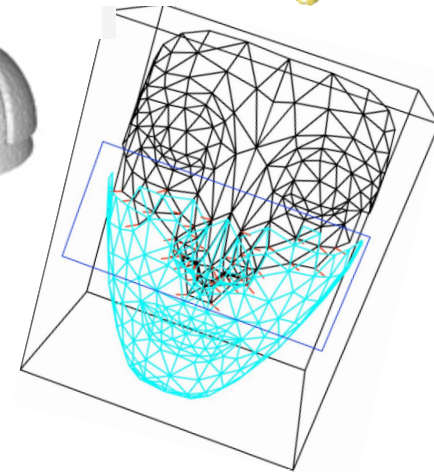
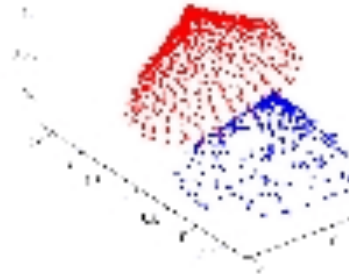
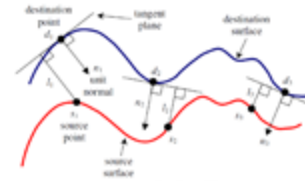


# A Tutorial on Rigid Registration

## Iterative Closed Point (ICP)



By  
Shireen Elhabian,  
Amal Farag, Aly Farag

University of Louisville, CVIP Lab  
March 2009

# Outline



- The problem
- Motivation
- Data types
- Mathematical preliminaries
  - Soma statistics: centroid, variance and covariance
  - Inner product
  - Transformation matrices
  - Eigenvalues problem
  - Mean square error
  - Rigid transformations (scaling, rotation and translation in 2D and 3D)
  - Quaternions

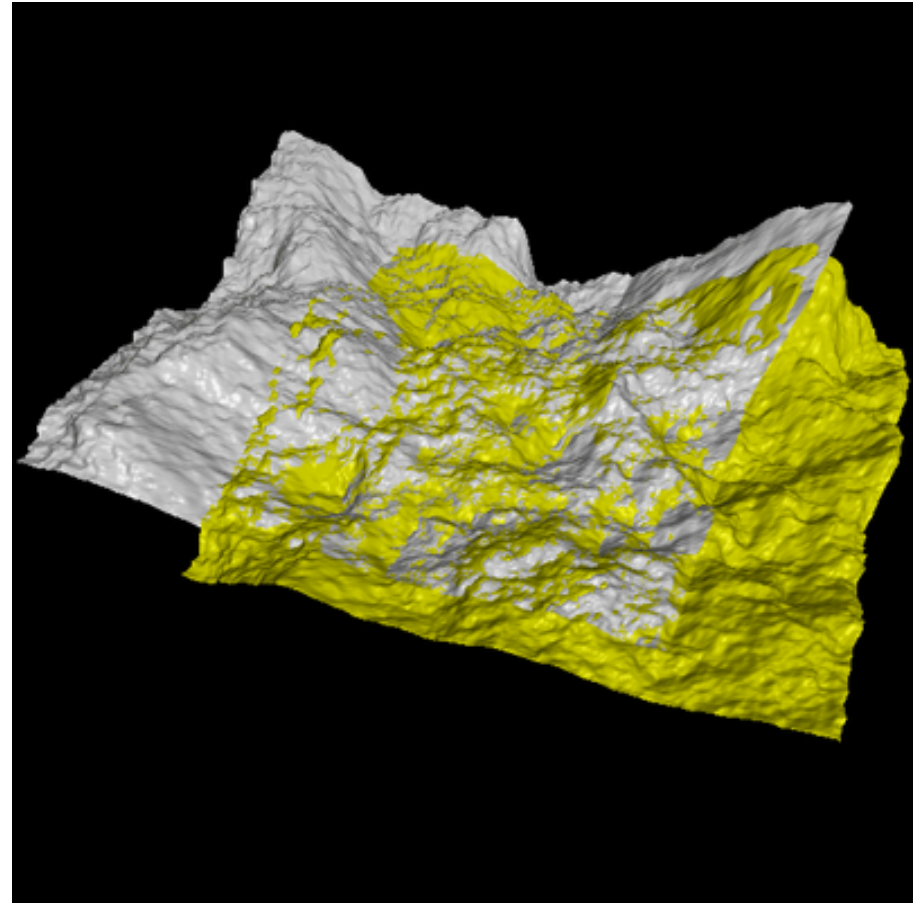
# Outline

- Iterative closest point algorithm
  - Problem statement
  - Main idea of ICP
  - Algorithm outline
  - Nomenclature
  - Finding correspondences
  - Alignment calculation
    - Finding the translational offset
    - Finding the scaling factor
    - Finding the best rotation
  - Let's do it ...
  - Experiments

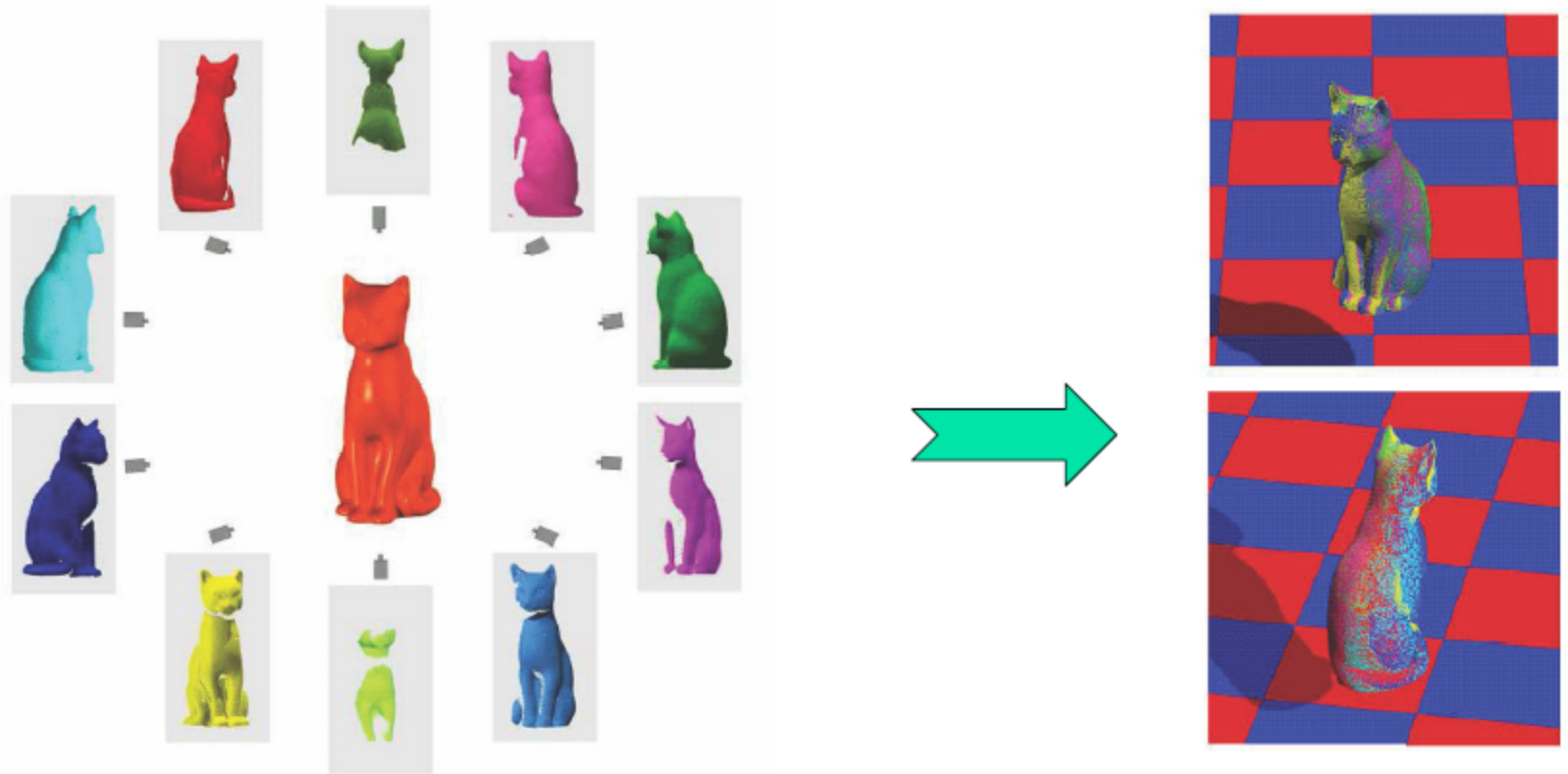


# The Problem

Align two partially-overlapping meshes given initial guess for relative transform



