Documentation of Shape Analyzer

Version 1.0

Emanuel Laude und Zorah Lähner

December 16, 2014

Contents

CHAPTER 1

Introduction

This LaTeX template is designed for the creation of thesis documents (bachelor, master, phd) and targets both beginner and experienced users of LaTeX. It supports all basic functionality and requirements of a technical document such as the inclusion of graphics, math, tables, references, bibliography and much more. In contrast to a standard LaTeX document this template not only loads all state of the art packages (`preamble/packages.tex`) to provide the best functions for each task, but also includes a separate document for the style/layout of the document (`preamble/style.tex`). It therefore tries to separate functionallity and layout as much as possible. And the best, everything is documented in the code and furthermore in a separate documentation file (`TemplateDocumentation.pdf`)

This document shows in **??** a general tutorial for LaTeX with links to the documentation for further tasks. You can view the underlying code in file `content/demo/latextutorial.tex` or in this document in **??**.

The code of the template itself is documented in `TemplateDocumentation.pdf`.

CHAPTER 2

Installation

ShapeAnalyzer is completely written in `C++` heavily making use of the recent `C++11` standart in the code. Moreover it builds up on the frameworks `VTK` version 6.1.0 or greater and `Qt` version 5.0 or greater for the visualization and rendering of the shapes and the creation of the graphical user interface respectively. For all matrix and vector related computations including the computation of the Laplace-Beltrami eigenvectors the libraries `Petsc` (basic linear algebra including basic solvers for linear systems) and `Slepc` (sparse eigensolver) are used. (Petsc and Slepsc - the funny dogs. This sounds like the names of funny cartoon series characters for kids.)
In order to compile *ShapeAnalyzer* it has be ensured that both, a recent `C++` compiler that fully supports the `C++11` standard (e.g. `gcc` 4.7 or newer) as well as `cmake` version greater ? is available . Furthermore all the aforementioned libraries and frameworks have to be installed.

2.1 Installation of Qt5

Since the most recent version of Qt currently (Dezember 2014) available via apt-get on Ubuntu is less than 5, it is recommended to download the most recent pre-compiled Qt5 package from the homepage of Qt: For installation just launch the installation assistant.

Hint: In case the installation file cannot be opened and executed you may have to make it executable via

```
sudo chmod +x <filename>
```

2.2 Installation of VTK

As it is the case for Qt the most recent version of VTK currently (Dezember 2014) available via apt-get on Ubuntu is less than 6.1. Therefore it is recommended to compile and install VTK 6.1 or newer from source.

2.3 Installation of Petsc

2.4 Installation of Slepc

asdf

2.5 Compiling the project

Finally follow these steps to compile ShapeAnalyzer

CHAPTER 3

Classes

TODO, + class diagram

3.1 ShapeAnalyzer

3.2 Shape

3.3 Correspondence

3.3.1 PointCorrespondence

3.3.2 FaceCorrespondence

CHAPTER 4

Exceptions

There are two types of exceptions you might want to throw: Exceptions (4.1) and Errors (4.2). The first one is kind of optional, it includes all normal exceptions C++ provides, the second one is for non-fatal problems that might occur and that can actually be handled. In both cases do not use the `throws` keyword in the function signature.

When you are using VTK classes you can use the ErrorObserver (4.3) to handle VTK specific exceptions.

4.1 Exceptions

- **Throw when** you have special cases that are due to previous faulty programming or unexpected errors
- **Documentation** not necessary
- **Found in** C++ std library

Any exception will be caught by the main program and shown within an error message. The program will then terminate. In order to make debugging easier a short description of the problem and the function name is useful.

```cpp
if(s >= points_->size()) {
        throw invalid_argument("Source point (" + to_string(s) + ")
        larger than number of points (" + to_string(points_->size()) + ")
        in " + __PRETTY_FUNCTION__);
}
```

4.2 Errors

- **Throw when** you notice faulty data or the like that can not be processed, but it is not necessary to terminate the whole program
- **Documentation** use the `@throws` command and define the Error class
- **Found in** util or the namespaces folder

Error is a ShapeAnalyzer specific class that inherits from exception but does not add any functionality. Every namespace has its own Error class that is named `<namespace>Error`. This allows more specific error handling by catching the different classes.

If your exception is not due to faulty programming logic or you think it can be solved by a higher level class, use this classes instead of a std exception. This can be for example detecting a mesh with borders but your algorithm does not support borders. Most of the time this means you terminate your algorithm without a result and let the user know about the problem with an pop-up. Again a short description of the problem and the function name is useful. You can use `QMessageBox::warning` for the pop-up but you must give the QMessageBox the ShapeAnalyzer as the parent widget, so catch the Error in your Tab or ContextMenuItem (see **??**) and create the QMessageBox there.

It is possible to give null as the parent widget but you must not do that. It might cause unwanted behavior like the QMessageBox not appearing in the foreground.

If your function is throwing an Error use the `@throws` or `@exception` keyword in the Doxygen documentation. Give the class of the error and a short description of what might cause this error.

```
if(ierr != 0) {
    throw LaplaceBeltramiError(string(
        "Error occured in computation of Laplace-Beltrami eigenfunction
  (Probably due to corrupted shape)")
  .append(" in ").append(__PRETTY_FUNCTION__));
}
```

```
try {
    PetscFEMLaplaceBeltramiOperator laplacian(shape_, 100);
    vtkSmartPointer<vtkDoubleArray> eigenfunction
        = laplacian.getEigenfunction(i);
} catch(LaplaceBeltramiError& e) {
    QMessageBox::warning(parent, "Exception", e.what());
}
```

Some namespaces do not have their own Error class yet. If you need it, just create it. You can use `MetricError` as a template.

4.3 ErrorObserver

- **Use when** you want to handle VTK specific errors
- **Found in** util

VTK classes in general do not throw exceptions instead they write error and warning messages to observers. If you want to handle those problems use the ErrorObserver class.

```
// vtk object that we want to observe
vtkSmartPointer<vtkTriangleFilter> triangleFilter
```

```
  = vtkSmartPointer<vtkTriangleFilter>::New();
// new ErrorObserver object
vtkSmartPointer<ErrorObserver> triangleObserver
  = vtkSmartPointer<ErrorObserver>::New();

// list the observer to the triangle filter
// if you are not interested in warnings leave the second line out
triangleFilter->AddObserver(vtkCommand::ErrorEvent, triangleObserver);
triangleFilter->AddObserver(vtkCommand::WarningEvent, triangleObserver);

// do the things you want to do with your vtk object
triangleFilter->SetInputConnection(input);
triangleFilter->Update();

// see if messages were written into the observer and handle that
if (triangleObserver->GetError()) {
        showErrorMessage("The file cound not be opended",
                      triangleObserver->GetErrorMessage());
        return;
}
if (triangleObserver->GetWarning()) {
        showErrorMessage("There was a warning reading the file",
                       triangleObserver->GetWarningMessage());
}
```

Notice that if you use the same ErrorObserver for several objects or call several functions before checking the messages, you will only receive the latest message.

List of Figures

List of Tables

APPENDIX A

First chapter of appendix

A.1 Parameters

Todo list