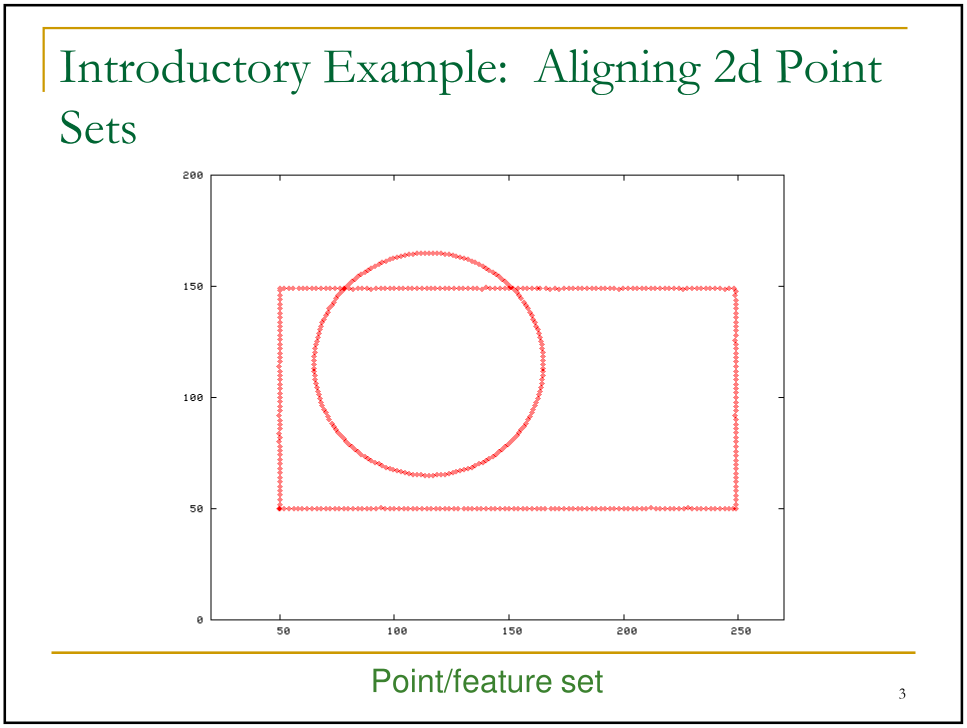
Image Registration
Lecture 12: RGRL

Prof. Charlene Tsai

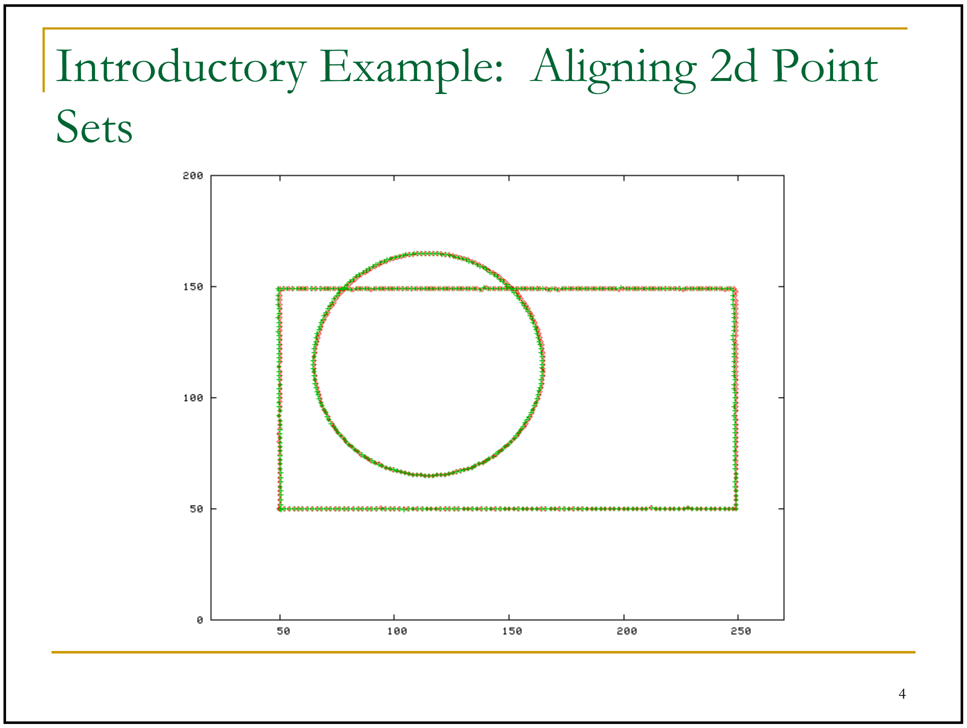
RGRL

- Rensselaer Generalized Registration Library
- Emphasizes feature-based registration
  - Really, correspondence-based registration
- Built on top of VXL
- Can interact with ITK
  - Please refer to the CMake lecture for details
- Doesn't include feature extraction!
  - Range image points will be the features
  - Use `vxl_src/core/vil/algo` to create features or use ITK algorithms.
- Work in progress!
  - Don't expect the completeness of ITK



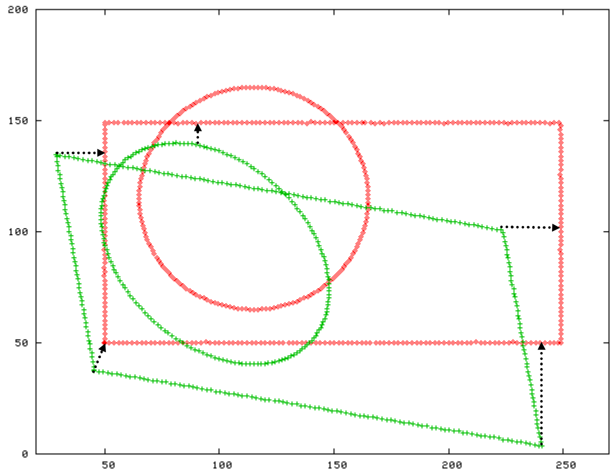
Introductory Example: Aligning 2d Point Sets

Point/feature set



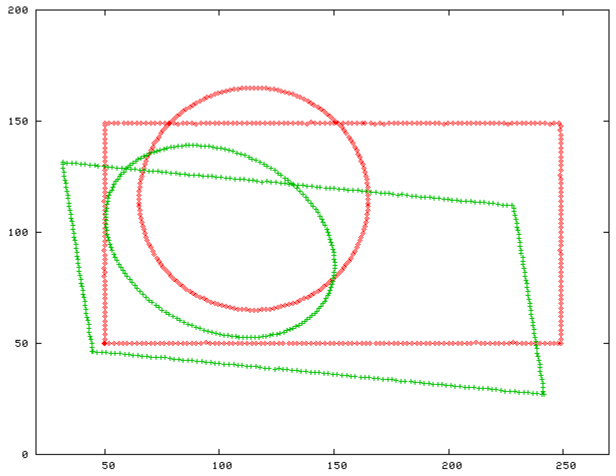
Introductory Example: Aligning 2d Point Sets