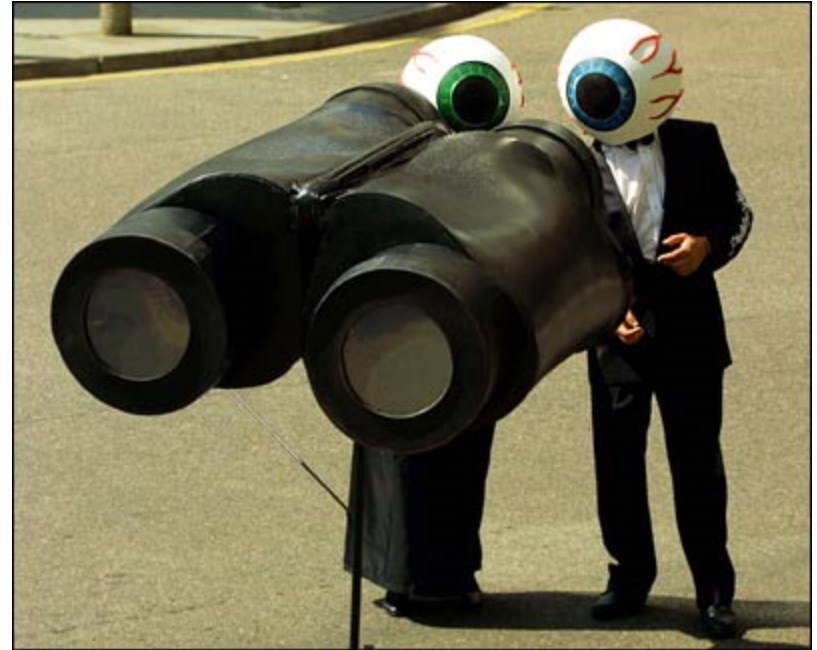
# The Problem



Images from: "Geometry and convergence analysis of algorithms for registration of 3D shapes" by Pottman

# Motivation

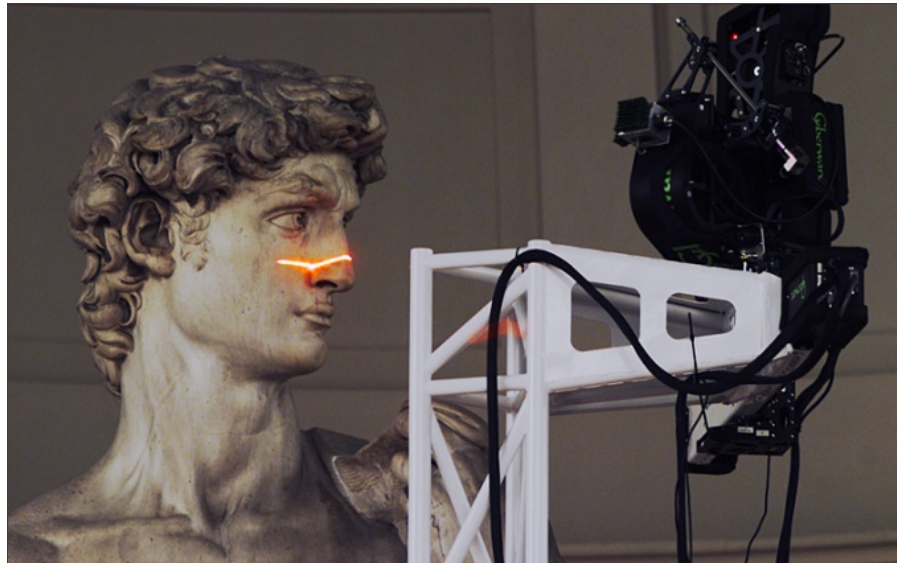
- Shape inspection
- Motion estimation
- Appearance analysis
- Texture Mapping
- Tracking



