Texture Extraction tool. User manual.

April 29, 2018

1 Introduction

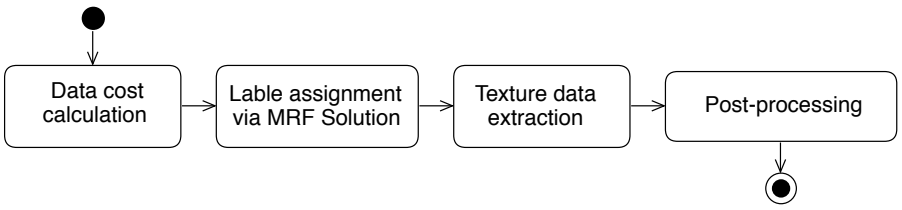
Tool generates texture images for 3D models using camera information from reconstruction tools and source images.

Supported reconstruction tools:

- Visual SFM.
- Bundler SFM.
- 3D laser scanning software developed in the CTU FEE Department of Computer Graphics and Interaction.

Specifications:

- Supports OBJ files for model input.
- Does not generate texture coordinates.
- Supports meshes with multiple objects.
- Written in C++ and can be run from command line.



2 Installation

2.1 Prerequisites

A tool can be build using CMake. (See `https://cmake.org` for platform specific instructions).

Required libraries are:

- **OpenCV** `https://opencv.org`
- **GLM** `https://glm.g-truc.net/0.9.8/index.html`
- **TBB** `https://www.threadingbuildingblocks.org`

2.2 Building

0. Dowload the project and navigate to the root folder of the project.

```
cd path\to\the\project\root
```

1.To keep the directory clean first create a new directory for building files.

```
mkdir cmake.
```

The directory should now look like this:

```
...
TextureExtractorV2.....the directory of source codes
CMakeLists.txt.....main CMake file
example.....directory with demo example
cmake ..... directory with building files
...
```

2. Navigate to cmake folder:

```
cd cmake
```

3. Use CMake to generate make file:

```
cmake ..
```

If all the libraries are found and linked, CMake should finish without errors. If you get any errors, check console output or logs to identify the problem.

4. Build a tool using make:

```
make
```

`tex_extract` executable should now be in the `cmake` folder

5. Move executable to a root folder and navigate back:

```
mv tex_extract ..
cd ..
```

`tex_extract` executable should now be in the root folder.

(For the Windows platform, please refer to the official CMake documentation for the instructions.)

3 Demo

Now that we have a working executable, we can check its functionality using example dataset provided with this project.

0. Navigate to the root folder with executable (if not there already).

1. Start an example demo. (It may take couple of minutes, depending on you system.)

```
./tex_extract example/demo.ini
```

When the extraction is finished, it should output the result into the `example/result` folder.

Example model consist of 4 seperate objects. Four textures are generated. Each texture can be identified by a object name added to a texture name.

NOTE: You may need to flip final textures vertically (y = -1) depending on what rendering software you use.

4 Using your own datasets

To run a tool for you own objects you will need the following files from any of the supported reconstruction tools.

1. **OBJ model with generated UV coordinates.** You can use Blender to automatically generate texture coordinates. See `https://wiki.blender.org/index.php/Doc%3A2.6/Manual/Textures/Mapping/UV/Unwrapping`
2. **Bundle file.** File that describes camera calibration and position. Usually named `bundle.rd.out`. Gets generated as an output from the reconstruction tool.
3. **Photo list**. A text file that lists paths to source images in the same order as the bundle file. Also gets generated by a reconstruction tool.

4. **Folder with source photographs**. A set of undistorted source images that were used for the reconstruction.

You should fill those paths in a configuration INI file. You can use example `demo.ini` file as a base file that you will need to modify. Or you can ask tool to generate template INI file using the following command:

```
./tex_extract -genIni
```

It will generate a `example.ini` configuration file, that you can modify to suit your needs.

To get the tool running you will need to at least fill up the following fields in the INI file:

- `objFilePath` : path that leads to a OBJ file.
- `cameraListFilePath` : path that leads to photo list file.
- `cameraInfoPath` : path that leads to bundle file (`bundle.rd.out`).
- `photoFolderPath` : path to a folder that will be added to a paths in a photo list file. So if each entry in photo list starts with "`folderA\...`", then `photoFolderPath` should point to a folder that contains `folderA` (NOT to `folderA` itself).

By default tool will output new textures in to a folder from which the execution was started (to change it, see INI configuration explanation in the next section).

To start the execution simply pass a ini file to a tool as an argument:

```
./tex_extract example.ini
```

5 Reading precomputed data costs

After the data costs are computed, by default they are save into a file that ends with "`...datacost.txt`". You can use this file to read precomputed values. This allows you to compute them once and then change color consistency strictness without having to rerun data cost calculation again.

To read data cost file, you should change the following fields in the config file:

- `getDataCostsFromFile` : set it to `true`
- `dataCostsFilePath` : path that leads to data cost file.