Introductory Example: Aligning 2d Point Sets

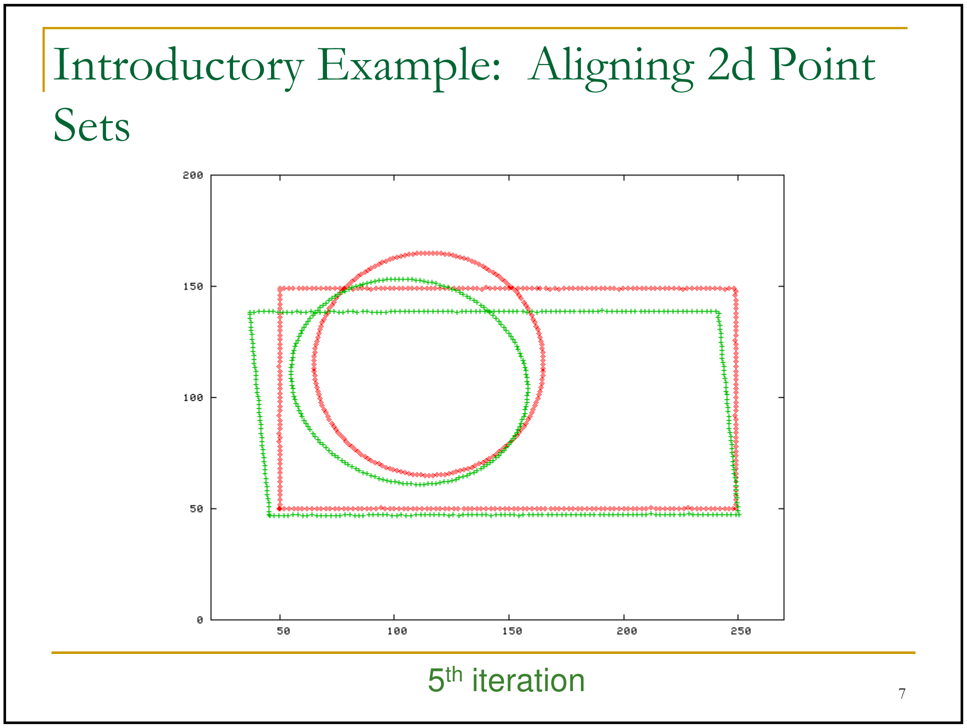


Initial affine transformation

Introductory Example: Aligning 2d Point Sets



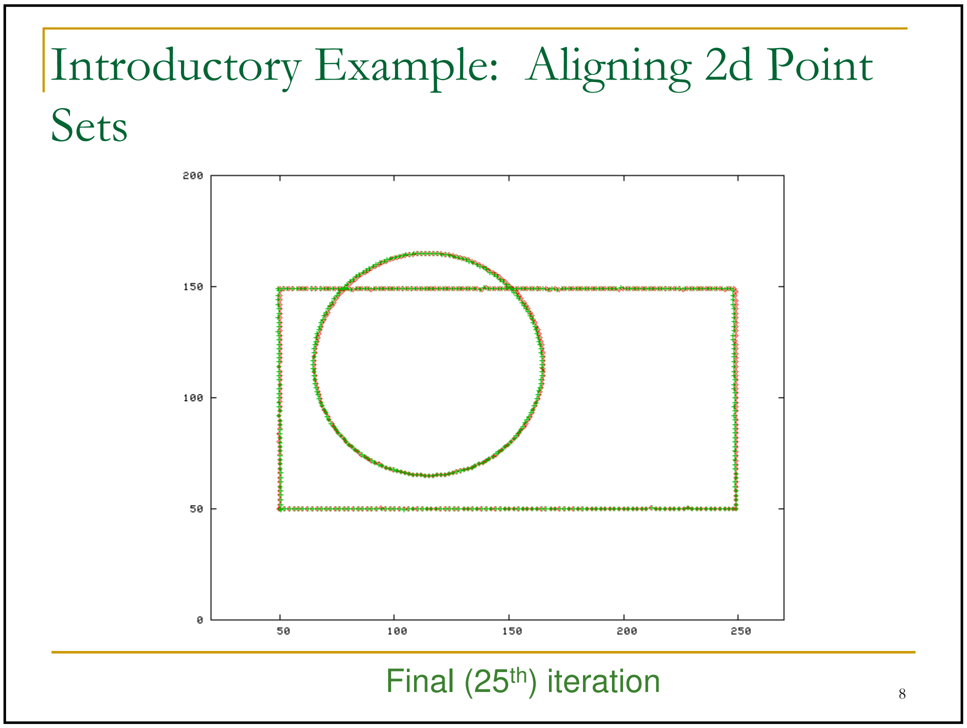
2nd iteration



Introductory Example: Aligning 2d Point Sets

5th iteration

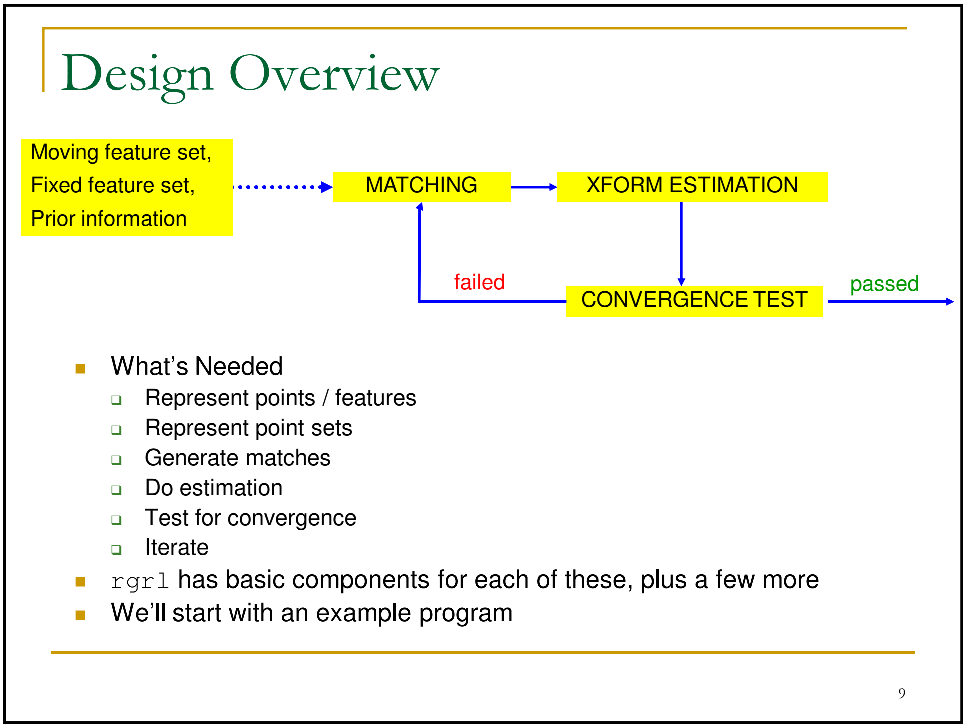
7



Introductory Example: Aligning 2d Point Sets

Final (25th) iteration

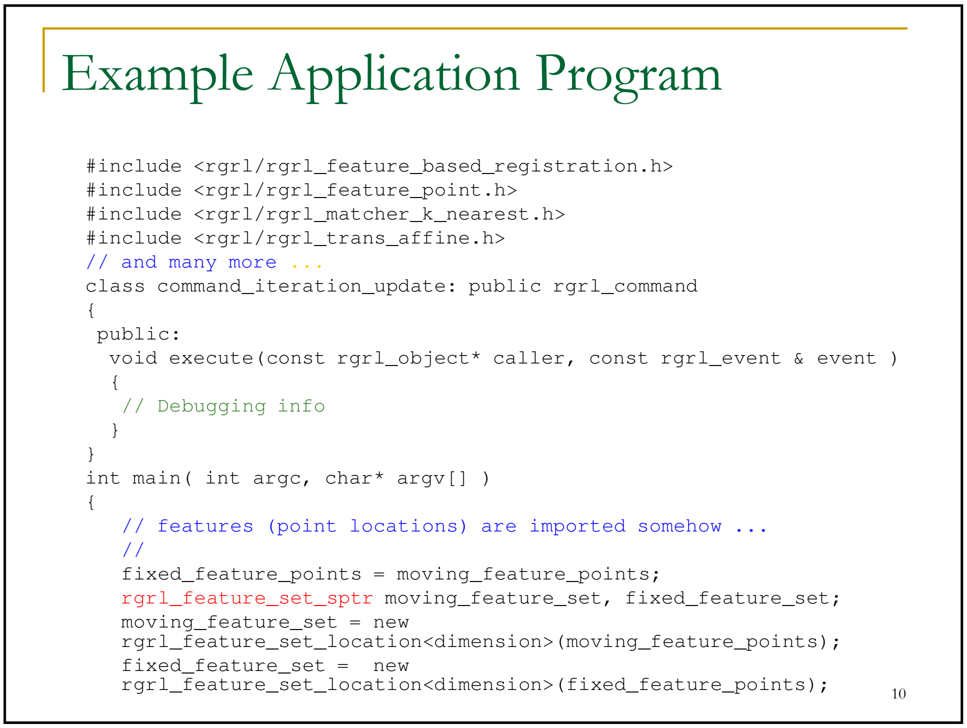
8



Design Overview

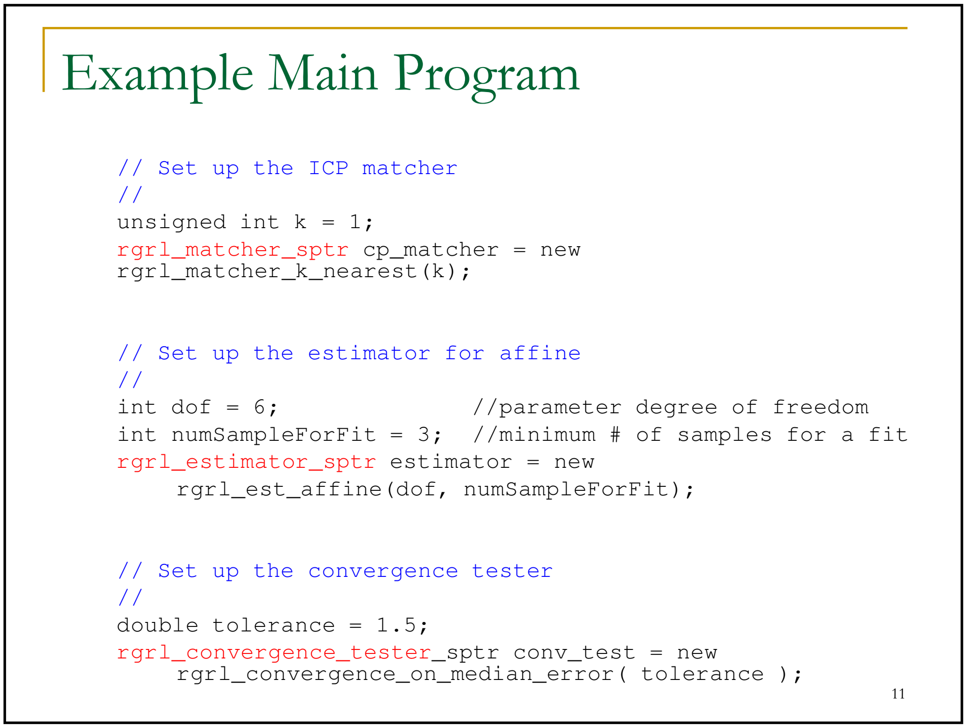
Moving feature set,
Fixed feature set,
Prior information

- What's Needed
  - Represent points / features
  - Represent point sets
  - Generate matches
  - Do estimation
  - Test for convergence
  - Iterate
- `rgrl` has basic components for each of these, plus a few more
- We'll start with an example program



