Image Registration
Lecture 15: Data Structures & Other RGRL Features

Prof. Charlene

Outline

- Review the basic toolkit components
- Spatial data structures
- Range data example
- Other toolkit topics:
  - ROI
  - Initializer
  - Multiple stages (multiresolution)
  - Multiple feature types
- Looking ahead to the rest of the semester

Basic Components

- `rgrl_feature`
- `rgrl_feature_set`
- `rgrl_scale`
- `rgrl_match_set`
- `rgrl_data_manager`
- `rgrl_transformation`
- `rgrl_converge_status`
- `rgrl_matcher`
- `rgrl_weighter`
- `rgrl_scale_estimator`
- `rgrl_estimator`
- `rgrl_convergence_tester`
- `rgrl_feature_based_registration`

3

Basic Loop

- Input the moving and fixed feature sets and an initial estimate
- Specify scale estimator, weighter, matcher and estimator
- Loop
  - Compute matches (rgrl_matcher)
  - Estimate scale
  - Run IRLS with fixed scale
  - Re-estimate scale
- Until converged

4

Variations and Generalizations

- Mixes of feature types
- Multiple stages (resolutions)
- Multiple initial estimates
- View-based registration

5

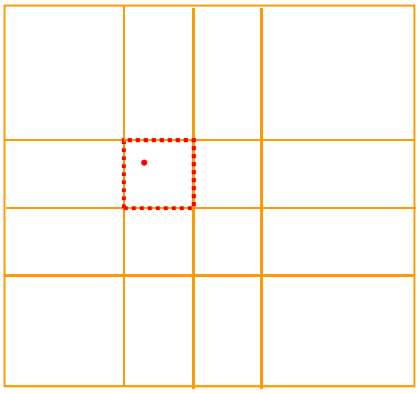
Spatial Data Structures

- `rgrl_feature_set` includes 3 search member functions:
  - `nearest_feature`
  - `features_within_distance`
  - `k_nearest_features`
- Naïve implementations of these functions would require O(N) search time for each "query"
- Thus, we need spatial data structures to make these queries efficient. Three are used (`vxl_src/contrib/rsdl`)
  - Binning data structures (`rsdl_bins` and `rsdl_bins_2d`)
  - Digital distance maps (`rsdl_borgefors`)
  - K-d trees (`rsdl_kd_tree`)

6

Binning Data Structures

- Divide rectangular region into rectangular bins
  - This is really just a coarse array
- Feature point locations and associated values are stored in a list associated with the bin the location falls into
- Region searches:
  - Conducted by examining all bins that intersect the query region
- Nearest neighbor searches:
  - Spiral search starting at the bin containing the query point.



7

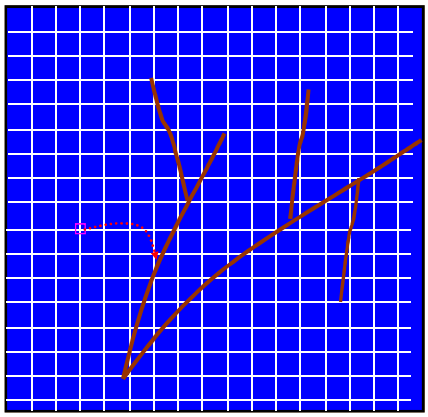
Binning Data Structures

- Advantages:
  - Simple and flexible
  - Insertions, queries and deletions are, on average, quite fast for sparse, scattered queries
- Disadvantages:
  - Works poorly for clustered data

8

Digital Distance Maps

- Given:
  - Feature points organized as points along a series of contours
- At each pixel in the image record:
  - Pointer to closest feature point
  - Distance to this feature point



