2D Tele-Registration Code/Lib/Demo

– User Guide

Maintainer: Kangxue Yin

yinkangxue@gmail.com

1. Introduction

Thanks for concern on our work. If you use the code/lib/demo for your research project, please cite our paper:

@ARTICLE{Telereg2013
title = {“Mind the Gap”: Tele-Registration for Structure-Driven Image Completion},
author = {H. Huang and K. Yin and M. Gong and D. Lischinski and D. Cohen-Or and U. Ascher and B. Chen},
journal = {ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA 2013)},
volume = {32},
issue = {6},
pages = {174:1--174:10},
year = {2013},
}

If you are using / will use them for commercial purpose, please contact Hui Huang / Kangxue Yin for patent issues.

Note that we do NOT provide any support for the open source package.

The package only contains our core tele-registration algorithm for curves. It does not include any image processing module. To apply it to image registration/completion, you should combine the use of our curve registration with other available image processing code/software. Detecting correspondences among curve endpoints without color information is less robust, so we also provide the programming interface that allows user to provide correct correspondences for the tele-registration to start. (Please refer to section 3).

Description of files

telereg-demo:
This directory contains the demo program, DLL libs required, and a few sample data files of curves.

telereg-lib:
This directory contains the class lib for our 2d tele-registration algorithm.

telereg-source:
Source codes of lib and demo.

telereg-doc

This user guide.

2. Run the Demo

2.1 Data Formation

After the demo program starts, it will load a curve data file “curves.txt” automatically. The data file contains information of the following 3 parts:

Curves; Grouping of curves; Active endpoints of curves.

Curves are the curves input for registration. They will be normalized by the program into the 2d space [0.2,0.8]×[0.2,0.8]. An example of curves storage formation:

```
2          //  Number of curves
4          //  Number of points of 1st curve
0.1 0.1
0.2 0.1
0.3 0.2
0.4 0.2
3          // Number of points of 2nd curve
0.5 0.6
0.4 0.5
0.4 0.4
```

Grouping of curves describes how the curves are grouped. For the application of non-overlap image registration described in our paper, all curves lie on the same image piece are considered as one group. The number of groups equals the number of image pieces. Suppose we have 12 curves, here is an example of grouping:

```
3          // number of groups
4          // number of members of first group
0 5 7 8    // IDs of curves in first group
3          // number of members of second group
1 2 4
5
3 6 9 10 11
```

Active endpoints of curves. Each curve has 2 endpoints. When we say an endpoint is active, that means it will be considered by the registration process. For the image registration application, they are the endpoints close to gaps among image pieces. For a curve i, its two endpoints are indexed by i*2 and i*2+1. Suppose we have 12 curves, here's an example of active endpoints,

```
0 5 8 17 20 21
```

It represents that the 1st endpoint of curve 0, the 2nd endpoint of curve 2, etc. are active.

To know more about the data formation, please open and read the examples under directory "telereg-demo". Also refer to the data loading code in "UI.cpp".

2.2 Operations

With the data file provided, the demo program will be launched. The interface is shown as fig1.