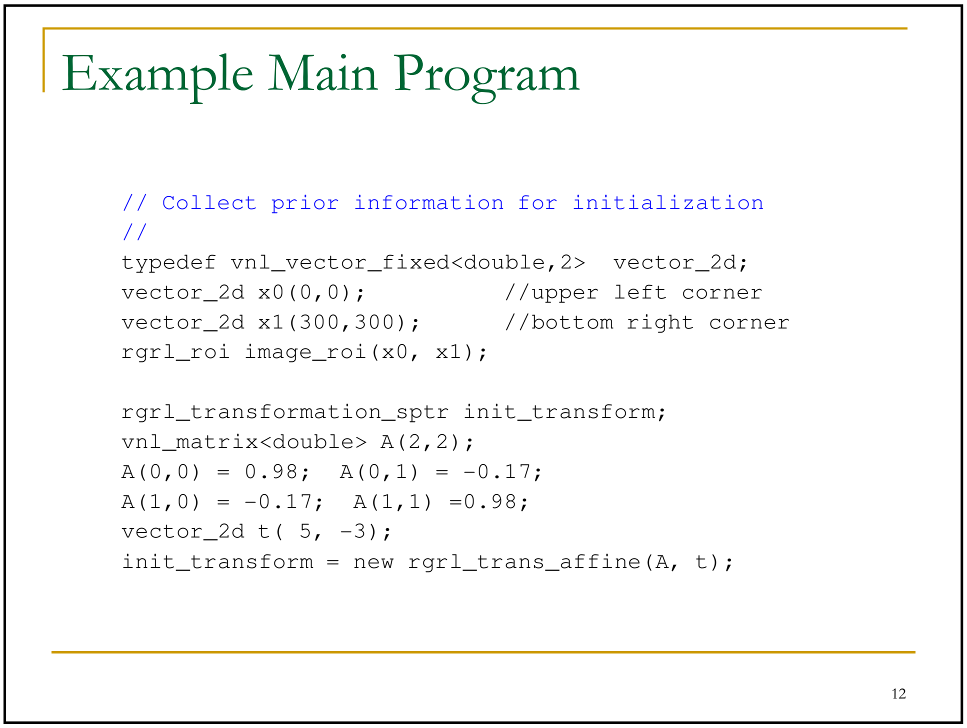
Example Application Program

```
#include <rgrl/rgrl_feature_based_registration.h>
#include <rgrl/rgrl_feature_point.h>
#include <rgrl/rgrl_matcher_k_nearest.h>
#include <rgrl/rgrl_trans_affine.h>
// and many more ...
class command_iteration_update: public rgrl_command
{
 public:
  void execute(const rgrl_object* caller, const rgrl_event & event )
  {
   // Debugging info
  }
}
int main( int argc, char* argv[] )
{
   // features (point locations) are imported somehow ...
   //
   fixed_feature_points = moving_feature_points;
   rgrl_feature_set_sptr moving_feature_set, fixed_feature_set;
   moving_feature_set = new
   rgrl_feature_set_location<dimension>(moving_feature_points);
   fixed_feature_set =  new
   rgrl_feature_set_location<dimension>(fixed_feature_points);
```



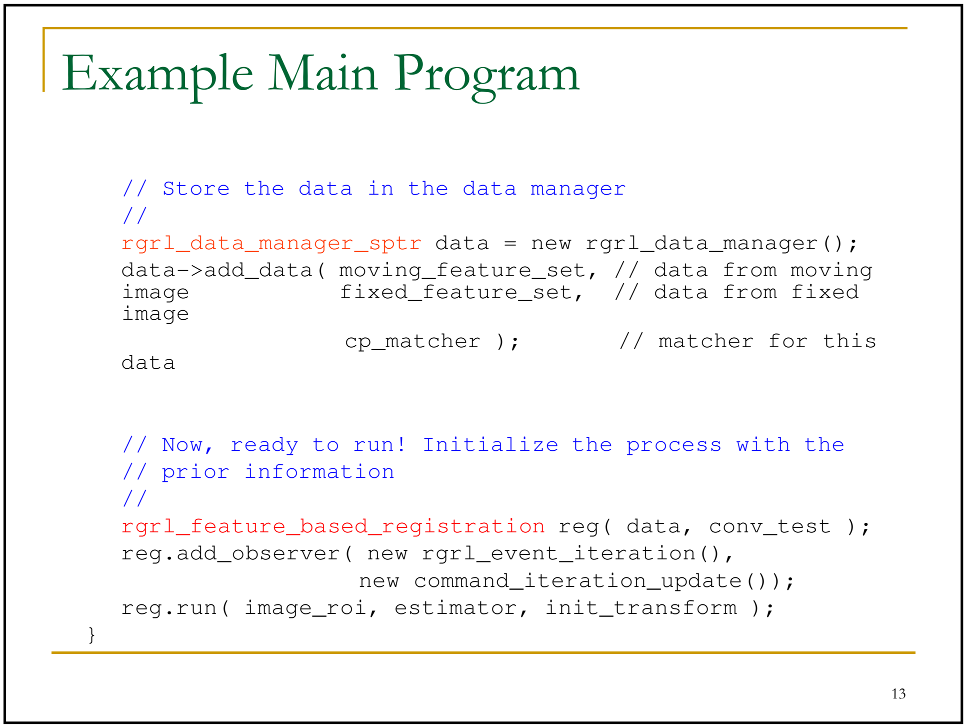
Example Main Program

```
// Set up the ICP matcher
//
unsigned int k = 1;
rgrl_matcher_sptr cp_matcher = new
rgrl_matcher_k_nearest(k);


// Set up the estimator for affine
//
int dof = 6;                 //parameter degree of freedom
int numSampleForFit = 3;  //minimum # of samples for a fit
rgrl_estimator_sptr estimator = new
    rgrl_est_affine(dof, numSampleForFit);


// Set up the convergence tester
//
double tolerance = 1.5;
rgrl_convergence_tester_sptr conv_test = new
    rgrl_convergence_on_median_error( tolerance );
```



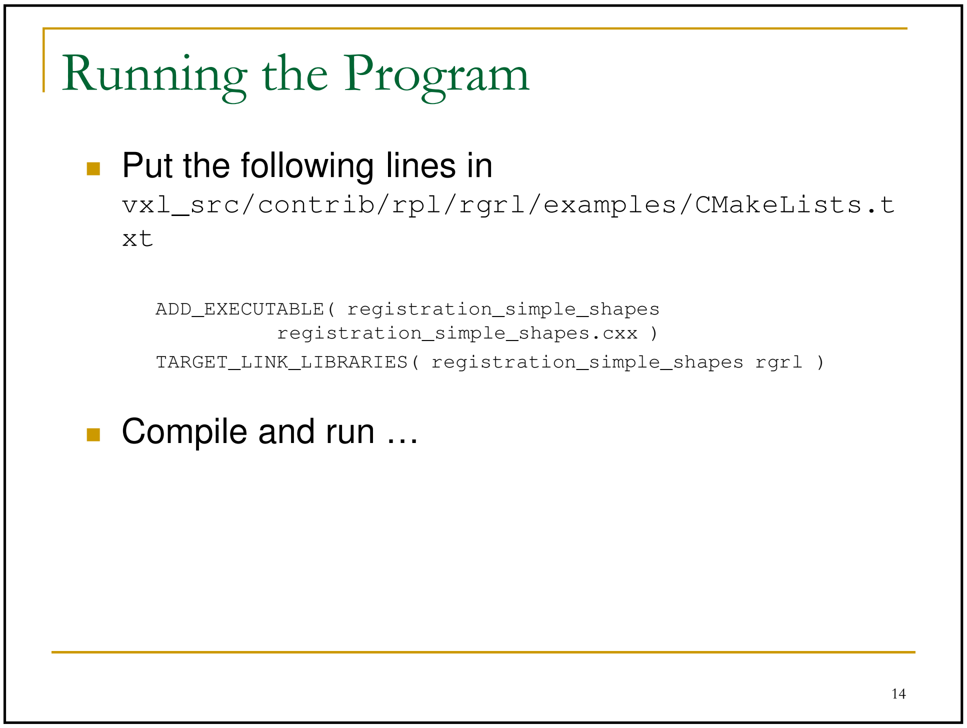
Example Main Program

```
// Collect prior information for initialization
//
typedef vnl_vector_fixed<double,2>  vector_2d;
vector_2d x0(0,0);          //upper left corner
vector_2d x1(300,300);      //bottom right corner
rgrl_roi image_roi(x0, x1);

rgrl_transformation_sptr init_transform;
vnl_matrix<double> A(2,2);
A(0,0) = 0.98;  A(0,1) = -0.17;
A(1,0) = -0.17;  A(1,1) =0.98;
vector_2d t( 5, -3);
init_transform = new rgrl_trans_affine(A, t);
```



Example Main Program

```
    // Store the data in the data manager
    //
    rgrl_data_manager_sptr data = new rgrl_data_manager();
    data->add_data( moving_feature_set, // data from moving
    image           fixed_feature_set,  // data from fixed
    image
                     cp_matcher );       // matcher for this
    data


    // Now, ready to run! Initialize the process with the
    // prior information
    //
    rgrl_feature_based_registration reg( data, conv_test );
    reg.add_observer( new rgrl_event_iteration(),
                      new command_iteration_update());
    reg.run( image_roi, estimator, init_transform );
  }
```

