TD5: DGtal, Tracing and Border Extraction

In this TP, we introduce `DGtal` and do some elementary image processing like reading, writting and format conversion.

Introduction. "Portable BitMap" formats

The image file format `PBM` (portable bitmap), `PGM` (portable grayscalemap) and `PPM` (portable pixmap) propose a simple solution to manipulate image files. In these three formats, an image is considered as a matrix where the values represent the light intensity in each pixel of the image: black or white (`PBM`), level of gray (`PGM`) or three level of colors RGB (Red, Green, Blue) (`PPM`).

Definition: The corresponding files are composed of the following elements:

1. A "magical number" to identify the type of file: P1 or P4 for `PBM`, P2/P5 for `PGM` and P3/P6 for `PPM`.
2. A whitespace character (white, TABs, CRs, LFs).
3. The image **lenght** (decimal value, encoded in ASCII) followed by a whitespace character. Then the image **height** (decimal value, ASCII) followed by a whitespace character.
4. Only for `PGM` and `PPM` : the maximum intensity (decimal value between 0 and 255, encoded in ASCII) followed by a whitespace character.
5. A matrix of number of size $length \times height$. These number are either:
   - (For P1/P2/P3) Decimal values (encoded in ASCII) separated by whitespace
   - (For P4/P5/P6) Binary values directly encoded on 1 or 2 octets. In this case, there is no whitespace between values.

Remarks:

- The lines starting by the character "#" are ignored.
- The lines have to contain less than 70 characters.

Examples

```
P1
# feep.pbm
24 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

File PBM of an image 24×7 whose values are coded in ASCII

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

File PGM of an image 24×7. The intensity values coded in ASCII are at most 15

```
P3
# pasbeau.ppm
4 4
15
0  0  0    0  0  0    0  0  0   15  0 15
0  0  0    0 15  7    0  0  0    0  0  0
0  0  0    0  0  0    0 15  7    0  0  0
15  0 15   0  0  0    0  0  0    0  0  0
```

File PPM of an image 4×4. The intensity values coded in ASCII are at most 15

The DGtal Project

DGtal is an open-source library focusing on digital geometry tools (http://dgtal.org). You must first clone DGtal from https://github.com/DGtal-team/DGtal

```
git clone https://github.com/DGtal-team/DGtal /path/to/DGtal
```

then compile it with ImageMagick enabled

```
cd /path/to/DGtal
mkdir build
cmake .. -DWITH_MAGICK=ON -DBUILD_EXAMPLES=OFF
make
```

You can see every options of the library using ccmake (the curse interface to cmake) instead of cmake.

Then you have to set up your environment:

- Get the DGtal cmake project skeleton:

  ```
  git clone  https://github.com/dcoeurjo/lectureDG.git
  ```

- Go to practical-work/ folder.

- Compile the examples using `cmake`[1] "out-of-source-build principle":

```
cd DGtalSkel
mkdir build
cd build
cmake .. -DDGtal_DIR=/path/to/DGtal/build
make
```

  The `make` command will build examples located in the source folder (path ".." here). If you want to clean your build, just remove the `build` folder (a "make clean" command also exists in the build folder).

Exercise 1. Conversion from `PGM` to `PBM`

You can find some image manipulations (loading, modifying and writting) in the file exampleImage.cpp. Implement a conversion from `PGM` to `PBM` format using the GenericReader to load the image, and the MagickWriter combined with a functor to convert unsigned char values to either black or white (0 or 1). You can for example use a threshold function to do the conversion.

Exercise 2. Conversion from `PGM` to `PNG`

An example of conversion from `PGM` to `PNG` is given in exampleImage.cpp. Implement your own functor from unsigned char to DGtal::Color.

Exercise 3. Conversion from `PNG` to `PGM`

An example of conversion from `PGM` to `PNG` is given in exampleImage.cpp. You can use the container ImageContainerBySTLVector¡Z2i::Domain, DGtal::Color¿ of `DGtal` to load a PNG image. Then implement your functor from DGtal::Color to unsigned char to convert the image. You can test your code with other input types (like JPEG).

Exercise 4. Image modifications.

Once an image is loaded into the container, you can modify a value using the function setValue. It takes two argument: the first one is the coordinate of the point you want to modify, and the second one is the new value.

Load a `PGM` image, modify some random pixels with new values and write it to a new image.

Exercise 5. Can you draw ?

You are asked here to draw simple cartesian equation. Assuming the center of the image is in $(0, 0)$, draw a circle into an image (remember that the cartesian equation of a circle is $(x-a)^2 + (y-b)^2 = r^2$ with (a, b) the center and r the radius).

Hint Consider testing if a pixel is inside the circle.

[1] `cmake` is a project generator (e.g. makefiles on Unix, Xcode projects on MacOS, VisualStudio projects on MS) from a very simple recipe file (see `CMakeLists.txt`)

Here is a code to convert HSV values (see `https://en.wikipedia.org/wiki/HSL_and_HSV`) to RGB one (the rgb values lies in $[0;1]$)

```
DGtal::Z3i::RealPoint HSVtoRGB( const DGtal::Z3i::RealPoint& HSV )
{
  double fC = hsv[2] * hsv[1]; // Chroma
  double fHPrime = fmod(hsv[0] / 60.f, 6.f);
  double fX = fC * (1.f - fabs(fmod(fHPrime, 2.f) - 1.f));
  double fM = hsv[2] - fC;

  RealPoint rgb;

  if(0 <= fHPrime && fHPrime < 1) {
    rgb[0] = fC;
    rgb[1] = fX;
    rgb[2] = 0;
  } else if(1 <= fHPrime && fHPrime < 2) {
    rgb[0] = fX;
    rgb[1] = fC;
    rgb[2] = 0;
  } else if(2 <= fHPrime && fHPrime < 3) {
    rgb[0] = 0;
    rgb[1] = fC;
    rgb[2] = fX;
  } else if(3 <= fHPrime && fHPrime < 4) {
    rgb[0] = 0;
    rgb[1] = fX;
    rgb[2] = fC;
  } else if(4 <= fHPrime && fHPrime < 5) {
    rgb[0] = fX;
    rgb[1] = 0;
    rgb[2] = fC;
  } else if(5 <= fHPrime && fHPrime < 6) {
    rgb[0] = fC;
    rgb[1] = 0;
    rgb[2] = fX;
  } else {
    rgb[0] = 0;
    rgb[1] = 0;
    rgb[2] = 0;
  }

  rgb += RealPoint( fM, fM, fM );

  return rgb;
}
```

You can use it to shade you circle: considering a pixel (x, y) in the image, compute its polar angle in degree. Calling HSVtoRGB(angle, 1., 1.) will return a nice shading value !