9

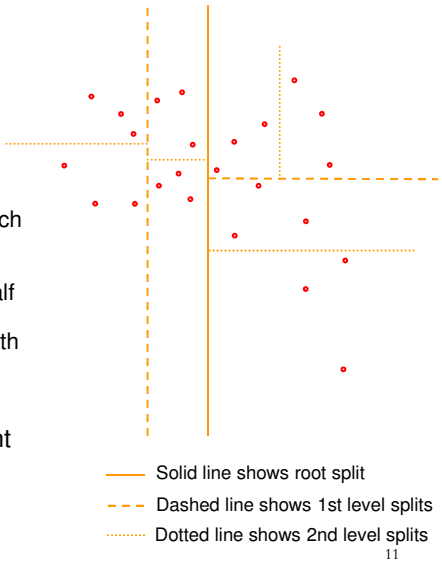
Digital Distance Maps

- Computed in time proportional to the area of the image using digital distance approximations
- Works well only with static data
- Once the map is computed, only O(1) time is required to find the nearest point on the contour at each pixel

10

k-d Tree

- Multi-dimensional analog to a binary search tree
- Partions region **and** partions data
- Recursive tree construction
  - Example shown in 2d
- At each node of the tree
  - Find the dimension *x* or *y*, across which the data is most widely spread
    - Say this is *x*
  - Find value of x, call it *xm* such that half the points have x values less than x.
  - Partition the data, assigning points with x < xm to the left subtree and the remaining points to the right
- Continue down the tree until each node has fewer than a small constant number of points



11

k-d Tree Search

- Region:
  - Recurse down the tree
  - At nodes completely contained inside the query region, gather all points
  - At nodes completely outside the query region, end this branch of the search
  - At nodes partially inside the query region, continue down each branch of the tree
- c points nearest to a query point
  - Go down the tree until node is found containing the query point
  - Gather points, sorting them
  - Proceed back up (and perhaps down) the tree, gathering and inserting points until the nodes that must be added are further from the query than the current c-th point

12

k-d Tree Discussion

- Construction requires O(N log N) worst-case time and space
- Queries require O(log n + c) on average, where c is the number of points “reported” (returned)
  - Worst-case is linear, but this does not happen in practice
- Designed for static data, but can handle a small number of insertions and deletions

13

Spatial Data Structures: Summary

- `rsdl_bins` and `rsdl_bins_2d`
  - Cache dynamic spatial information
  - Can’t handle cluttering
- `rsdl_borgefors`
  - Represent static set of 2d edge or contour points
  - O(1) access
- `rsdl_kd_tree`
  - Multi-dimensional version of binary search tree
  - Static structure
  - Fast average case access, but slow worst-case
- Potential research paper topic:
  - Data structures for registration

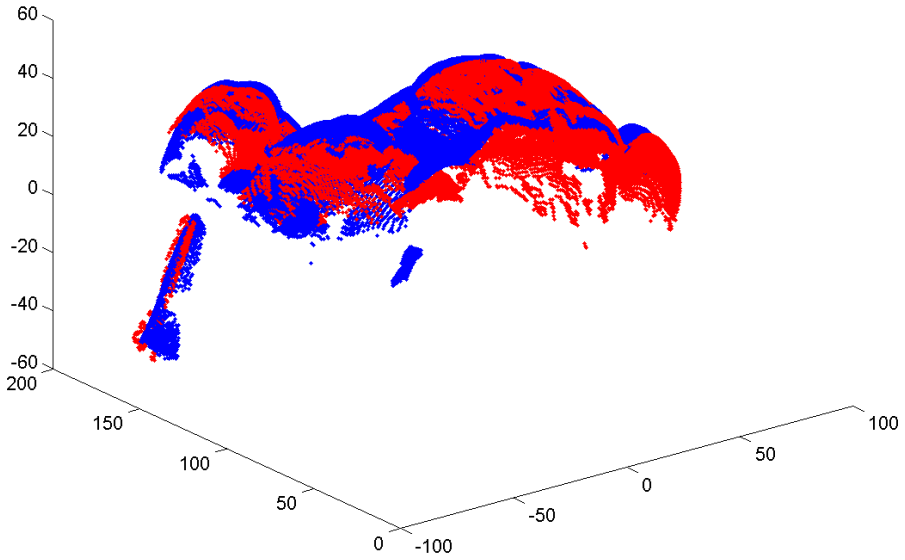
14

Range Data Example

- The famous Stanford Bunny
- Points stored in k-d tree
- Surface normals calculated by finding the c nearest points to each data point and fitting a planar surface