<http://blog.wired.com/defense/2007/08/danger-room-inb.html>

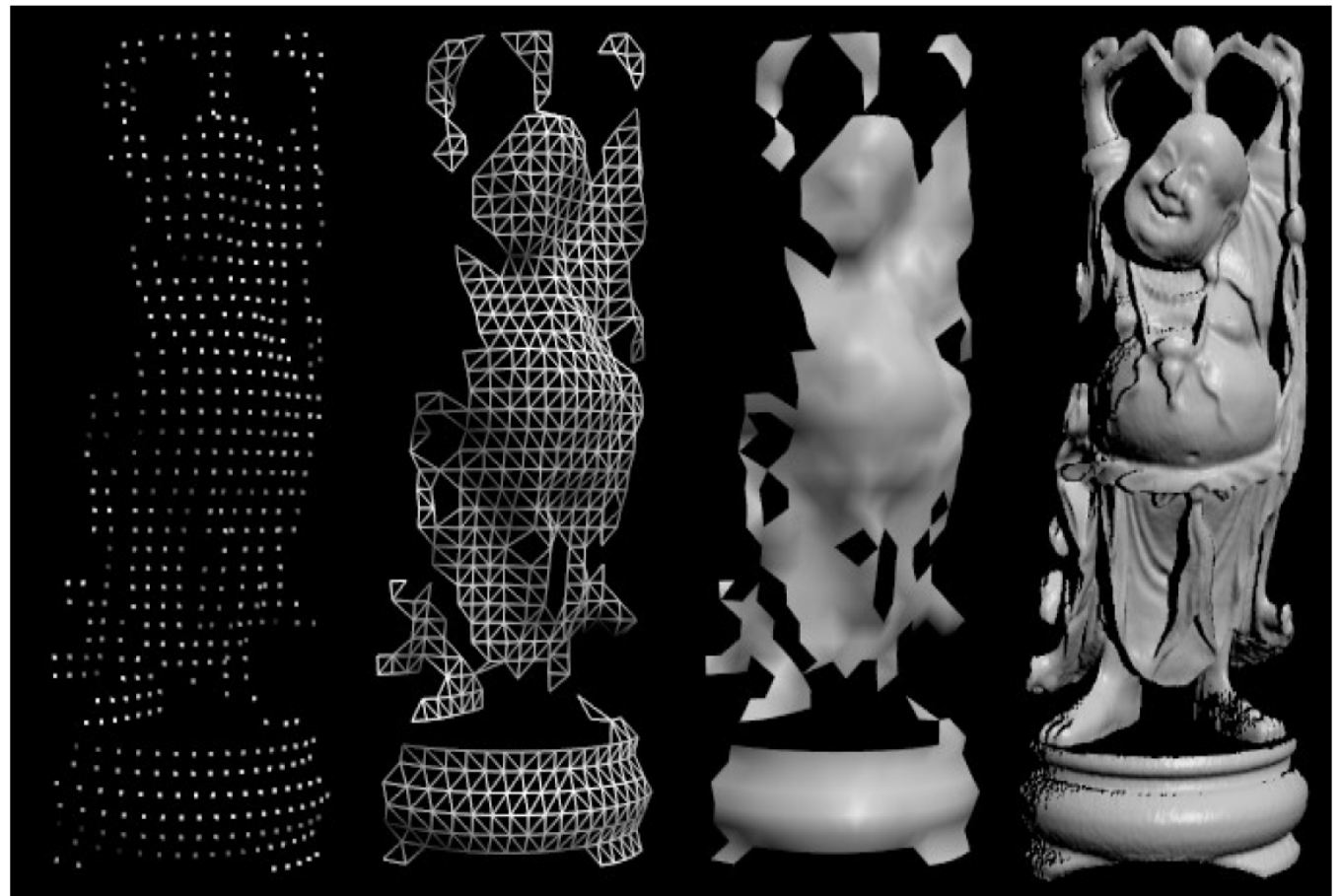
# Motivation

- Range images registration



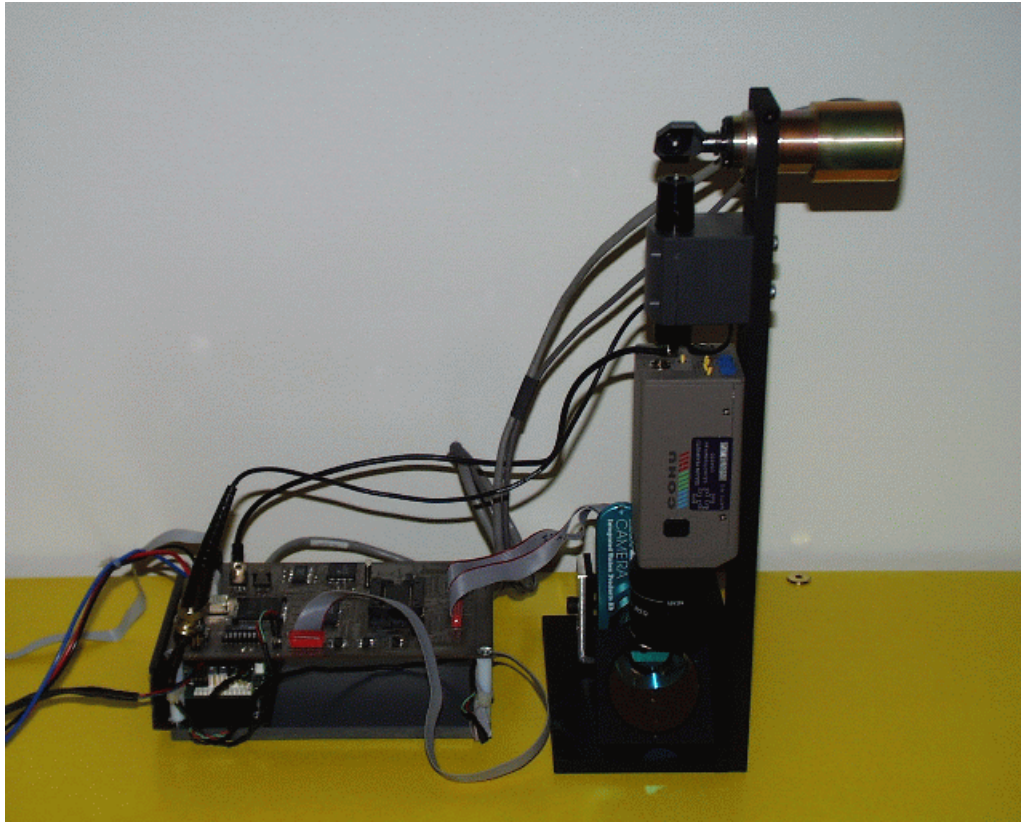
# Motivation

- Range images registration



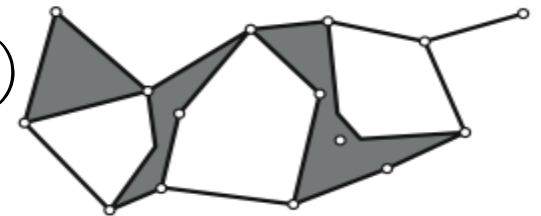
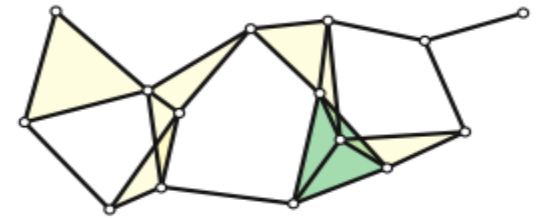
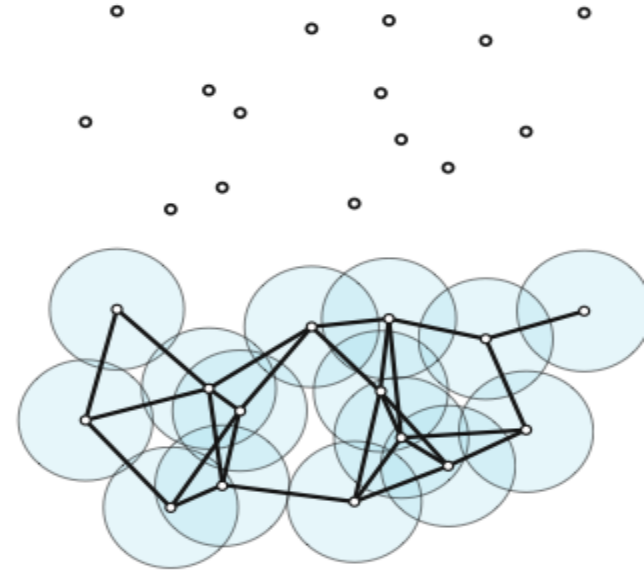


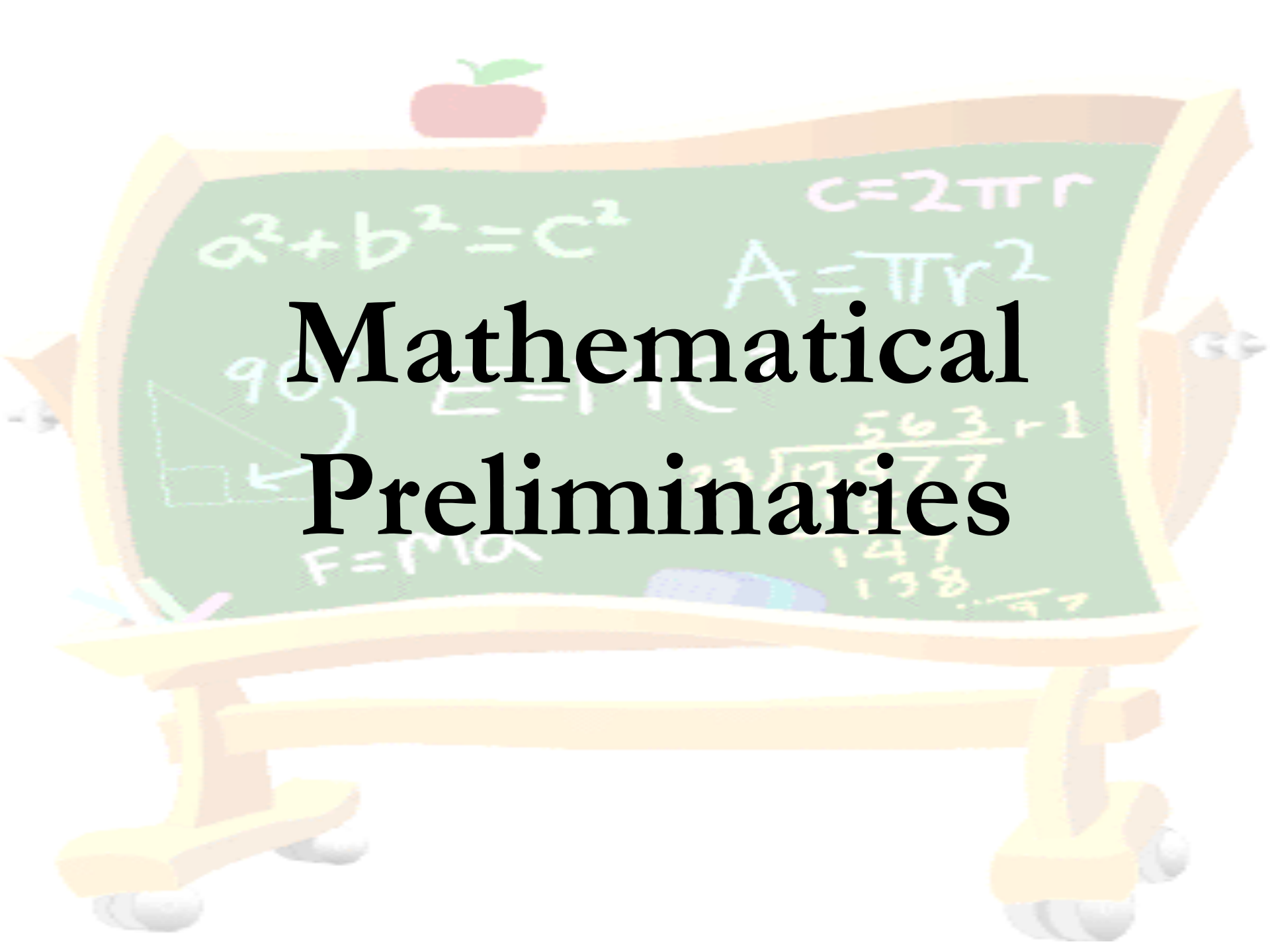
# Range Scanners



# Data Types

- Point sets
- Line segment sets (polylines)
- Implicit curves :  $f(x,y,z) = 0$
- Parametric curves :  $(x(u),y(u),z(u))$
- Triangle sets (meshes)
- Implicit surfaces :  $s(x,y,z) = 0$
- Parametric surfaces  $(x(u,v),y(u,v),z(u,v))$





The image features a green chalkboard on a yellow frame with wheels. At the top center, there is a red apple with a green leaf. The chalkboard contains several mathematical formulas and diagrams written in white chalk:

- Top left:  $a^2 + b^2 = c^2$
- Top right:  $C = 2\pi r$
- Middle right:  $A = \pi r^2$
- Middle left:  $90^\circ$  and a diagram of a right-angled triangle with a dashed line and an arrow pointing to the right angle.
- Middle:  $E = mc^2$
- Bottom left:  $F = ma$
- Bottom right: A long division problem: 
$$\begin{array}{r} 563 \text{ r } 1 \\ 23 \overline{) 12977} \\ \underline{46} \phantom{7} \\ 147 \phantom{7} \\ \underline{138} \phantom{7} \\ 97 \end{array}$$

# Mathematical Preliminaries

# Centroid

- The centroid of a data (point) set is the weighted mean of all data points presented in the set.
- For a data set  $\mathbf{A}$ , having  $n$  points, each denoted by  $a_i$ , the centroid is given by :-

$$\mu_A = \frac{1}{n} \sum_{i=1}^n a_i$$

# Variance

- A measure of the spread of the data in a data set  $\mathbf{A}$  with mean  $\mu_A$ :

$$\sigma_A^2 = \frac{\sum_{i=1}^n (a_i - \mu_A)^2}{(n - 1)}$$

- Variance is claimed to be the original statistical measure of spread of data.

