2D Tele-Registration Code/Lib/Demo

Kangxue Yin

yinkangxue@gmail.com

1 Introduction

The package only includes our core tele-registration algorithm for curves. It does not include any image processing module. To apply it to image registration/completion, you are supposed to 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 programming interface that allows user to provide correct correspondences to registration algorithm. (Please refer to section 3).

Description of files

telereg-demo:

This directory contains the demo program, DLL libs required by the demo, and a few sample files of curves.

telereg-lib:

This directory contains the class lib of our 2d tele-registration algorithm.

source:

Source codes of lib and demo.

telereg-doc:

This document.

2 Run the demo

2.1 File formats

When the demo program starts, it loads file "curves.txt" automatically. The data file contains information of the following 3 parts:

```
Curves;   Groupings of curves;   Active endpoints of curves.
```

Curves refer to the curve input to registration. They will be normalized by the program to fit into the 2d space $[0.2, 0.8] \times [0.2, 0.8]$. An example of curve file:

```
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
```

Groupings of curves describe how the curves are grouped. In the application of non-overlap image registration described in our paper, all curves lie on the same image piece are considered to be in the same group. The number of groups equals the number of image pieces. Let's suppose we have 12 curves, here is an example of groupings:

```
3          // number of groups
4          // number of members of 1st group
0 5 7 8    // IDs of curves in 1st group
3          // number of members of 2nd 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 paired with an endpoint of another curve by the registration algorithm. Non-active endpoints will not be paired. In the image registration application, they are the endpoints which are close to gaps among image pieces. For a curve i, its two endpoints are indexed as $i * 2$ 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 data formats, you can open and read the examples under directory "telereg-demo". You may also also refer to the data loading code in "UI.cpp".

2.2 Operations

With the data file provided, the demo program is able to be launched. The interface is shown in Figure 1.

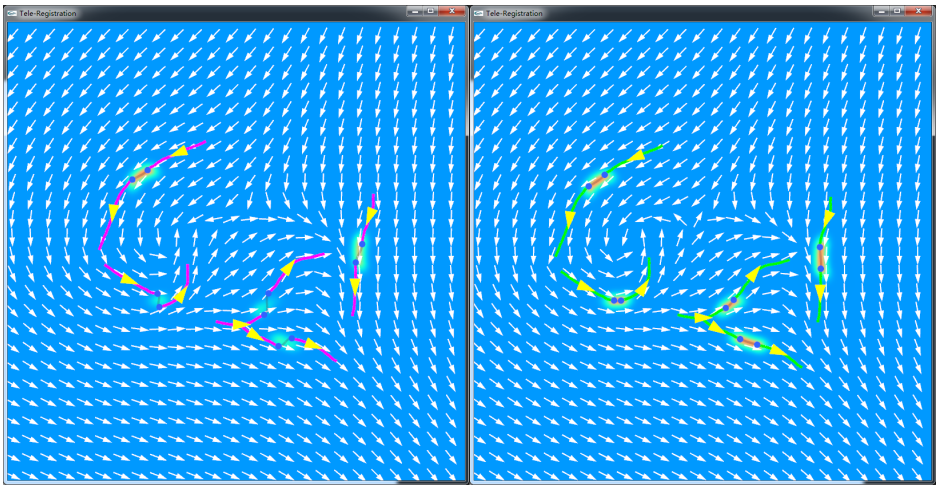

Figure 1: Before and after tele-registration

You can press 'r' on the keyboard to start the registration and get the result shown in Figure 1(b). 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 can have a look at its header "tele2d.h" to check its public interface. Here is

a piece of example code that use the tele2d class:

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

The constructor:

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

offers interface to initialize the parameters:

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

h - σ_2 in the paper. It controls the smoothness of scalar field(magnitudes of vector field).

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

Then send data to the object:

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

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

If you have correspondences detected in your own code, you can send it to teleRegister through the function

```
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 by

```
teleRegister->resCurves
```

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 to you, you may optimize and parallelize the code of energy function.