15

Animation



Advanced Properties of RGRL

- Straightforward stuff (conceptually, at least)
  - Regions of Interest (ROIs) and masks
  - Initializers
  - Multiple stages
  - Multiple feature types
- Feature extraction
- More complicated stuff
  - Initialization techniques

17

Masks

- Mask (rgrl_mask):
  - Used to indicate which pixels in an image are currently “active” and to be used in an operation
  - In particular, you may want to register sections of images rather than entire images.
  - Subclasses implement:
    - Binary image mask
    - Rectilinear mask
    - Circular / spherical masks

18

Initializer (rgrl_initializer)

- Provide initial information to the registration engines, including ROIs, estimator, initial estimates and initial scale
  - In examples thus far this information has been provided to the registration engine “by hand”
- Provide multiple initial estimates
  - Test different initial estimates in attempt to obtain global convergence
- Lectures 18 and 19 will discuss initialization techniques

19

Multiple Stages

- Thus far we’ve assumed registration has had a single stage
  - Although you might have used multiple stages in your solution to HW 6, Question 4
- Registration can proceed in multiple stages, however.
- The solution to the registration problem at each stage is used to initialize registration at the next stage
- Each stage contains
  - Moving and fixed feature sets
  - Matcher
  - Weighter
  - Xform estimators

20

Multiple Stages (continued)

- Information on stages need only be specified by
  - Constructing `rgrl_data_manager` with the single argument `true`
  - Calling `rgrl_data_manager::add_data` for each stage, with stages indexed by 0, 1, etc.
- The registration engine (`rgrl_feature_based_registration`) handles multiple stages automatically by receiving the initial stage index as an argument
  - Either from the initializer or directly to the `run` member function
  - The registration functions count down until stage 0 has completed

21

Multiple Stages == Multiple Resolutions

- Many registration techniques work first on coarser resolution versions of the images, extracting features at each resolution
- Intuitively, we can think of resolutions and stages as the same thing
  - This requires (potentially) scaling the transformation and ROIs between resolution
  - This occurs if the feature-set is in image coordinates
- BUT the notion of a stage is more general than resolution.

22

Example

- rgrl/examples/registration3.cxx
- Registration of multiresolution (2 resolutions) feature sets from a pair of retinal images
- Most of the foregoing discussion is mirrored in this example

23

Example

```
int  main( int argc, char* argv[] )
{
  // ….
  //read in the feature files
  //
  feature_vector moving_fps_high_res;
  feature_vector moving_fps_low_res;
  feature_vector fixed_fps_high_res;
  feature_vector fixed_fps_low_res;

  const char* fixed_file_name_high_res = argv[1];
  const char* fixed_file_name_low_res = argv[2];
  const char* moving_file_name_high_res = argv[3];
  const char* moving_file_name_low_res = argv[4];

  read_feature_file( moving_file_name_high_res,
moving_fps_high_res );
  read_feature_file( moving_file_name_low_res,
moving_fps_low_res );
  read_feature_file( fixed_file_name_high_res,
fixed_fps_high_res );
  read_feature_file( fixed_file_name_low_res,
```

24

Example

```
// Prepare feature sets
//
rgrl_feature_set_sptr moving_feature_set_high_res
   = new
rgrl_feature_set_location<2>(moving_fps_high_res);
rgrl_feature_set_sptr moving_feature_set_low_res
   = new rgrl_feature_set_location<2>(moving_fps_low_res);
rgrl_feature_set_sptr fixed_feature_set_high_res
   = new rgrl_feature_set_location<2>(fixed_fps_high_res
rgrl_feature_set_sptr fixed_feature_set_low_res
   = new rgrl_feature_set_location<2>(fixed_fps_low_res);

// Set the initial transformation to be identity
//
int dof = 6;                        //dof for affine model
rgrl_transformation_sptr init_trans;
vnl_matrix <double> A(2,2,vnl_matrix_identity);
vector_2d          t(0.0, 0.0);
init_trans = new rgrl_trans_affine(A, t)
```