# Covariance

- Covariance matrix gives a measure of similarity between 2 data sets to be matched.
- If  $\mu_A$  and  $\mu_B$  are the centroids of the data sets **A** and **B** respectively then, the covariance matrix between the two sets is given by:-

$$\sum_{AB} = \frac{\sum_{i=1}^n (a_i - \mu_A)(b_i - \mu_B)}{(n-1)}$$

# Inner Product

- Let  $\mathbf{a}$  and  $\mathbf{b}$  be two vectors defined as:  $\mathbf{a} = [a_1, \dots, a_n]^T$  and,  $\mathbf{b} = [b_1, \dots, b_n]^T$

- Inner (dot) product:  $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = \sum_{i=1}^n a_i b_i$

- Length (Euclidean norm) of a vector  $\mathbf{a}$  is normalized iff  $\|\mathbf{a}\| = 1$

$$\|\mathbf{a}\| = \sqrt{\mathbf{a}^T \cdot \mathbf{a}} = \sqrt{\sum_{i=1}^n a_i^2}$$

- The angle between two n-dimensional vectors  $\cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$
- An inner product is a measure of collinearity:

- $\mathbf{a}$  and  $\mathbf{b}$  are orthogonal iff  $\mathbf{a} \cdot \mathbf{b} = 0$

- $\mathbf{a}$  and  $\mathbf{b}$  are collinear iff  $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\|$

- A set of vectors is *linearly independent* if no vector is a linear combination of other vectors.

# Transformation Matrices

- Consider the following:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = 4 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

- The square (transformation) matrix scales (3,2)
- Now assume we take a multiple of (3,2)

$$2 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \end{bmatrix} = 4 \times \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$



# Transformation Matrices

- Scale vector  $(3,2)$  by a value 2 to get  $(6,4)$
- Multiply by the square transformation matrix
- And we see that the result is still scaled by 4.

## WHY?

A vector consists of both length and direction. Scaling a vector only changes its length and not its direction. This is an important observation in the transformation of matrices leading to formation of **eigenvectors and eigenvalues**.

Irrespective of how much we scale  $(3,2)$  by, the solution (under the given transformation matrix) is always a multiple of 4.

# Eigenvalue Problem

- The eigenvalue problem is any problem having the following form:

$$\mathbf{A} \cdot \mathbf{v} = \lambda \cdot \mathbf{v}$$

$\mathbf{A}$ :  $m \times m$  matrix

$\mathbf{v}$ :  $m \times 1$  non-zero vector

$\lambda$ : scalar

- Any value of  $\lambda$  for which this equation has a solution is called the eigenvalue of  $\mathbf{A}$  and the vector  $\mathbf{v}$  which corresponds to this value is called the eigenvector of  $\mathbf{A}$ .

# Eigenvalue Problem

- Going back to our example:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = 4 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$\mathbf{A} \cdot \mathbf{v} = \lambda \cdot \mathbf{v}$

- Therefore,  $(3,2)$  is an eigenvector of the square matrix  $\mathbf{A}$  and 4 is an eigenvalue of  $\mathbf{A}$
- The question is:

**Given matrix  $\mathbf{A}$ , how can we calculate the eigenvector and eigenvalues for  $\mathbf{A}$ ?**

# Calculating Eigenvectors & Eigenvalues

- Simple matrix algebra shows that:

$$\mathbf{A} \cdot \mathbf{v} = \lambda \cdot \mathbf{v}$$

$$\Leftrightarrow \mathbf{A} \cdot \mathbf{v} - \lambda \cdot \mathbf{I} \cdot \mathbf{v} = \mathbf{0}$$

$$\Leftrightarrow (\mathbf{A} - \lambda \cdot \mathbf{I}) \cdot \mathbf{v} = \mathbf{0}$$

- Finding the roots of  $|\mathbf{A} - \lambda \cdot \mathbf{I}|$  will give the eigenvalues and for each of these eigenvalues there will be an eigenvector

Example ...

# Calculating Eigenvectors & Eigenvalues

- Let

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

- Then:  
$$|A - \lambda I| = \left| \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = \left| \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right|$$
$$= \left| \begin{bmatrix} -\lambda & 1 \\ -2 & -3 - \lambda \end{bmatrix} \right| = (-\lambda \times (-3 - \lambda)) - (-2 \times 1) = \lambda^2 + 3\lambda + 2$$

- And setting the determinant to 0, we obtain 2 eigenvalues:

$$\lambda_1 = -1 \text{ and } \lambda_2 = -2$$

# Calculating Eigenvectors & Eigenvalues

- For  $\lambda_1$  the eigenvector is:

$$(A - \lambda_1 I) \cdot v_1 = 0$$

$$\begin{bmatrix} 1 & 1 \\ -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} v_{1:1} \\ v_{1:2} \end{bmatrix} = 0$$

$$v_{1:1} + v_{1:2} = 0 \quad \text{and} \quad -2v_{1:1} - 2v_{1:2} = 0$$

$$v_{1:1} = -v_{1:2}$$

- Therefore the first eigenvector is any column vector in which the two elements have equal magnitude and opposite sign.

# Calculating Eigenvectors & Eigenvalues

- Therefore eigenvector  $v_1$  is

$$v_1 = k_1 \begin{bmatrix} +1 \\ -1 \end{bmatrix}$$

where  $k_1$  is some constant.

- Similarly we find that eigenvector  $v_2$

$$v_2 = k_2 \begin{bmatrix} +1 \\ -2 \end{bmatrix}$$

where  $k_2$  is some constant.

# Properties of Eigenvectors and Eigenvalues

- Eigenvectors can only be found for square matrices and not every square matrix has eigenvectors.
- Given an  $m \times m$  matrix (with eigenvectors), we can find  $m$  eigenvectors.
- All eigenvectors of a symmetric<sup>\*</sup> matrix are perpendicular to each other, no matter how many dimensions we have.
- In practice eigenvectors are normalized to have unit length.

\*Note: covariance matrices are symmetric!



# Mean Square Error

- In statistics, the **mean squared error** or **MSE** of an estimator is one of many ways to quantify the amount by which an estimator differs from the true value of the quantity being estimated.
- The Mean Square error (MSE) between 2 data sets **A** and **B** having  $n$  points each is given by :-

$$MSE = \frac{1}{n} \sum_{i=1}^n \|a_i - b_i\|^2$$

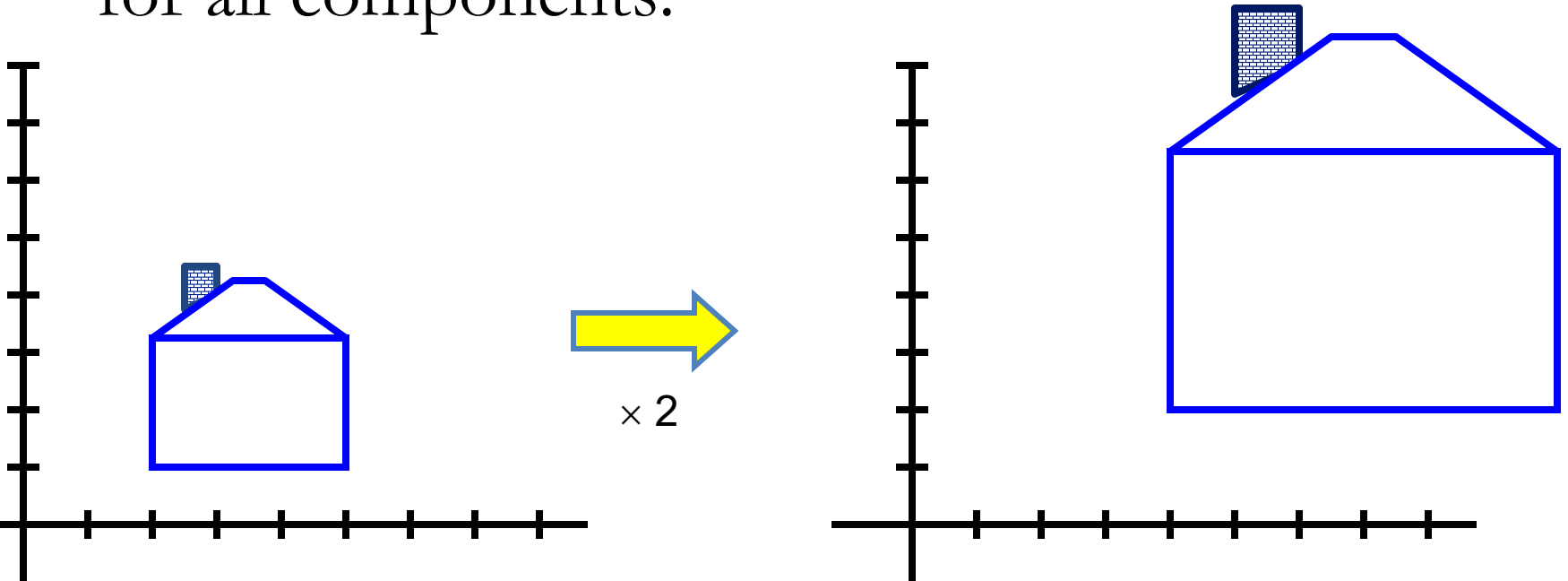
where  $\|\cdot\|$  denotes the L2-Norm/Euclidean Norm between two data points.

