName, SizeY, SizeX, SizeZ); // using Shanno-Fano coding

a = Pic::decodeHuff(fileNameIn, treeFileName, SizeY, SizeX, SizeZ); // using Huffman coding

a = Pic::decodeAdaptHuff(fileNameIn, SizeY, SizeX, SizeZ); // using Adaptive Huffman

* Note:

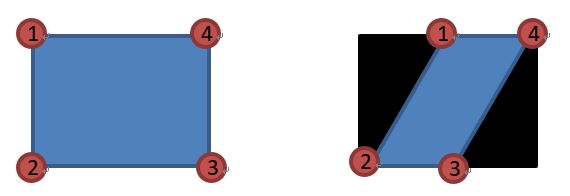
1. Shannon-Fano and Huffman decoding need the tree files that were generated during encoding

[file to file adaptive Huffman encoding and decoding]

#include “AdaptHuffNode.h”

AdaptHuffNode::encodeFile(filenameIn, filenameOut);

AdaptHuffNode::decodeFile(filenameIn, filenameOut);



[homographic transform]

#include “PIC.h”

Pic a(fileNameIn, sizey, sizex, sizez);

Point\_2D corner1(0, 0);

Point\_2D corner2(300, 100);

Point\_2D corner3(400, 500);

Point\_2D corner4(0, 400);

//change the original four corner to new 4 corner

Pic b = a.homoTransform(corner1,corner2,corner3,corner4);

b.output(fileNameOut);

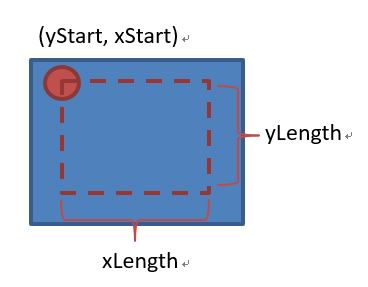
[change colorspace]

Pic a(fileNameIn, sizey, sizex, sizez);

a.RGBToCMY(); // a.RGBToHSL()

[crop image]

#include “PIC.h”

// way#1: return cropped copy

Pic a(fileNameIn, sizey, sizex, sizez);

int yStart=0, xStart=0, yLength=200, xLength=200;

Pic b = a.crop(yStart, xStart, yLength, xLength);

b.output(fileNameOut);

// way#2 take dst as input argument, return nothing

Pic c;

a.crop(c, yStart, xStart, yLength, xLength);

c.output(fileNameOut);

[constrast\_equalize]

Pic a(fileNameIn, sizey, sizex, sizez);

a.histEqualize();

a.output(fileNameOut);

[NLM denoise]

Pic a(fileNameIn, sizey, sizex, sizez);

Filter kernel = Filter::makeFilter\_gauss(3, 3, 3, 0.8);

kernel.print();

int searchSize = 10;

float h = 200;

Pic b = a.NLM(kernel, searchSize, h);

b.output(fileNameOut);

[resize]

Pic a(fileNameIn, sizey, sizex, sizez);

int newY = 200, newX = 200;

a.resize(newY, newX);

a.output(fileNameOut);