# .oO-| GraPeS |-Oo.
# version 1.0
# User Manual

Olivier Teboul

April 2011

## Contents

# 1 Introduction

GraPeS is a software created for academic purposes by Olivier Teboul, in Ecole Centrale Paris, France and supported by Microsoft France and Microsoft Research Cambridge. If you intend to use it in a publication please cite explicitely the software GraPes and the related publication [1]. The name GraPeS means "a GRAmmar ParsEr for Shapes" and comes from the shapes of the binary parse trees associated to the obtained parses.

GraPes provides a tool to analyze the facade images semantically and geometrically using a class of shape grammars called Binary Split Grammars (BSG). It enables the user to:

- Manually rectify an input image.
- Efficiently parse a rectified image with a shape grammar.
- Use one of the 3 proposed grammars.
- Define its own shape grammars in XML.
- Use 3 different predefined *merits* or data-term.
- Define its own merits in text files.
- Visualize the intermediary and final results in images as well as in dot language format.
- Test the different options on provided examples.

This manual briefly explains how to perform all these tasks.

# 2 Load Inputs

The inputs of the parsing algorithm are a rectified image of a facade and a shape grammar. To load an image, there are basically three options:

1. Enter the path of the image in the `image` field and press `Enter`
2. Browse your computer using the button lying on the left of the `image` field.
3. Drag-and-drop an image from your computer to the application.

The load a grammar, GraPeS proposes the three same possibilities. The BSGs are XML files that are parsed by GraPeS. In case you modify the proposed grammars or define your own ones, the parsing may fail. There are two ways to check that the grammar you have loaded has been correctly parsed.

1. The parsing output is given on the command line.
2. The combo box dictionary is updated with the terminal elements and their corresponding colors if the grammar has been correctly loaded.

# 3 Rectification

If the loaded image is not rectified, it is easy to rectify it. To do so, select `rectify` is the `mode` combo box. Then, use the left button of the mouse to select the limit of the facade, and double click for the last point to launch the rectification. The selected quadrilateral is exactly the one that will constitute the rectify facade.

# 4 Output

You can specify were to dump the file output during the parsing. The default path is `user/username/pictures/grape` and it is created in case it does not exist. You can change this output path by entering another one in the corresponding field. It is possible to let a prefix for the output files (by default `untitled`).

These files are:

- the best parse blended with the rectified image
- the best parse as an image of symbols
- the merits for all the terminal elements
- the prior used in the data-driven exploration
- the image of pixel-wise best terminal elements (no procedural shape prior)
- a gmm classifier file, in case the reward is GMM and the user manually painted some regions of the image.
- the brush strokes of each class, in case the reward is GMM and the user manually painted some regions of the image.
- a .dot file, that can be turned into a .pdf, .eps, .svg, etc. by using graphviz (www.graphviz.org). To turn it into a pdf file for instance, do:

```
dot -Tpdf [.dot file] -o output [.pdf file]
```

# 5 Choose and define a Reward

3 predefined rewards can be used:

1. RF: randomized forest classifier build with RFlib. GraPeS provides an example for haussmannian style. It has to be specified in the field `reward file`. The features used are patches which sizes are defined in the .rf file.
2. GMM: Gaussian mixtures models. There are 3 ways to specify a GMM.
   (a) By painting some strokes on the image. To do so, the reward should be `gmm` and the selection mode `paint`. Then, choose in the combo box `dictionary` the terminal elements that you want to specifY (some can be omitted in they are not needed) and directly paint onto the image. After the parsing, a gmm file appears in the output directory.

(b) By specifying the gmm file (.txt) in the field `reward file`.

(c) By specifying an image (.png) with brush strokes in the field `reward file`.

3. Hue: no file is needed since it is totally unsupervised, but it only works on grammars with two terminal symbols called `wall` and `window`.

If you want to implement your own merit functions, based on more complex features for instance, you can pre-process the data (rectify image), and provide .txt files. There should be as many text files as the number of terminal elements of the grammar, and they should have the format `prefix_terminal.txt`, were `prefix` is given in the field `reward file`. These text files are ascii files, that contains for each pixel its merits. An example of such files are given for the New York building.

# 6 Known bugs

So far, there is a known bug that occurs while closing the application. If you close the main window you may have an error due to the python thread. This bug does not affect the parsing algorithm.

If you have any questions or suggestions, please feel free to contact me at olivier.teboul@ecp.fr

# References

[1] Olivier Teboul, Iasonas Kokkinos, Panagiotis Koutsourakis, Loic Simon, and Nikos Paragios. Shape Grammar Parsing via Reinforcement Learning. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3105–3112. IEEE, 2011.