# Rigid Transformations

- Rigid transformations can be classified into:
  - Scaling
  - Rotation
  - Translation

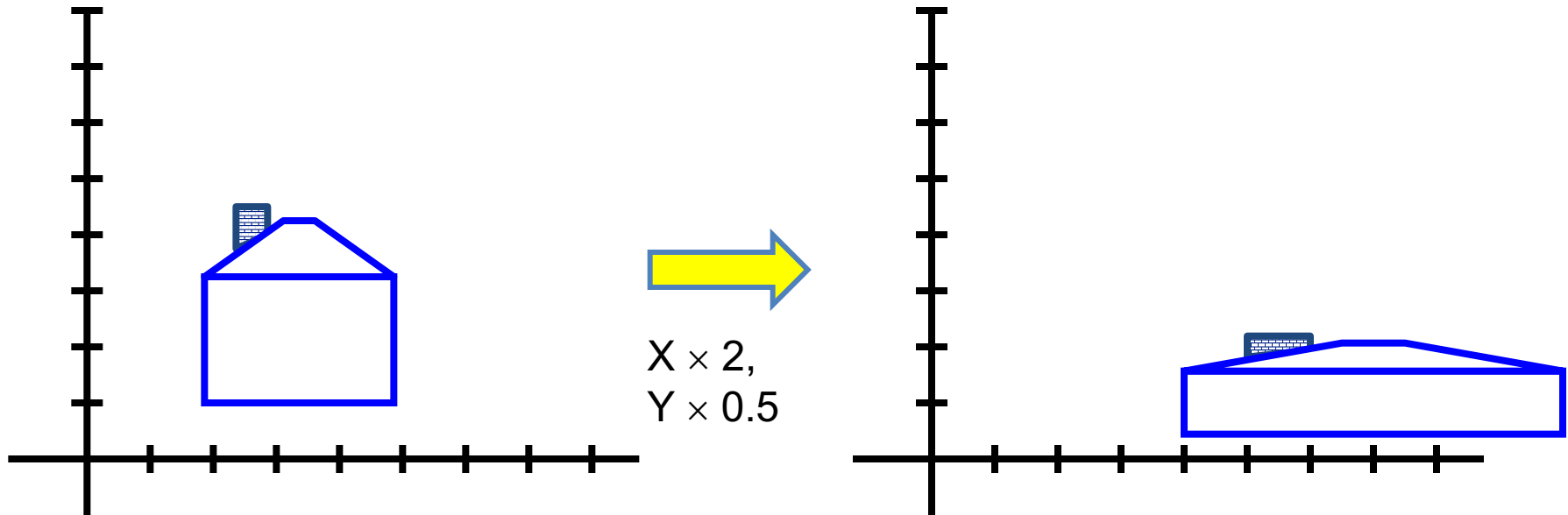
# Transformations – Scaling in 2D

- **Scaling** a coordinate means multiplying each of its components by a scalar
- **Uniform scaling** means this scalar is the same for all components:



# Transformations - Scaling in 2D

- **Non-uniform scaling:** different scalars per component:



- How can we represent this in matrix form?

# Transformations - Scaling in 2D

- Scaling operation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax \\ by \end{bmatrix}$$

- Or, in matrix form:

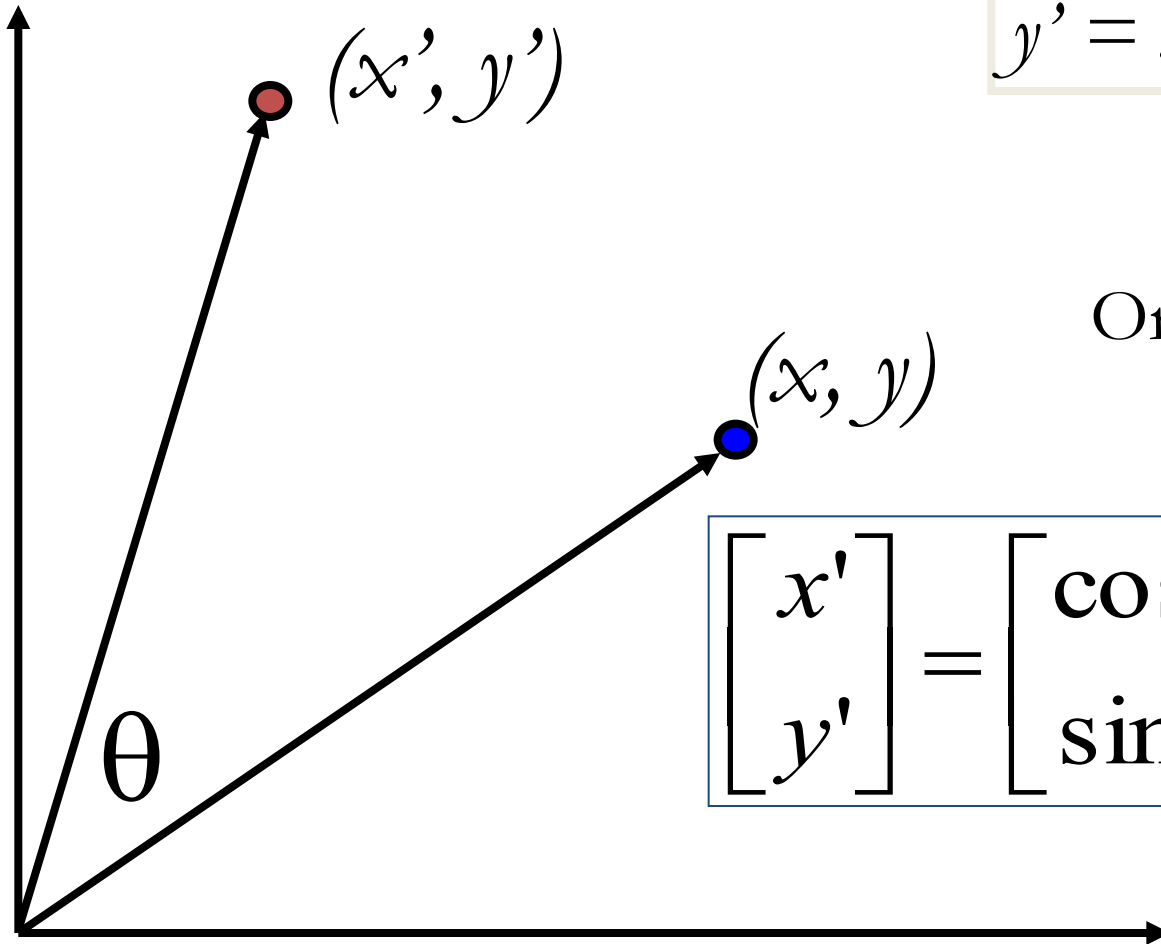
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix}} \begin{bmatrix} x \\ y \end{bmatrix}$$

# Transformations – Rotation in 2D

$$\begin{aligned}x' &= x \cos(\theta) - y \sin(\theta) \\y' &= x \sin(\theta) + y \cos(\theta)\end{aligned}$$

Or, in matrix form

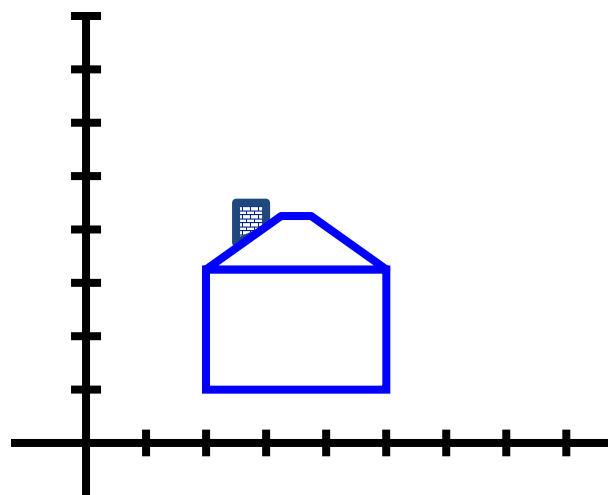
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