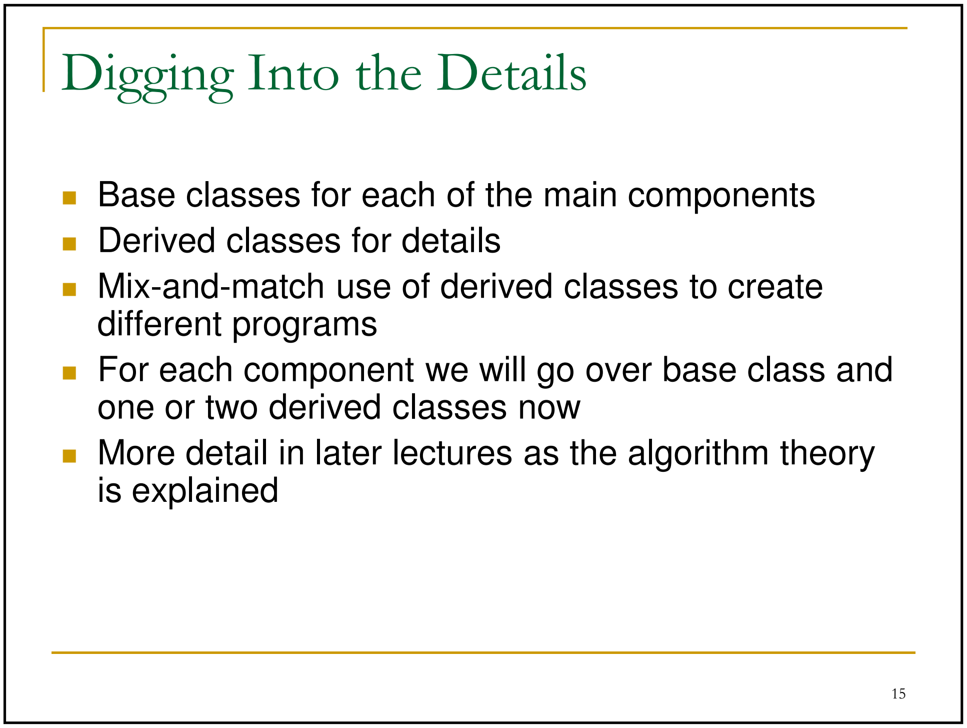


Running the Program

- Put the following lines in `vxl_src/contrib/rpl/rgrl/examples/CMakeLists.txt`

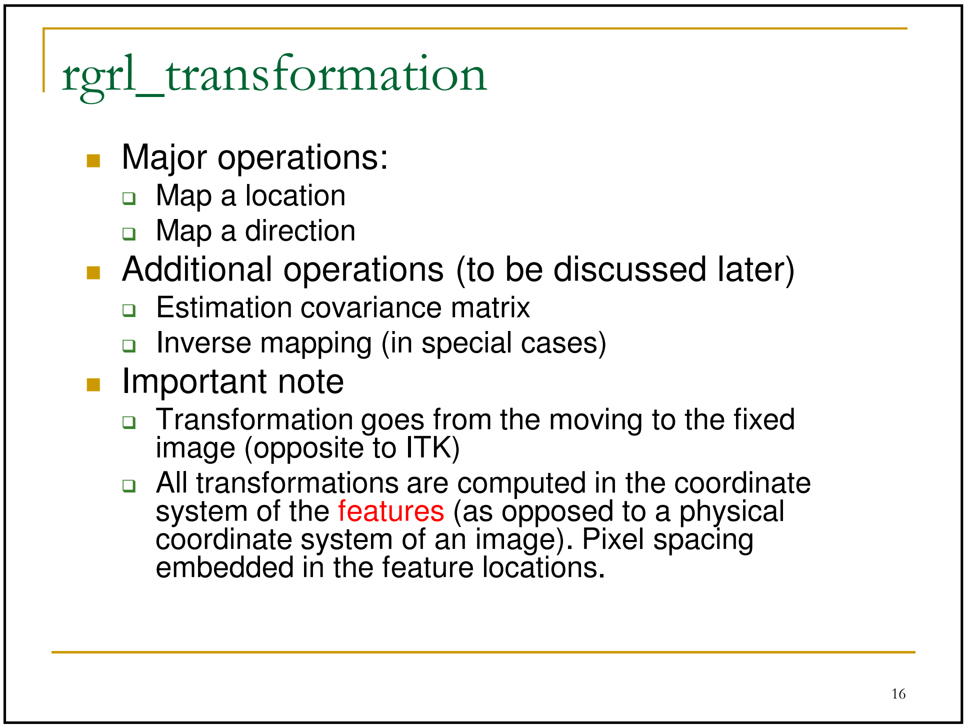
```
ADD_EXECUTABLE( registration_simple_shapes
          registration_simple_shapes.cxx )
TARGET_LINK_LIBRARIES( registration_simple_shapes rgrl )
```

- Compile and run …



Digging Into the Details

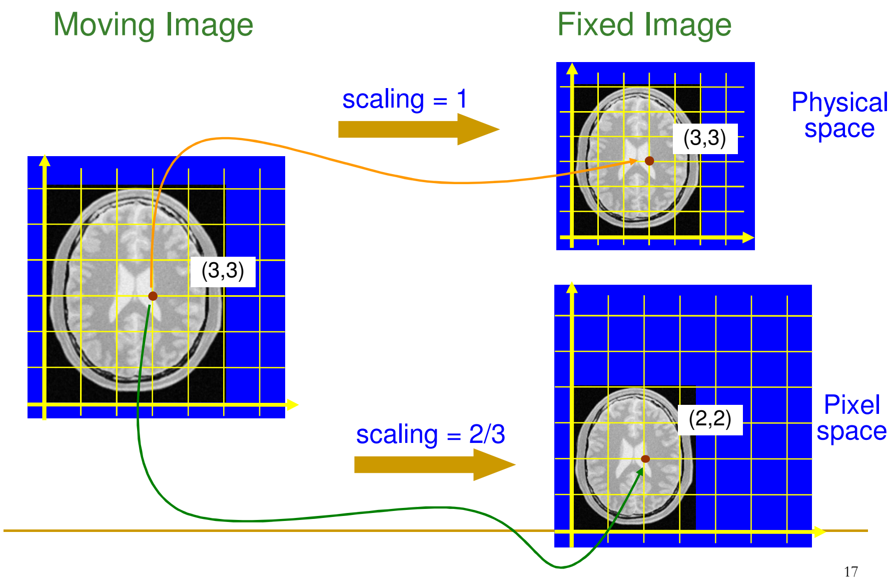
- Base classes for each of the main components
- Derived classes for details
- Mix-and-match use of derived classes to create different programs
- For each component we will go over base class and one or two derived classes now
- More detail in later lectures as the algorithm theory is explained



rgrl_transformation

- Major operations:
  - Map a location
  - Map a direction
- Additional operations (to be discussed later)
  - Estimation covariance matrix
  - Inverse mapping (in special cases)
- Important note
  - Transformation goes from the moving to the fixed image (opposite to ITK)
  - All transformations are computed in the coordinate system of the features (as opposed to a physical coordinate system of an image). Pixel spacing embedded in the feature locations.

Quiz #1



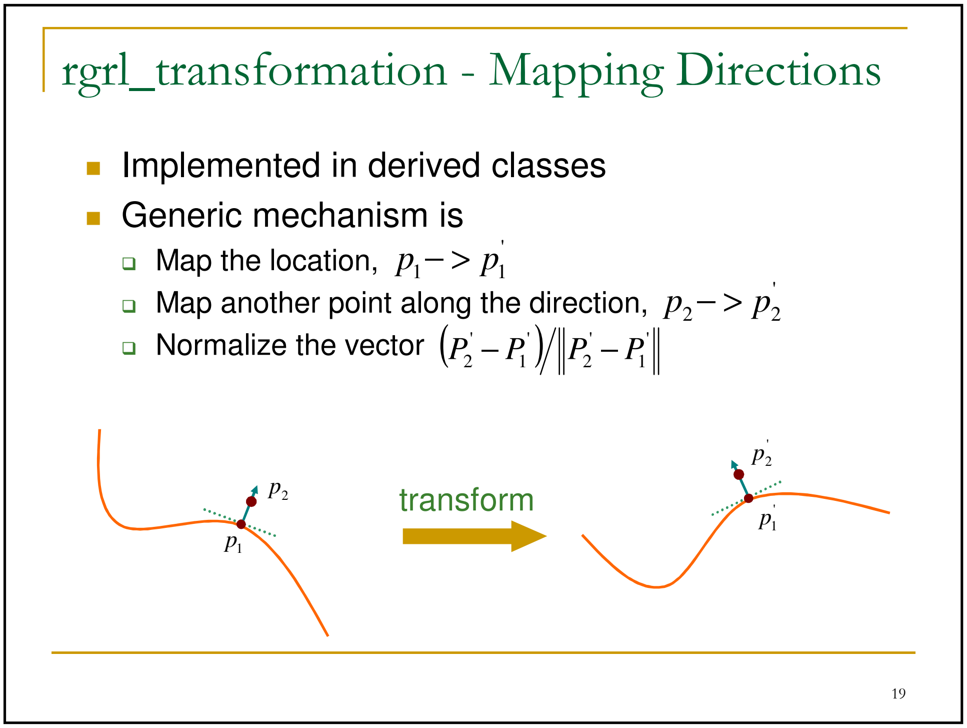
A Bit of the .h file

- Map location:

```
void map_location( vnl_vector<double> const& from,
                     vnl_vector<double>      & to )
const;
```

- Map direction:

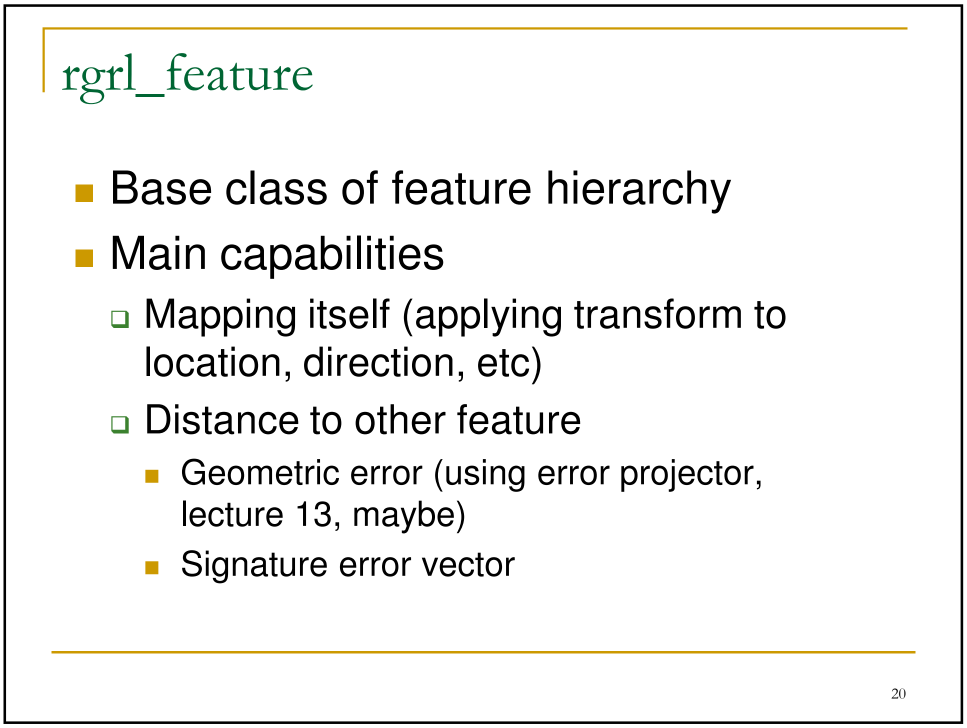
```
void map_direction(vnl_vector<double> const& from_loc,
                     vnl_vector<double>const& from_dir,
                     vnl_vector<double> & to_dir)
const;
```



rgrl_transformation - Mapping Directions

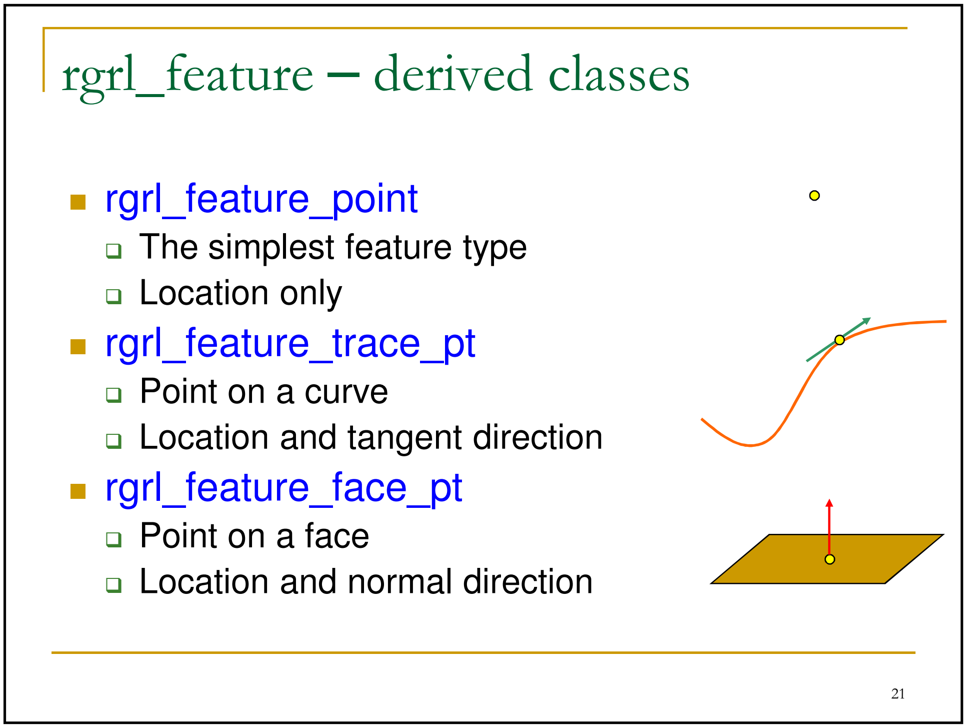
- Implemented in derived classes
- Generic mechanism is
  - Map the location, $p_1 -> p_1'$
  - Map another point along the direction, $p_2 -> p_2'$
  - Normalize the vector $\left(P_2' - P_1'\right) / \left\| P_2' - P_1' \right\|$

p_2
p_1
transform
p_2'
p_1'



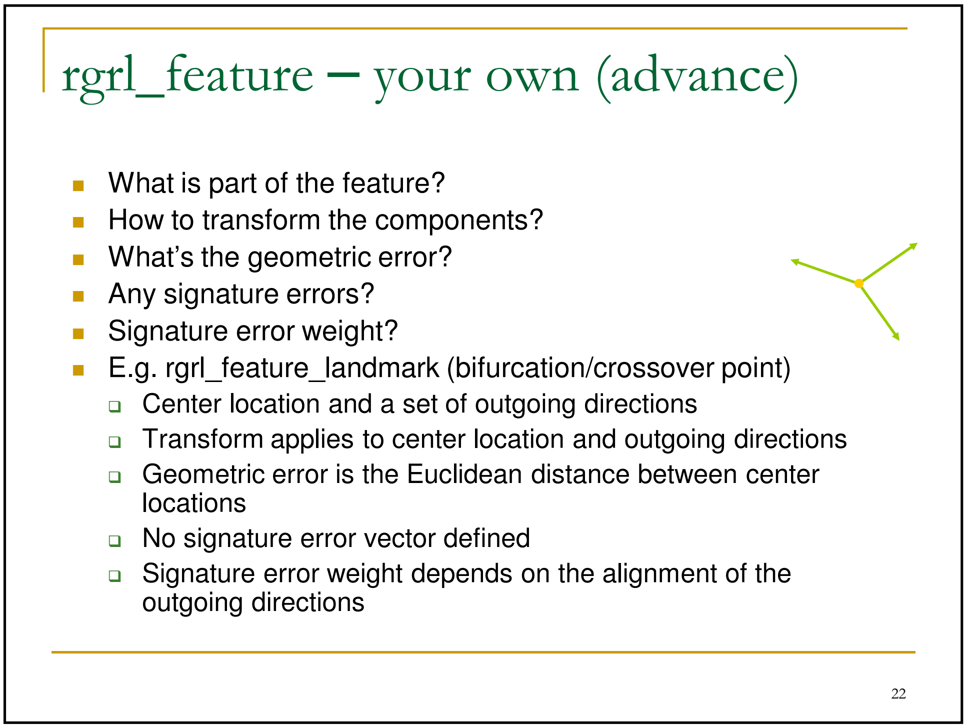
rgrl_feature

- Base class of feature hierarchy
- Main capabilities
  - Mapping itself (applying transform to location, direction, etc)
  - Distance to other feature
    - Geometric error (using error projector, lecture 13, maybe)
    - Signature error vector



rgrl_feature – derived classes

- rgrl_feature_point
  - The simplest feature type
  - Location only
- rgrl_feature_trace_pt
  - Point on a curve
  - Location and tangent direction
- rgrl_feature_face_pt
  - Point on a face
  - Location and normal direction



rgrl_feature – your own (advance)

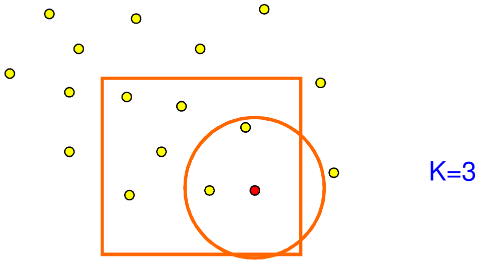
- What is part of the feature?
- How to transform the components?
- What's the geometric error?
- Any signature errors?
- Signature error weight?
- E.g. rgrl_feature_landmark (bifurcation/crossover point)
  - Center location and a set of outgoing directions
  - Transform applies to center location and outgoing directions
  - Geometric error is the Euclidean distance between center locations
  - No signature error vector defined
  - Signature error weight depends on the alignment of the outgoing directions

rgrl_feature_set

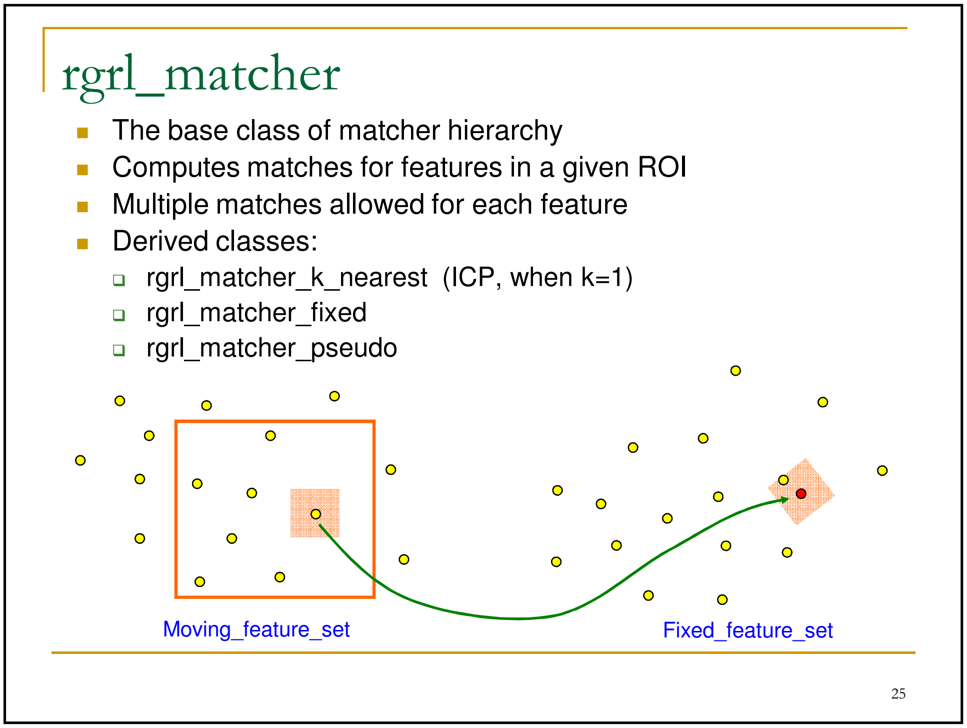
- Base class of feature set hierarchy
- Collects features of
  - the same type
  - the same resolution
- Provides methods to access the set of registration features
- All methods defined in the derived classes
- What are the methods?