25

Example

```
// We first instantiate the 2 transformation models, and the
// initializer for the lower resolution. Please note, the
// initializer has to have the knowledge of the resolution level at
// which the registration begins. The higher the number is for the
// resolution level, the lower the image resolution. The highest
// resolution is always at the $0^{th}$ level.
//
rgrl_estimator_sptr affine_model = new rgrl_est_affine(6, 6);
rgrl_estimator_sptr quadratic_model = new rgrl_est_quadratic(12, 12);

vector_2d x0(0,0);
vector_2d x1(511, 511);
rgrl_roi moving_image_region(x0, x1);
rgrl_roi fixed_image_region(x0, x1);
int initial_resolution = 1;
rgrl_initializer_sptr initializer =
  new rgrl_initializer_prior(moving_image_region,
                             fixed_image_region,
                             affine_model,
                             init_trans,
                             initial_resolution );
```

26

Example

```
// Create matcher
//
unsigned int k = 1;
rgrl_matcher_sptr cp_matcher = new rgrl_matcher_k_nearest( k );

// Robust weighting
//
vcl_auto_ptr<rrel_m_est_obj>  m_est_obj( new rrel_tukey_obj(4) );
rgrl_weighter_sptr wgter = new rgrl_weighter_m_est(m_est_obj, false,


// Scale estimator both weighted and unweighted
//
int max_set_size = 3000;
vcl_auto_ptr<rrel_objective> muset_obj( new rrel_muset_obj( max_set_s

rgrl_scale_estimator_unwgted_sptr unwgted_scale_est;
rgrl_scale_estimator_wgted_sptr wgted_scale_est;
unwgted_scale_est = new rgrl_scale_est_closest( muset_obj );
wgted_scale_est = new rgrl_scale_est_all_weights();
```

27

Example

```
// Convergence tester
//
double tolerance = 1.5;
rgrl_convergence_tester_sptr conv_test =
  new rgrl_convergence_on_weighted_error( tolerance );
```

28

Example

```
// We add data and appropriate models to both stages the same way as
// we have done in the previous 2 examples. The only new step here
// is to specify the transition going from the lower to the higher
// resolution using
// \code{set\_dimension\_increase\_for\_next\_stage()} method. This
// method sets the parameter that allows the internal data to be
// scaled properly when moving to the next resolution level. In this
// example, since the lower resolution image is half the size for
// each dimension of the original, the \code{dimension\_increase} is
// set to 2 for the lower resolution.
//
bool multi_resol = true;
rgrl_data_manager_sptr data = new rgrl_data_manager( multi_resol );

int resolution = 0;              //original resolution
data->add_data( resolution,
                moving_feature_set_high_res,
                fixed_feature_set_high_res,
                cp_matcher,
                wgter,
                unwgted_scale_est, wgted_scale_est );
data->add_estimator( resolution, quadratic_model);
```

29

Example

```
resolution = 1;                  //lower resolution
double dimension_increase = 2;
data->add_data( resolution,
                moving_feature_set_low_res,
                fixed_feature_set_low_res,
                cp_matcher,
                wgter,
                unwgted_scale_est,
                wgted_scale_est );
data->add_estimator( resolution, affine_model);
data->set_dimension_increase_for_next_stage( resolution,
                                             dimension_increase);
```

30

Example

```
  // Start the registration process
  //
  rgrl_feature_based_registration reg( data, conv_test );
  reg.run( initializer );

  // Output Results
  // …

  return 0;
}
```

31

Multiple Feature Sets at One Stage

- We can also use simultaneous different types of features at a single stage
  - For example, vascular landmarks and trace points
- Each feature set must have a different feature type
- The `rgrl_data_manager` handles multiple feature sets automatically.
- The registration objects automatically detect the existence of multiple feature sets:
  - It requests them from the data manager
  - It applies matching to them individually
  - It passes all matches to the estimation objects

32

Feature Extraction

- This far we have discussed little about feature extraction
- Feature extraction is unnecessary for range data
- We have good algorithms for feature extraction in retinal images
- Algorithms in vxl and itk can also be adapted to extract features

33

The rest of semester

- Uncertainty-Driven registration
- Initialization
- Key-point matching
- Mutual Information
- Tracking
- Multi-sensor registration
- Mosaicking

34