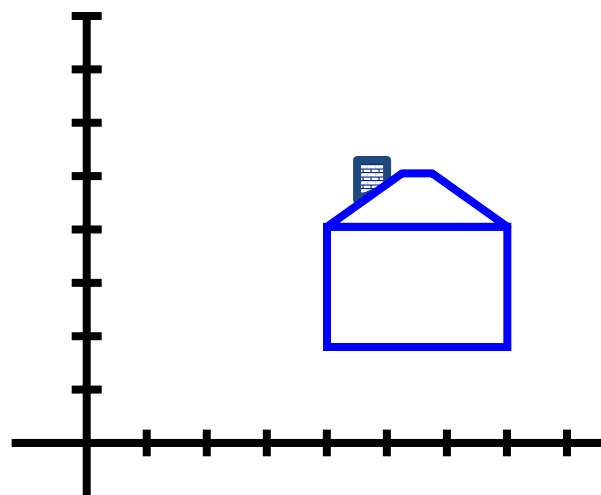
# Transformations – Translation in 2D

$$x' = x + t_x$$

$$y' = y + t_y$$

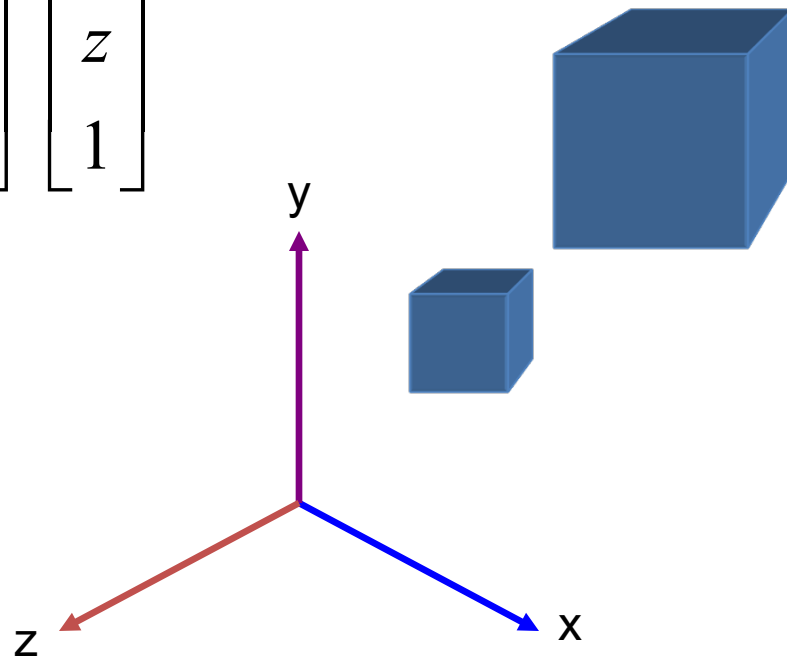


$$t_x = 2$$
$$t_y = 1$$



# Transformations – Scaling in 3D

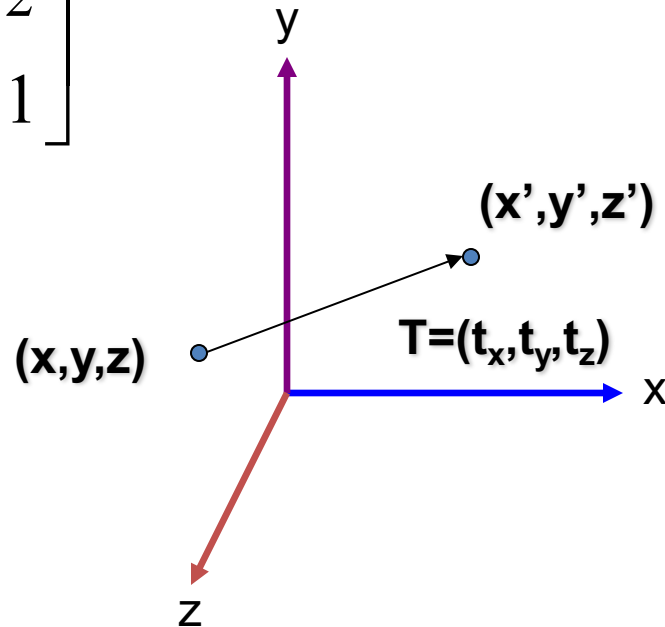
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$





# Transformations – Translation in 3D

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



# Transformations – Rotation in 3D

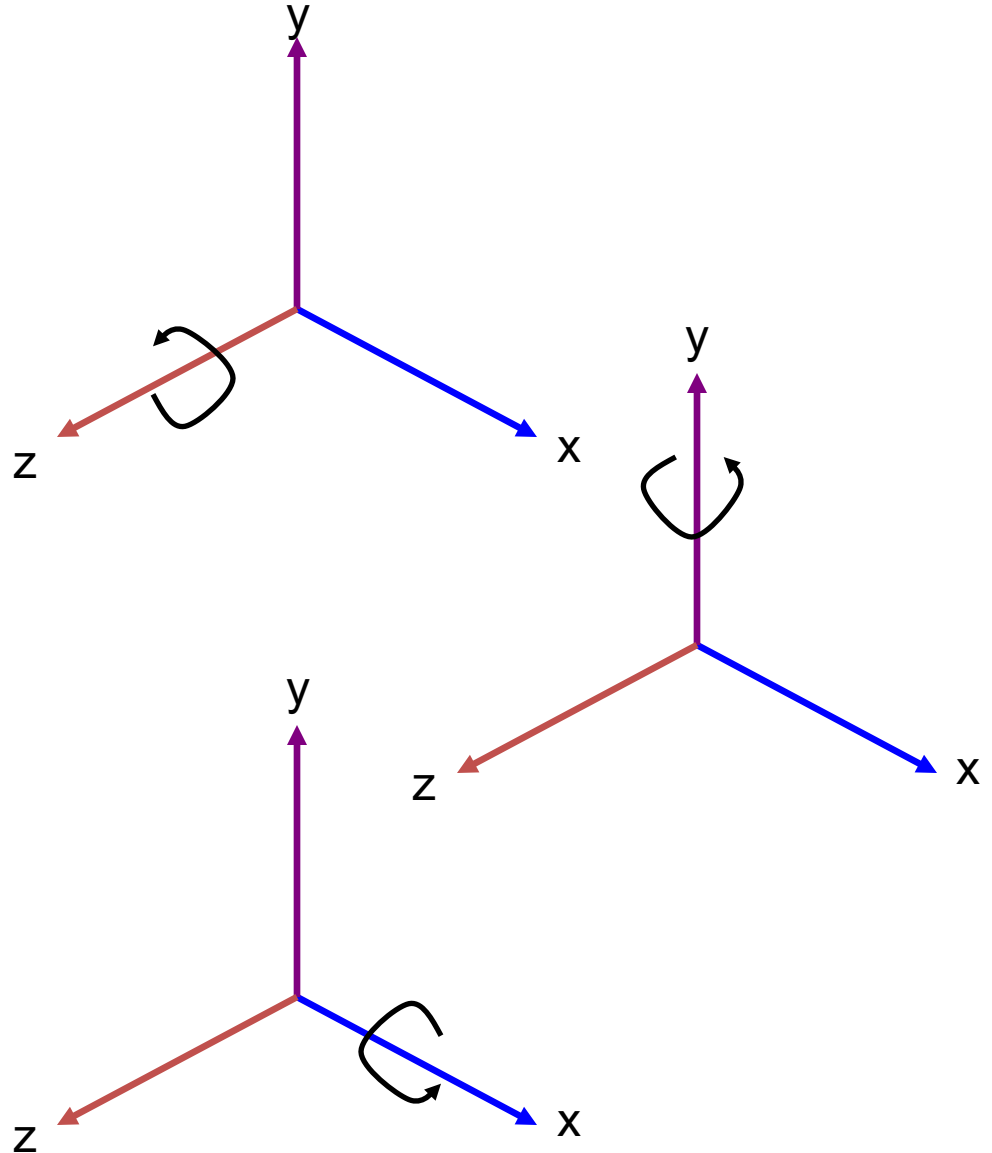
- How should we specify rotations?
- In 2D, it was always counterclockwise in the  $xy$ -plane.
- In 3D, we have more choices
  - $xz$ -plane,  $yz$ -plane, an arbitrary plane.
- We could specify these in terms of the vector perpendicular to the plane of rotation.
  - $z$  axis,  $y$ -axis,  $x$ -axis, arbitrary axis

# Rotation in 3D – Euler Angles

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 & 0 \\ \sin \gamma & \cos \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



# Quaternions

- A quaternion  $\mathbf{q}$  can be thought of as:
  - A vector with four components:  $\mathbf{q} = [q_0, q_x, q_y, q_z]^T$ .
  - A composite of a scalar and an ordinary vector:  $\mathbf{q} = [q_0, \mathbf{q}]$ .
  - A complex number with three different imaginary parts:  $\mathbf{q} = q_0 + iq_x + jq_y + kq_z$ .

# Products of Quaternions

- Multiplication of quaternions can be defined in terms of products of their components.
- Suppose that we let :  $i^2 = j^2 = k^2 = -1$

$$ij = k, jk = i, ki = j$$

$$\text{and } ji = -k, kj = -i, ik = -j$$

- Then if  $\mathbf{r} = r_0 + ir_x + jr_y + kr_z$ ,

we get;

- The product  $\mathbf{qr}$  has a similar form with six of the signs changed.

$$\begin{aligned} \mathbf{rq} = & (r_0q_0 - r_xq_x - r_yq_y - r_zq_z) \\ & + i(r_0q_x + r_xq_0 + r_yq_z - r_zq_y) \\ & + j(r_0q_y - r_xq_z + r_yq_0 + r_zq_x) \\ & + k(r_0q_z + r_xq_y - r_yq_x + r_zq_0) \end{aligned}$$

# Products of Quaternions

- The product of two quaternions can also be expressed in terms of the product of an orthogonal 4x4 matrix and a vector with four components.
- One may choose to expand either the first or the second quaternion in a product into an orthogonal 4x4 matrix as follows:

- Note that  $\bar{\mathbf{R}}$  differs from  $\mathbf{R}$  in that the lower-right-hand 3x3 sub-matrix is transposed.

$$\begin{aligned}
 \mathbf{r}\mathbf{q} &= \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & -r_z & r_y \\ r_y & r_z & r_0 & -r_x \\ r_z & -r_y & r_x & r_0 \end{bmatrix} \mathbf{q} = \mathbf{R}\mathbf{q} \\
 \mathbf{q}\mathbf{r} &= \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{bmatrix} \mathbf{q} = \bar{\mathbf{R}}\mathbf{q}
 \end{aligned}$$