rgrl_feature_set_location

- A set of features grouped by N-d location
- Proximity is determined only by location



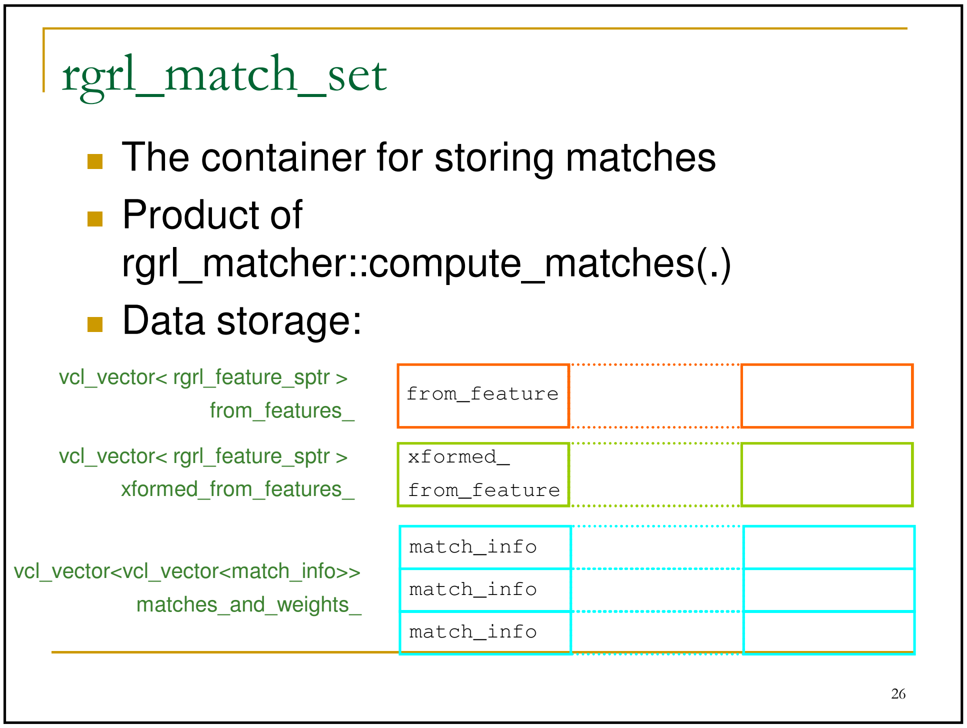
```
feature_vector
features_within_distance( rgrl_feature_sptr feature, double distance )
feature_vector
k_nearest_features( rgrl_feature_sptr feature, unsigned int k )
```



rgrl_matcher

- The base class of matcher hierarchy
- Computes matches for features in a given ROI
- Multiple matches allowed for each feature
- Derived classes:
  - rgrl_matcher_k_nearest (ICP, when k=1)
  - rgrl_matcher_fixed
  - rgrl_matcher_pseudo

Moving_feature_set
Fixed_feature_set



rgrl_match_set

- The container for storing matches
- Product of rgrl_matcher::compute_matches(.)
- Data storage:

vcl_vector< rgrl_feature_sptr > from_features_

vcl_vector< rgrl_feature_sptr > xformed_from_features_

vcl_vector<vcl_vector<match_info>> matches_and_weights_

from_feature
xformed_
from_feature
match_info
match_info
match_info

rgrl_match_set

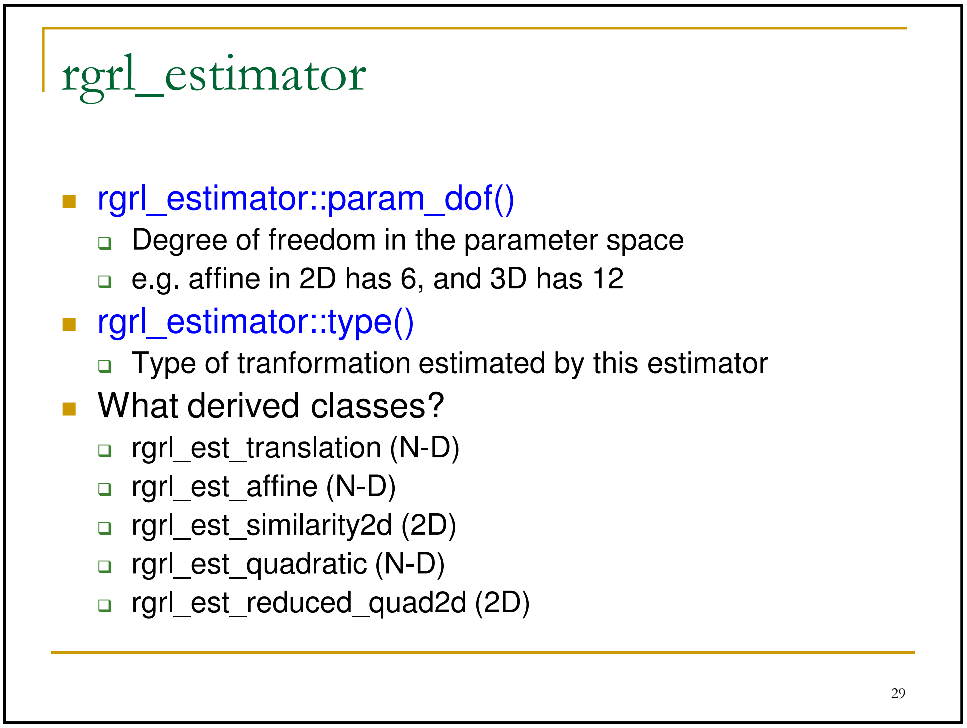
- What's in rgrl_match_set::match_info?
  - rgrl_feature_sptr to_feature;
  - double geometric_weight;
  - double signature_weight;
  - double cumulative_weight;
- To update rgrl_match_set:

```
void add_feature_and_match(
     rgrl_feature_sptr  from_feature,
     rgrl_feature_sptr  matching_to,
     double             wgt = 1.0 );

void add_feature_and_matches(
     rgrl_feature_sptr  from_feature,
     vcl_vector< rgrl_feature_sptr > const& matching_to );

void remap_from_features( rgrl_transformation const&
trans );
```

rgrl_estimator

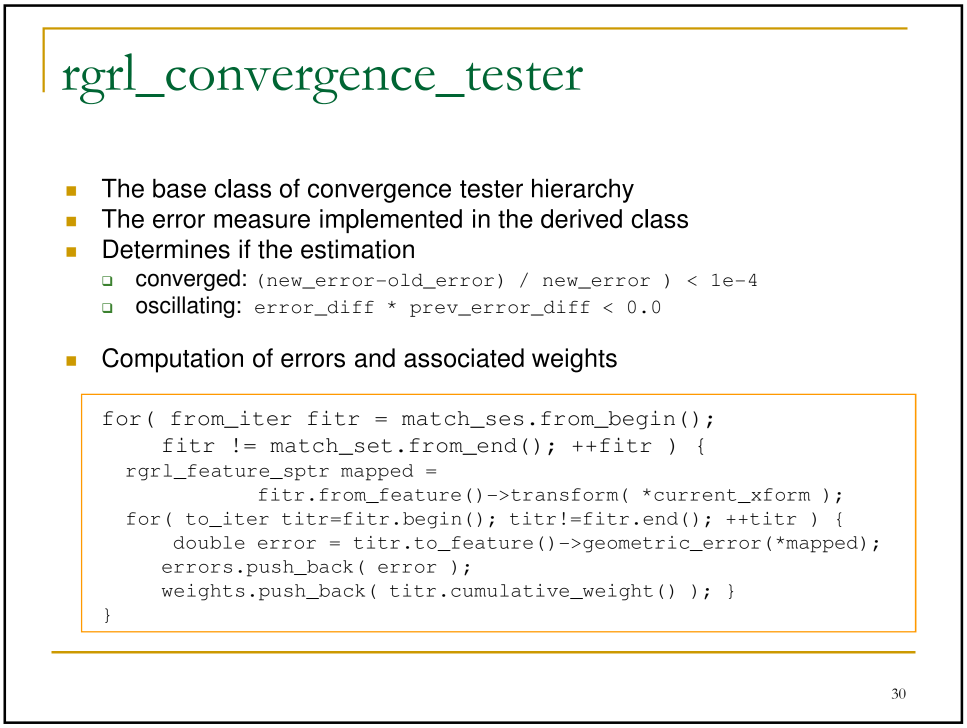
- The base class of estimator hierarchy
- A set of derived classes for different transformation types
- rgrl_estimator::estimate(.)
  - Compute the transform of a given match set
  - A virtual function
  - The derived class implements the details

```
virtual rgrl_transformation_sptr
estimate( rgrl_set_of<rgrl_match_set_sptr> const& match_sets,
          rgrl_transformation const& cur_transform ) const = 0;

virtual rgrl_transformation_sptr
estimate( rgrl_match_set_sptr const& match_set,
          rgrl_transformation const& cur_transform ) const = 0;
```



