Sehbau: Demo Segmentation Tree

The demo programs for generating the segmentation tree (“baum”) are introduced, called **baumgrau** and **baumfarb**. Program **baumgrau** generates the tree based on the gray-scale image using global thresholding as the divisive (segregation) technique. Program **baumfarb** generates the tree based on the three chromatic channels (RGB) using 2-Means as the divisive technique. The programs can be used to obtain annotations with precise boundaries.

Read	`https://www.researchgate.net/publication/360033329`
Repository	`https://github.com/Sehbau/DemoTree`
See also	`https://github.com/Sehbau/SgrRGB`

The directories in the repository contain the following:

- /**Imgs**	sample images for immediate testing
- /**UtilMb**	Matlab scripts to read the output data files

The programs are available as:
- **baumgrau.exe**, Windows 10 [x64]
- **baumgrau_ubu**, Ubuntu 20.04.3 LTS (Focal Fossa) [x64]

- **baumgfarb.exe**, Windows 10 [x64]
- **baumgfarb_ubu**, Ubuntu 20.04.3 LTS (Focal Fossa) [x64]

The program **baumgrau** takes a (color) image as input and performs a divisive (global-to-local) segmentation based on its gray-scale values (using global thresholding). It generates three maps (by default) that show increasingly local regions, called **Igrau_0.png**, **Igrau_1.png** and **Igrau_2.png**.

The program **baumfarb** performs this divisive segmentation using clustering in the chromatic space (RGB). Since this occurs in three-dimensional space, this takes substantially longer (than the gray-scale analysis) and therefore can take several seconds for large images. The outputted maps are called **Ifarb_0.png**, **Ifarb_1.png** and **Ifarb_2.png**. This process is also demonstrated in a Matlab script called `e_Fleckmentation.m`.

These programs are provided for demonstrating the global-to-local process, but can also be exploited to build an interface for collecting annotation material with precise boundaries. The example scripts `run1img.m` and `run1farb.m` show how to run the two programs and load the output files. More details follow below.

1 Program Use

An image path needs to be specified, here taken from the directory /**Imgs**:

```
> baumgrau Imgs/img1.jpg
> baumfarb Imgs/img1.jpg
```

This will generate the three maps as explained above. In addition, this will generate a file containing the boundary pixels for each region, called **Grau.bonPix** when run with **baumgrau** and **Farb.bonPix** when run with **baumfarb**. How those are loaded will be explained in Section 2. In the following it is explained how the segmentation tree can be manipulated.

1.1 Options

The following long options are available, to be specified with a double dash ’--’; single letter options are not in use.

--depth: depth of the segmentation tree. Default `depth=3`. For `depth=1` no tree is grown: in case of the gray-scale analysis this corresponds to global thresholding only (with a single threshold); in case of the chromatic analysis this means that the 2-means algorithm is applied once only. `depth=4` can be useful for large images, e.g. larger than 1000 pixel for one image side.

--regMinPixNode: minimum number of pixels for a region to be segregated by the divisive technique. This will affect the region count from the 2nd segmentation map on. It will not affect the 1st segmentation map, as that is the input to the 2nd for which segregation starts.

Default equal 6. With larger values, processing occurs more rapidly, but may not capture texture properly anymore.

2 Output

The **.bonPix** files containing the pixels of the region boundaries and are organized as shown in Matlab script `LoadBonPix.m` of directory /**UtilMb**. The first three integer values hold the total number of boundaries (`nBon`) and the size of the image (height, width), assigned to variable `szM`:

```
nBon      = fread(fileID, 1, 'int=>single');
szM       = fread(fileID, 2, 'int=>single');
```

Then follow the individual pixels. The first two entries of a boundary contain the number of pixels (`nPx`) and the depth, the segmentation map where the boundary was taken (assigned to variable `Org`); this latter value is of type unsigned char:

```
nPx      = fread(fileID, 1, 'int=>single');
Org(b)   = fread(fileID, 1, 'uint8=>single');
```

Then the row and column coordinates/indices are read en bloc:

```
Pix.Rw   = fread(fileID, nPx, 'int16=>single');
Pix.Cl   = fread(fileID, nPx, 'int16=>single');
```

The struct is then assigned to cell `APix{b} = Pix;`

While the values are stored with minimal storage, e.g. type `short int` for coordinates, they are here loaded to type `single` in Matlab for convenience.