# Dot Product of Quaternions

- Considering a quaternion as a vector of four components, the dot product of two quaternions is the sum of products of corresponding components:

$$\mathbf{r} \cdot \mathbf{q} = r_0 q_0 + r_x q_x + r_y q_y + r_z q_z$$

- The square of the magnitude of a quaternion is the dot product of the quaternion with itself.

$$\|\mathbf{q}\|^2 = \mathbf{q} \cdot \mathbf{q} = q_0^2 + q_x^2 + q_y^2 + q_z^2$$

- A unit quaternion is a quaternion whose magnitude equals 1.

$$\mathbf{q} \cdot \mathbf{q} = 1$$

# Conjugate of Quaternions

- Taking the conjugate of a quaternion negates its imaginary part, thus:  $\mathbf{q}^* = q_0 - iq_x - jq_y - kq_z$ .
- The 4x4 matrix associated with the conjugate of a quaternion is just the transpose of the matrix associated with the quaternion itself, i.e.  $\mathbf{Q}^T$

where

$$\mathbf{Q} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix}$$



# Conjugate of Quaternions

- Since  $\mathbf{Q}$  is an orthogonal matrix, thus the product with its transpose is a diagonal matrix, that is:  $\mathbf{Q}\mathbf{Q}^T = (\mathbf{q}\cdot\mathbf{q})\mathbf{I}$ , where  $\mathbf{I}$  is the 4x4 identity matrix.
- Correspondingly, the product of  $\mathbf{q}$  and  $\mathbf{q}^*$  is real, that is:  
$$\mathbf{q}\mathbf{q}^* = q_0^2 + q_x^2 + q_y^2 + q_z^2 = \mathbf{q}\cdot\mathbf{q}$$
- We immediately conclude that a non-zero quaternion has an inverse:  
$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{\mathbf{q}\cdot\mathbf{q}}$$
- In case of a unit quaternion ( $\mathbf{q}\cdot\mathbf{q} = 1$ ), the inverse is just the conjugate

# Useful Properties of Products

- Since the matrices associated with quaternions are orthogonal, hence dot products are preserved, that is:  
 $(\mathbf{qv}) \cdot (\mathbf{qr}) = (\mathbf{q} \cdot \mathbf{q}) (\mathbf{v} \cdot \mathbf{r})$

- **Proof:**  $(\mathbf{qv}) \cdot (\mathbf{qr}) = (\mathbf{Qv}) \cdot (\mathbf{Qr}) = (\mathbf{Qv})^T (\mathbf{Qr})$   
 $= \mathbf{v}^T \mathbf{Q}^T \mathbf{Qr} = \mathbf{v}^T (\mathbf{q} \cdot \mathbf{q}) \mathbf{r} = (\mathbf{q} \cdot \mathbf{q}) (\mathbf{v} \cdot \mathbf{r})$

- In the case of a unit quaternion :  $(\mathbf{qv}) \cdot (\mathbf{qr}) = (\mathbf{v} \cdot \mathbf{r})$
- A special case follows immediately:

$$(\mathbf{qr}) \cdot (\mathbf{qr}) = (\mathbf{q} \cdot \mathbf{q}) (\mathbf{r} \cdot \mathbf{r})$$

that is the magnitude of a product is just the product of the magnitudes.

# Vectors

- Vectors can be represented by purely imaginary quaternions. If  $\mathbf{r} = (x, y, z)^T$ , we can use the quaternion:  $\mathbf{r} = 0 + i x + j y + k z$ .
- Similarly, scalars can be represented by using real quaternions.
- Note that the matrices  $\mathbf{R}$  and  $\overline{\mathbf{R}}$  associated with a purely imaginary quaternion and its conjugate are skew symmetric, that is

$$\mathbf{R}^T = -\mathbf{R} \quad \text{and} \quad \overline{\mathbf{R}}^T = -\overline{\mathbf{R}}$$

# Unit Quaternions and Rotation

- Since the length of a vector is not changed by rotation nor is the angle between vectors, **thus rotation preserves dot product.** (note that  $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos\theta$ ).
- Now, we have already established that **multiplication by a unit quaternion preserves dot products** between two quaternions, that is  $(\mathbf{q}\mathbf{v}) \cdot (\mathbf{q}\mathbf{r}) = (\mathbf{q} \cdot \mathbf{q}) (\mathbf{v} \cdot \mathbf{r}) = \mathbf{v} \cdot \mathbf{r}$
- And since a vector can be represented as a purely imaginary quaternion, thus **we can represent rotation by using unit quaternions if we can find a way of mapping purely imaginary quaternions into purely imaginary quaternions in such a way that preserves dot products.**

# Unit Quaternions and Rotation

- However, we can not use simple multiplication to represent rotation, since the product of a unit quaternion and a purely imaginary quaternion is generally not a purely imaginary quaternion.
- What we can use instead is the **composite product** defined as follows:
  - Let  $\mathbf{r}$  be a purely imaginary quaternion representing a 3D point (vector) such that :  $\mathbf{r} = 0 + i x + j y + k z$ .
  - Let  $\mathbf{q}$  be a unit quaternion such that  $\mathbf{q} = q_0 + i q_x + j q_y + k q_z$
  - Rotating the vector (point)  $\mathbf{r}$  by a unit quaternion  $\mathbf{q}$  can be defined as  $\mathbf{r}' = \mathbf{q}\mathbf{r}\mathbf{q}^*$ , where  $\mathbf{r}'$  is a purely imaginary quaternion representing the vector  $\mathbf{r}$  after rotation by  $\mathbf{q}$ .

# Unit Quaternions and Rotation

*Prove that the composite product leads to purely imaginary quaternion, hence it can be used to represent rotation.*

## Proof:

- The objective is representing the composite product as a matrix multiplied by a vector  $\mathbf{r}$ . let's find this matrix in terms of the matrices associated to the unit quaternion  $q$  and its conjugate.

$$qrq^* = (\mathbf{Qr})q^* = \overline{\mathbf{Q}}^T (\mathbf{Qr}) = (\overline{\mathbf{Q}}^T \mathbf{Q})\mathbf{r}$$

$$\text{where } \mathbf{Q} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} \text{ and } \overline{\mathbf{Q}} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & q_z & -q_y \\ q_y & -q_z & q_0 & q_x \\ q_z & q_y & -q_x & q_0 \end{bmatrix}$$

Where  $\mathbf{Q}$  and  $\overline{\mathbf{Q}}$  are the 4x4 matrices corresponding to the unit quaternion  $q$ .

# Unit Quaternions and Rotation

Proof: cont

$$\begin{aligned} \therefore \bar{\mathbf{Q}}^T \mathbf{Q} &= \begin{bmatrix} q_0 & q_x & q_y & q_z \\ -q_x & q_0 & -q_z & q_y \\ -q_y & q_z & q_0 & -q_x \\ -q_z & -q_y & q_x & q_0 \end{bmatrix} \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{q} \cdot \mathbf{q} & 0 & 0 & 0 \\ 0 & (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix} \end{aligned}$$

where  $\mathbf{q} \cdot \mathbf{q} = q_0^2 + q_x^2 + q_y^2 + q_z^2$

$\therefore \mathbf{q} \mathbf{r} \mathbf{q}^* = (\bar{\mathbf{Q}}^T \mathbf{Q}) \mathbf{r}$  is a purely imaginary quaternion if  $\mathbf{r}$  is a purely imaginary quaternion.