rgrl_estimator

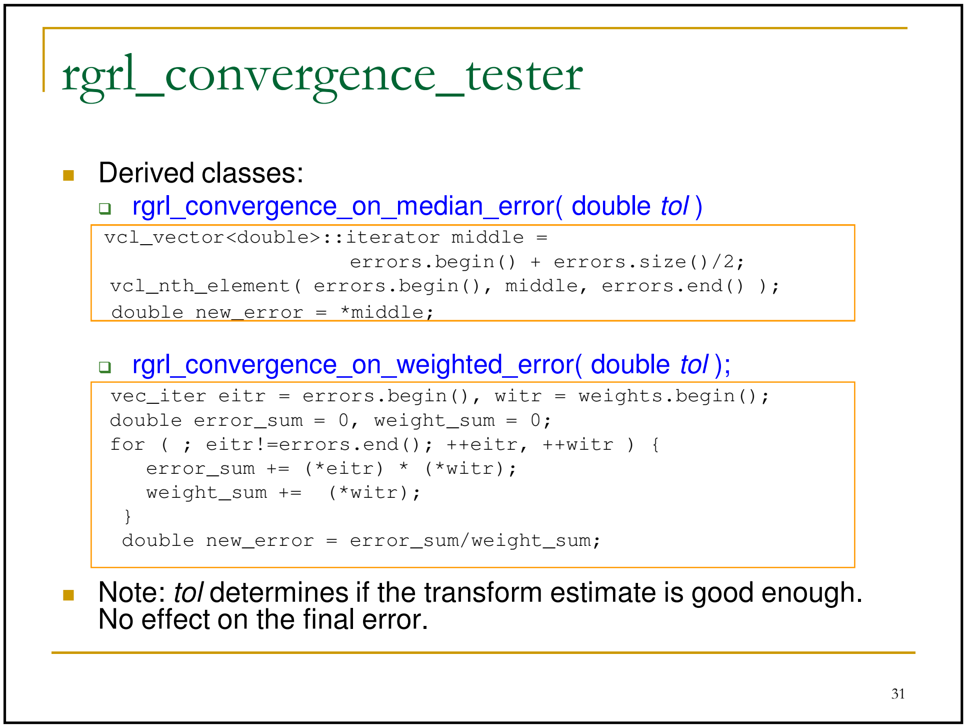
- rgrl_estimator::param_dof()
  - Degree of freedom in the parameter space
  - e.g. affine in 2D has 6, and 3D has 12
- rgrl_estimator::type()
  - Type of tranformation estimated by this estimator
- What derived classes?
  - rgrl_est_translation (N-D)
  - rgrl_est_affine (N-D)
  - rgrl_est_similarity2d (2D)
  - rgrl_est_quadratic (N-D)
  - rgrl_est_reduced_quad2d (2D)



rgrl_convergence_tester

- The base class of convergence tester hierarchy
- The error measure implemented in the derived class
- Determines if the estimation
  - converged: `(new_error-old_error) / new_error ) < 1e-4`
  - oscillating: `error_diff * prev_error_diff < 0.0`
- Computation of errors and associated weights

```
for( from_iter fitr = match_ses.from_begin();
     fitr != match_set.from_end(); ++fitr ) {
  rgrl_feature_sptr mapped =
            fitr.from_feature()->transform( *current_xform );
  for( to_iter titr=fitr.begin(); titr!=fitr.end(); ++titr ) {
      double error = titr.to_feature()->geometric_error(*mapped);
     errors.push_back( error );
     weights.push_back( titr.cumulative_weight() ); }
}
```



rgrl_convergence_tester

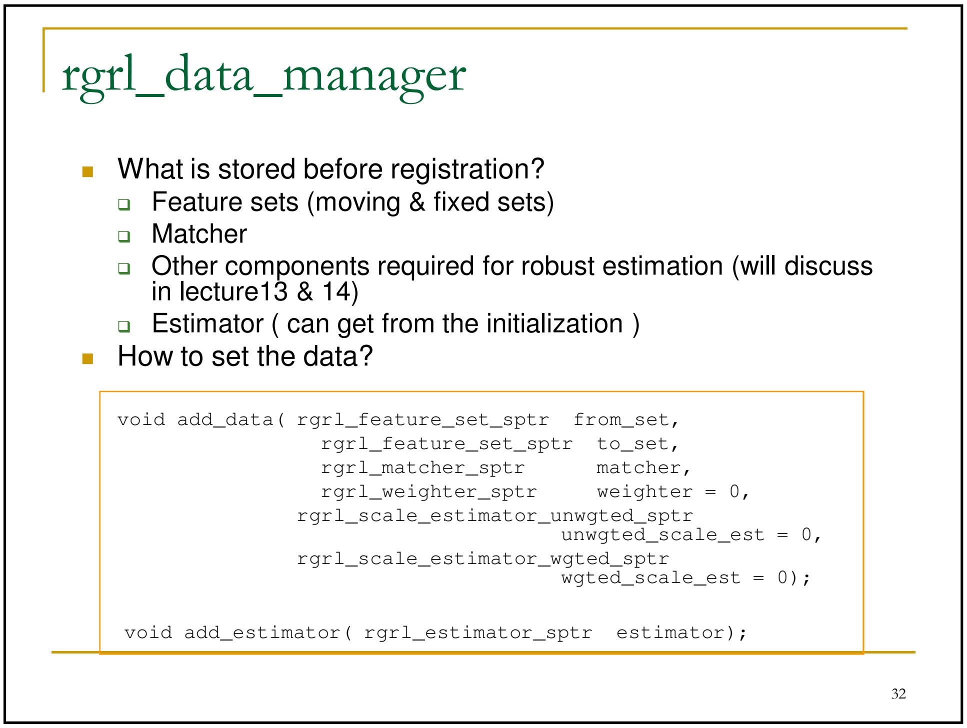
- Derived classes:
  - rgrl_convergence_on_median_error(double *tol*)

```
vcl_vector<double>::iterator middle =
                     errors.begin() + errors.size()/2;
vcl_nth_element( errors.begin(), middle, errors.end() );
double new_error = *middle;
```

  - rgrl_convergence_on_weighted_error(double *tol*);

```
vec_iter eitr = errors.begin(), witr = weights.begin();
double error_sum = 0, weight_sum = 0;
for ( ; eitr!=errors.end(); ++eitr, ++witr ) {
   error_sum += (*eitr) * (*witr);
   weight_sum +=  (*witr);
 }
double new_error = error_sum/weight_sum;
```

- Note: *tol* determines if the transform estimate is good enough. No effect on the final error.



rgrl_data_manager

- What is stored before registration?
  - Feature sets (moving & fixed sets)
  - Matcher
  - Other components required for robust estimation (will discuss in lecture13 & 14)
  - Estimator (can get from the initialization)
- How to set the data?

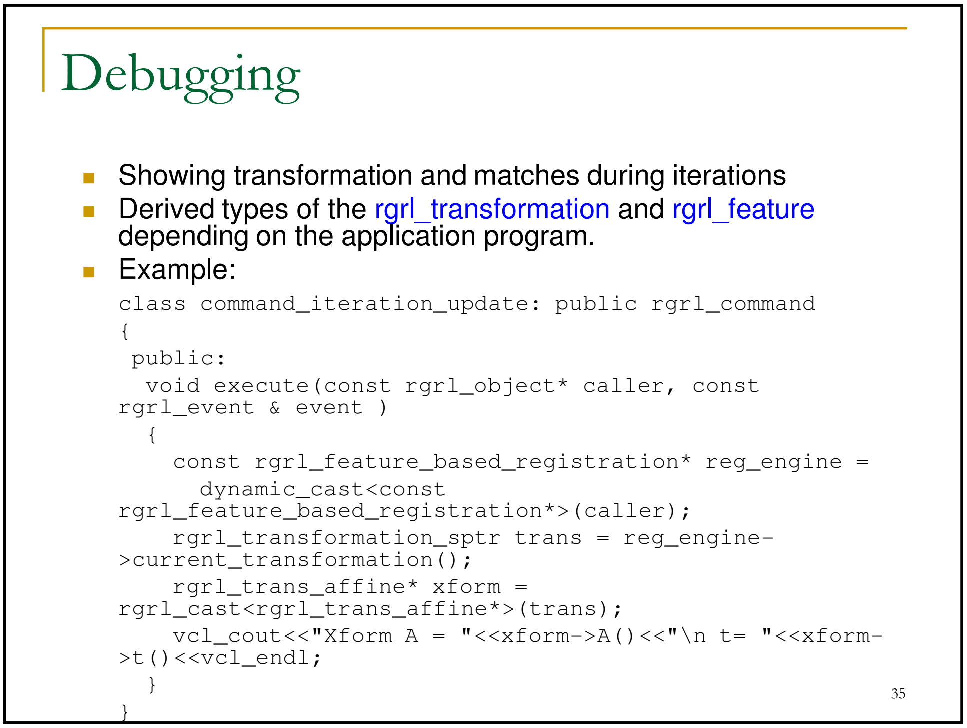
```
void add_data( rgrl_feature_set_sptr  from_set,
                 rgrl_feature_set_sptr  to_set,
                 rgrl_matcher_sptr      matcher,
                 rgrl_weighter_sptr     weighter = 0,
               rgrl_scale_estimator_unwgted_sptr
                                      unwgted_scale_est = 0,
               rgrl_scale_estimator_wgted_sptr
                                      wgted_scale_est = 0);


 void add_estimator( rgrl_estimator_sptr  estimator);
```

rgrl_data_manager – advance features

- Data can be stored in multiple stages
- Each stage can store multiple types of feature set, and multiple estimators
- What determines a “stage”?
  - All feature sets in one stage together estimate a transformation
- Need to specify relation between stages in terms of resolution
- You don’t need this yet!