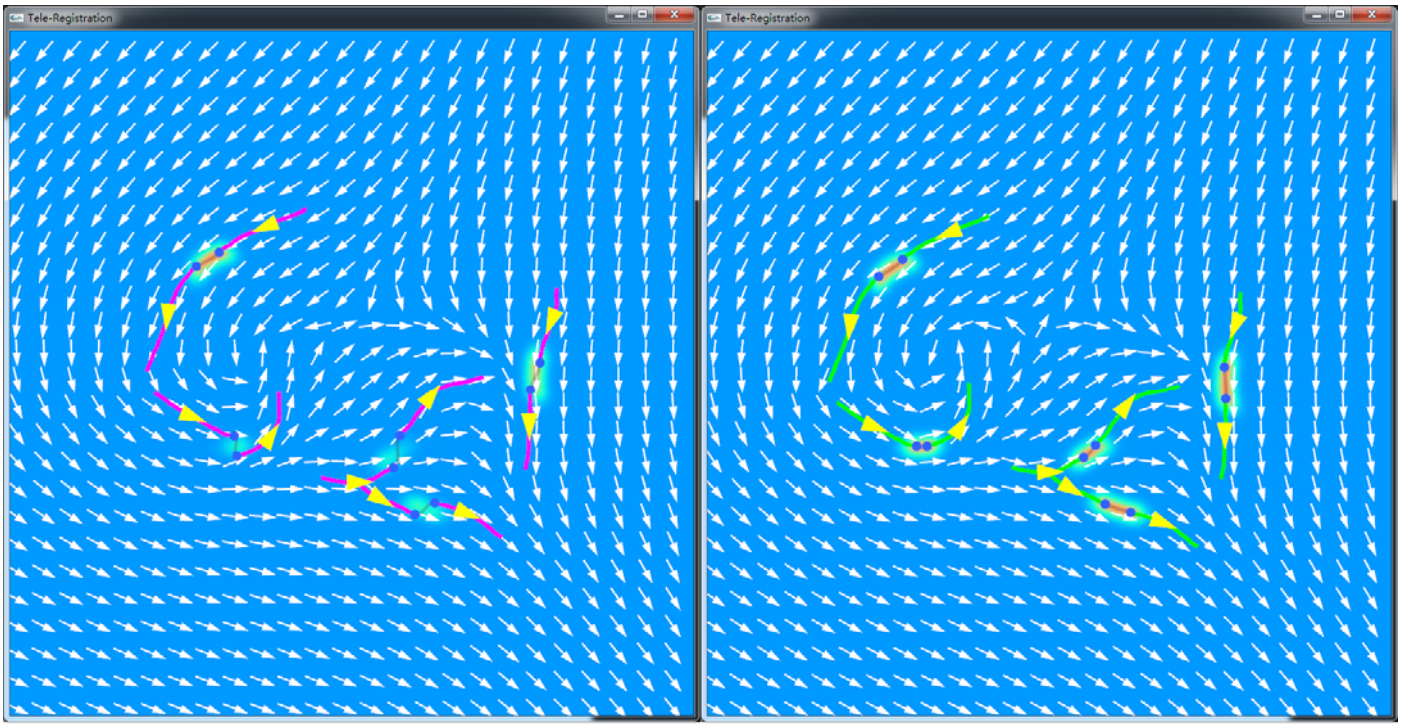

Figure 1. Before and After Tele-registration.

You can press ‘r’ on the keyboard to start the registration and get the result shown in fig1b. Then press ctrl+s to save the result curves in the file “rescurves.txt”. You can also press “TAB” to switch between view of result and view of input.

3. Programming Interface

The 2d tele-registration algorithm is implemented by the “tele2d” class. You could have a look at its header “tele2d.h” to know its interface better. Here is a piece of example code that use our tele2d class:

```
/* load data…… */
CURVES curves;
std::vector<std::vector<int>> group ;
std::vector<int2> endpoints ;
load_Curves( "curves.txt", curves, group, endpoints );
tele2d *teleRegister = new tele2d( 100, 0.02, 0.1 ) ;          // initialize parameters
teleRegister->init( curves, group, endpoints) ;                // write data
teleRegister->runRegister() ;                                  // start registration
teleRegister->outputResCurves( "rescurves.txt") ;              // output curves
```

The constructer:

tele2d::tele2d(int res, double h, double w) ;

offers user interface to initialize the parameters:

res - resolution of the vector field. 100 works good. If you want it run faster, a smaller value could be used.

h - sigma2 in the paper. It controls the smoothness of scalar field(or magnitudes of vector field).

w - lambda in the paper, it's the weight of scalar field(or magnitudes of vector field) in the energy

function.

Then transfer data from client to server:

```
teleRegister->init( curves, group, endpoints    ) ;
```

Here curves, group and endpoints are exactly what described in section 2.1.

If you have correspondences detected in your own code, you can transfer it to **teleRegister** through the interface

```
void init( CURVES curves, std::vector<std::vector<int>> group,
                std::vector<int2> endpoints, std::vector<int2> corres    );
```

The statement

```
teleRegister->runRegister() ;
```

starts the process of tele-registration.

After registration done, you can access the result

```
teleRegister->resCurves
```

to do any further processing in your own code.

Or save it with

```
teleRegister->outputResCurves( "rescurves.txt") ;
```

If you want to save bridged curves, you can use the statement

```
teleRegister->outputResCurves( "rescurves.txt", true) ;
```

4. Performance

This implementation of tele-registration requires around 1 or 2 seconds to converge to the result. If it's not fast enough for you, I suggest you optimize and parallelize the code of energy function.