6 Reading precomputed labeling

After the labeling is computed, by default they are save into a file that ends with "`...labeling.txt`". You can use this file to read precomputed values. This allows you to compute them once and then change texture size or post-processing options without having to rerun labeling calculation again.

To read labeling file, you should change the following fields in the config file:

- `getLabelingFromFile` : set it to `true`
- `labelingFilePath` : path that leads to labeling file.

7 INI file configutration

In this section the meaning of all INI file entries will be explained.

7.1 objFilePath

Path that leads to a OBJ file of reconstructed model. OBJ file should have texture coordinates generated beforehand. Tool supports meshes with multiple objects.

7.2 cameraListFilePath

Path to a text file that lists all the source images in the same order they are described in bundle file. Gets generated by a reconstruction tool.

7.3 cameraInfoPath

Path to a bundle file. File that describes camera calibration and position. Usually named `bundle.rd.out`. Gets generated as an output from the reconstruction tool.

7.4 photoFolderPath

Path to a folder that will be added to a paths in a photo list file. So if each entry in photo list starts with "`folderA\...`", then `photoFolderPath` should point to a folder that contains `folderA` (NOT to `folderA` itself)

7.5 newTextureFolderPath

Output folder. All new textures and saved computations will be written to this folder.

7.6 textureWidth

Width of the new texture in pixels. Will be the same for every texture if more than one texture is generated

7.7 textureHeight

Height of the new texture in pixels. Will be the same for every texture if more than one texture is generated

7.8 getLabelingFromFile

Flag that if set to `true` signifies we should read precomputed labeling from file. Labeling gets read from path provided in the `labelingFilePath` argument of the config.

7.9 writeLabelingToFile

Flag that if set to `true` signifies that we should write calculated labeling into the file.

7.10 getDataCostsFromFile

Flag that if set to `true` signifies we should read precomputed data costs from file. Data costs gets read from path provided in the `dataCostsFilePath` argument of the config.

7.11 writeDataCostsToFile

Flag that if set to `true` signifies that we should write calculated data costs into the file.

7.12 verbose

Is set to`true` all the logging output will be written to the console. If set to `false` only warnings and errors will be displayed

7.13 imageFormat

Image format for the output textures. Our tool uses OpenCV for image manipulation so it naturally supports all the same image formats as OpenCV. (eg. png, jpeg).

7.14 threadCount

Number of working thread for the cost calculation step. Must be ¿= 1.

7.15 labelingFilePath

Path that leads to precomputed labeling file. Gets read if `getLabelingFromFile` flag is set to true.

7.16 dataCostsFilePath

Path that leads to precomputed data cost file. Gets read if `getDataCostsFromFile` flag is set to true.

7.17 addProjectNameToFiles

If set to true each generated file will have a prefix added to it. Prefix can be defined using `projectName` variable in the config.

7.18 projectName

Prefix that will be added to each generated file if `addProjectNameToFiles` flag is set to true.

7.19 doGloabalAdjustment

If `true` Tool will perform global adjustment step.

7.20 doSeamLeveling

If `true` Tool will perform seam leveling step.

7.21 doTextureExtension

If `true` Tool will perform texture extension step.

7.22 doColorConsistency

If `true` Tool will perform color consistency check.

7.23 colorConsistencyThreshold

Color consistency deviation threshold delta. Is a real number between 0 and 1. The higher it is the stricter the color consistency check.

7.24 BVHMinNode

Defines a how many polygons should belong to minimal BVH node. Should be ¿1. If node size is lower or equal to this value it is not separated any further.

7.25 genRawTexture

DEBUG flag. Defines if along side final texture a 'raw' texture should be generated. Raw texture does not have any modifications performed on it.

7.26 genLevelingTexture

DEBUG flag. Defines if along side final texture a 'leveling' texture should be generated. Leveling texture shows a effect of seam leveling.

7.27 genMaskTexture

DEBUG flag. Defines if along side final texture a 'mask' texture should be generated. mask texture shows defined area off all texture patches. Everything outside of those edges is a result of texture extension.

7.28 genLebelingTexture

DEBUG flag. Defines if along side final texture a 'labeling' texture should be generated. Labeling texture shows how different photograph get assigned to each polygon. Different colors signify different source photographs.

7.29 genGlobalTexture

DEBUG flag. Defines if along side final texture a 'global' texture should be generated. Global texture shows a effect of performing global color adjustment on top of the raw texture.

7.30 justRender

DEBUG flag. Defines if what tool should do is just render replicas of all the source images using mesh and texture, and finish running. Renders get outputed to path defined in `resultRenderFolder`.

7.31 renderInTheEnd

DEBUG flag. Defines if what tool should do is render replicas of all the source images using mesh and texture after doing the main run. Renders get outputed to path defined in `resultRenderFolder`.

7.32 resultRenderFolder

DEBUG variable. Rendering of the replicas gets outputed here. Gets used if `renderInTheEnd` or `justRender` is set to true.