Quiz #2

Q. How to register two images using multi-resolution in two levels?

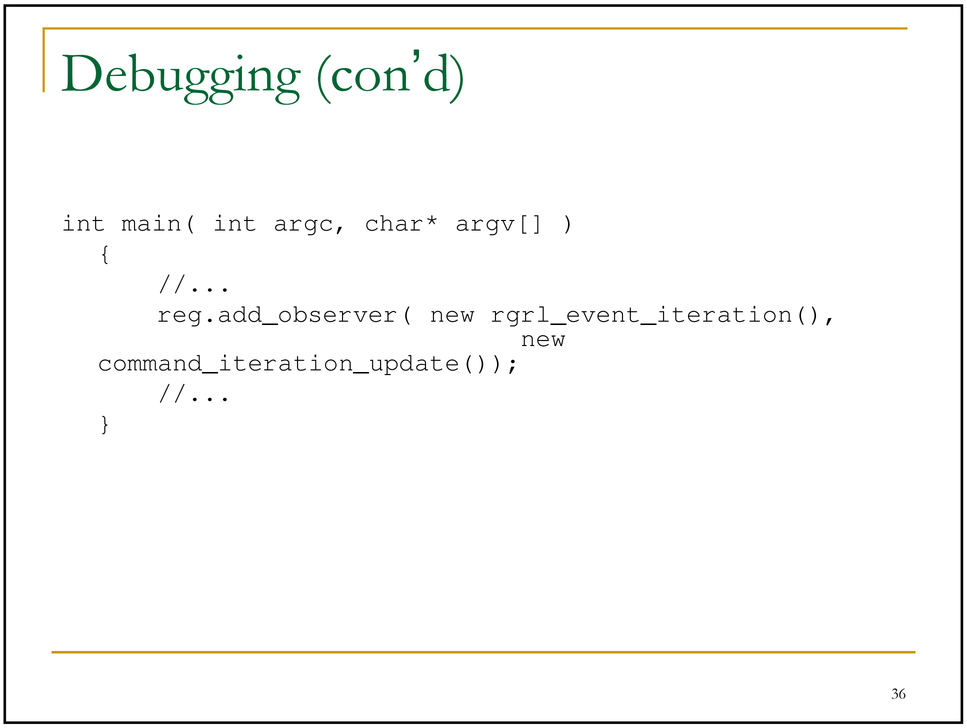


Debugging

- Showing transformation and matches during iterations
- Derived types of the rgrl_transformation and rgrl_feature depending on the application program.
- Example:

```
class command_iteration_update: public rgrl_command
{
 public:
  void execute(const rgrl_object* caller, const
rgrl_event & event )
  {
    const rgrl_feature_based_registration* reg_engine =
      dynamic_cast<const
rgrl_feature_based_registration*>(caller);
    rgrl_transformation_sptr trans = reg_engine-
>current_transformation();
    rgrl_trans_affine* xform =
rgrl_cast<rgrl_trans_affine*>(trans);
    vcl_cout<<"Xform A = "<<xform->A()<<"\n t= "<<xform-
>t()<<vcl_endl;
  }
}
```

35



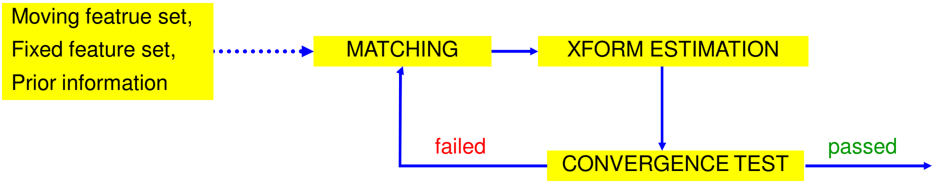
Debugging (con'd)

```
int main( int argc, char* argv[] )
   {
       //...
       reg.add_observer( new rgrl_event_iteration(),
                                              new
   command_iteration_update());
       //...
   }
```

36

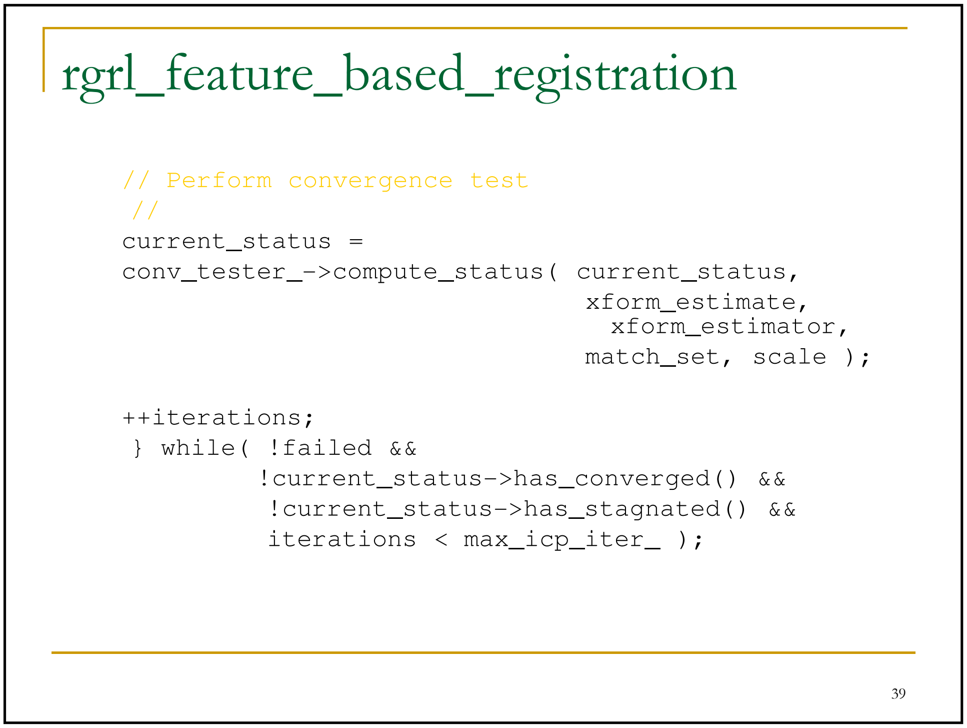
rgrl_feature_based_registration

- The registration engine
- Put everything together (one stage, one data item, one estimator)



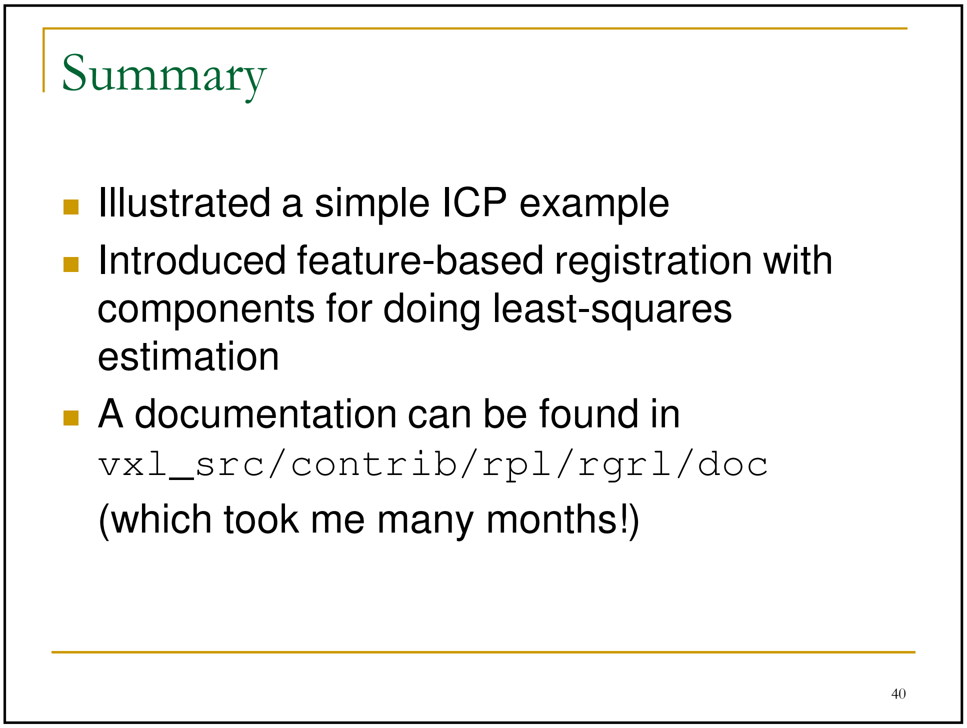
rgrl_feature_based_registration

```
unsigned iterations = 0; //total iteration
rgrl_converge_status_sptr current_status = 0;
bool failed = false;

do {
// Compute matches, and scales for each feature set.
//
match_set= matcher->compute_matches( *from_set, *to_set,
                                          *xform_estimate,
                                     image_region, *scale );

// Transformation estimation
//
double alignment_error;
if ( !rgrl_util_irls( match_set, scale, weighter,
                            *conv_tester_, xform_estimate,
                            xform_estimate,
alignment_error) ) {
    failed = true;
     continue; //no valid xform, so exit the loop
   }
```



rgrl_feature_based_registration

```
// Perform convergence test
 //
current_status =
conv_tester_->compute_status( current_status,
                               xform_estimate,
                                 xform_estimator,
                               match_set, scale );

++iterations;
 } while( !failed &&
         !current_status->has_converged() &&
          !current_status->has_stagnated() &&
          iterations < max_icp_iter_ );
```



Summary

- Illustrated a simple ICP example
- Introduced feature-based registration with components for doing least-squares estimation
- A documentation can be found in `vxl_src/contrib/rpl/rgrl/doc` (which took me many months!)