# Unit Quaternions and Rotation

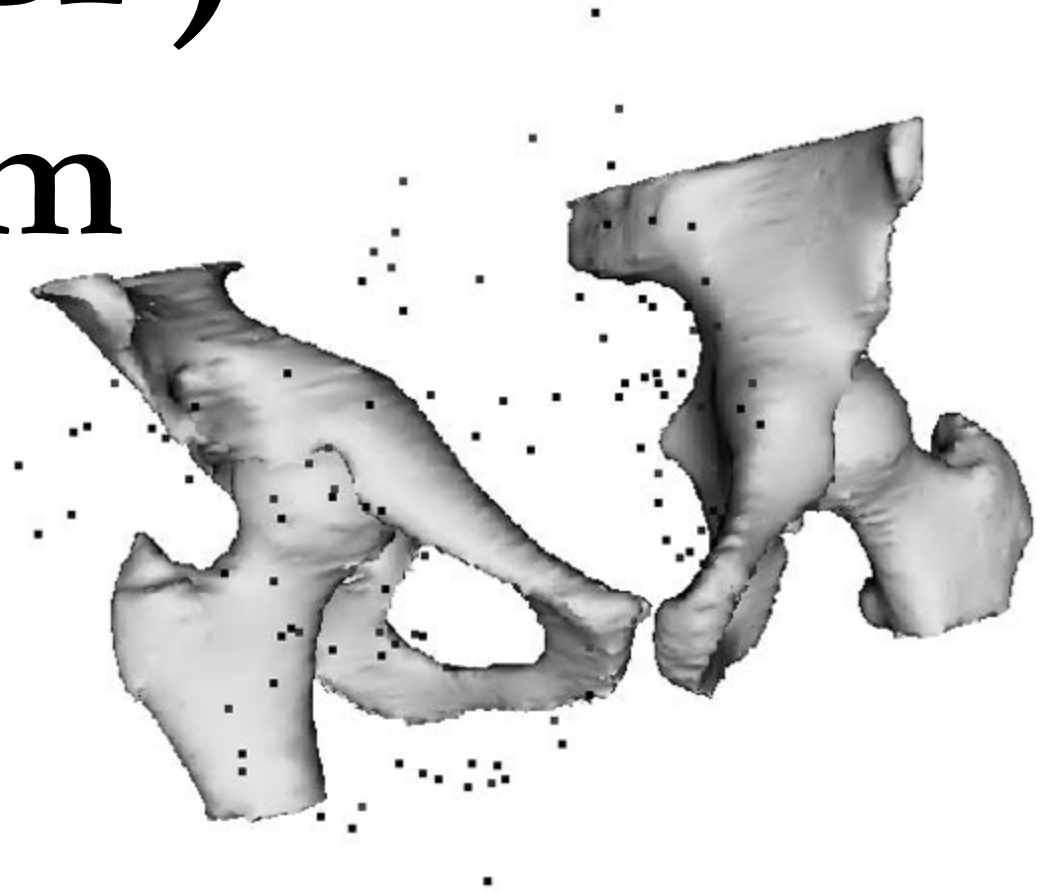
## Proof: cont

- Since  $q$  is a quaternion, then  $Q$  and  $\bar{Q}$  are orthogonal matrices by definition.
- Since  $q$  is a unit quaternion, then  $Q$  and  $\bar{Q}$  are orthonormal matrices,.
- Hence the lower-right-hand 3x3 sub-matrix of  $\bar{Q}^T Q$  must also be orthonormal , hence it is the rotation matrix  $R$  that take  $r$  to  $r'$  such that  $r' = Rr$ .
- The expansion of  $\bar{Q}^T Q$  provides an explicit method for computing the orthonormal rotation matrix  $R$  from the components of the unit quaternion  $q$ .



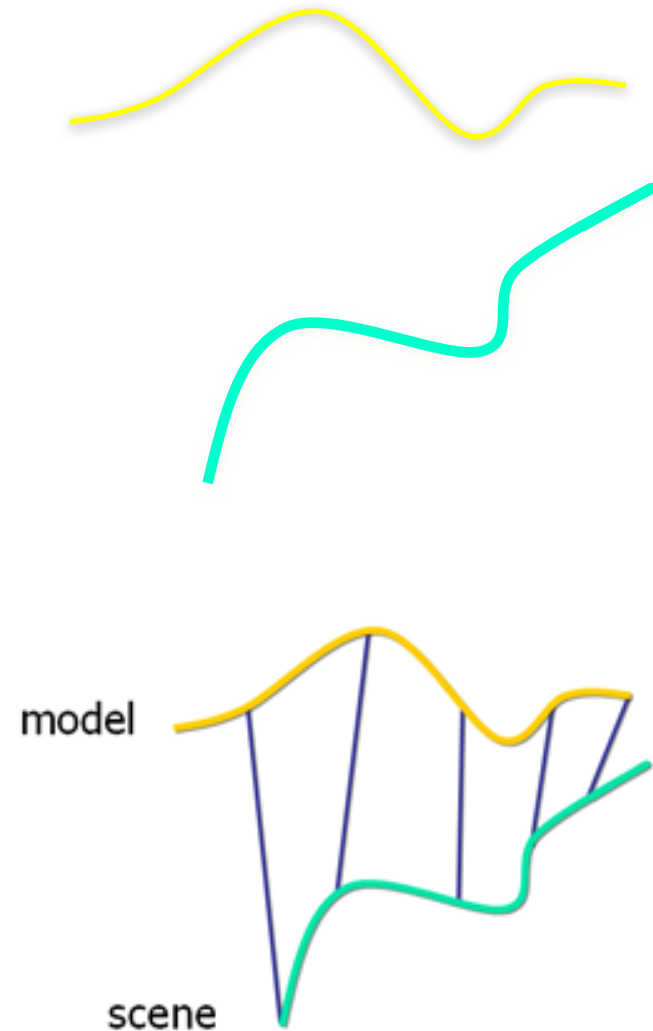


# Iterative Closest Point (ICP) Algorithm



# Problem Statement

- Given a model shape which maybe represented as:
  - Point Sets, Line Segment Sets, Implicit Curves, Parametric Curves, Triangle Sets, Implicit Surfaces, Parametric Surfaces
- Given a scene shape which is represented as a point set, the scene shape may correspond to the model shape
- It is required to estimate the optimal rotation, translation and scaling that aligns or registers the scene shape to the model shape
- Main Application is to register digitize(sensed) data from un-fixtured rigid objects with an idealized geometric model prior to shape inspection.

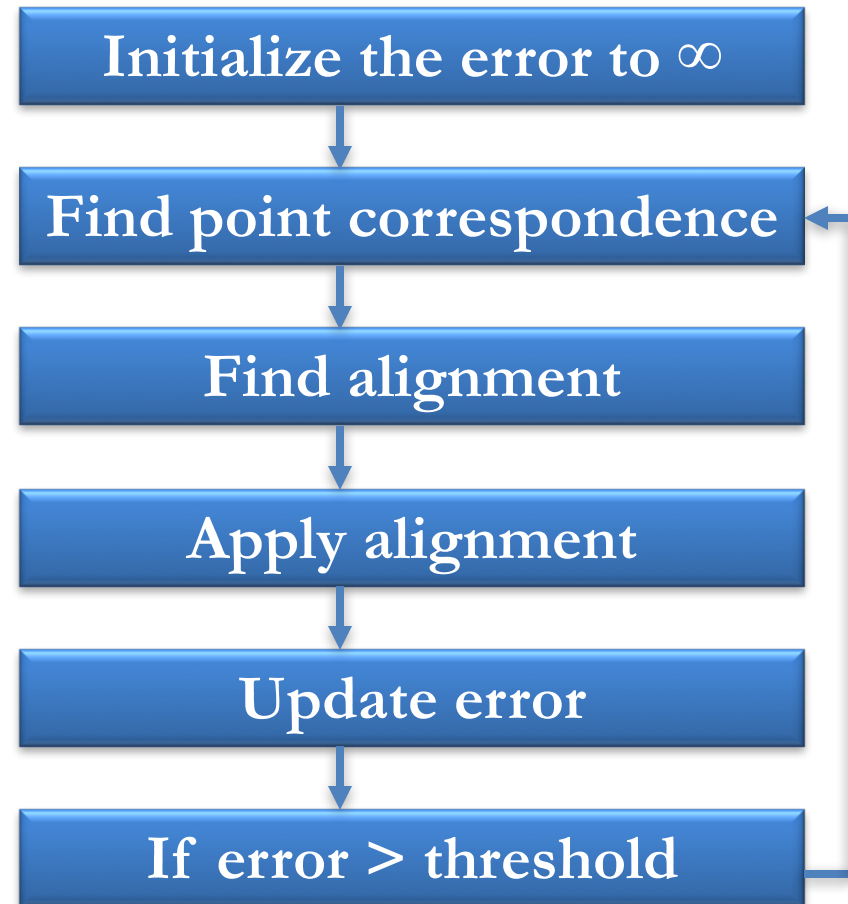


# Main Idea of ICP

1. Begin with initial rotation, translation and scaling (initial value for registration parameters).
2. Fix the model shape and start moving the scene shape by applying the initial registration parameters. i.e. scale, rotate and then translate.
3. Compute the error metric that reflects the dissimilarity of the scene shape from the model shape.
4. If the error is minimum, we have correctly aligned the scene shape to the model shape, return with the aligned scene shape.
5. Else, calculate the new values for the registration parameters and go back to step 2 with the new parameter values.

# Algorithm Outline

1. Initialize registration parameters  $(R, t, s)$  and registration error;  $\text{Error} = \infty$
2. For each point in the scene shape, find the corresponding closest point in the model shape.
3. Calculate registration parameters given point correspondences obtained from step 2.
4. Apply the alignment to the scene shape.
5. Calculate the registration error between the currently aligned scene shape and the model shape.
6. If  $\text{error} > \text{threshold}$ , return to step 2, else return with new scene shape.



# Nomenclature

- Let the model shape be represented as a set of points,  $m_i$ , where  $i = 1, 2, \dots, N_M$ .
  - Where  $N_M$  is the number of points in the model shape and  $M = \{m_i\}$  denotes the model point set
  - Note:  $m_i = [m_{xi} \ m_{yi} \ m_{zi}]^T$  in the case of 3D
- Let the scene shape be represented as a set of points  $p_i$ , where  $i = 1, 2, \dots, N_P$ .
  - Where  $N_P$  is the number of points in the scene shape and  $P = \{p_i\}$  denotes the scene point set.
  - Note:  $p_i = [p_{xi} \ p_{yi} \ p_{zi}]^T$  in the case of 3D

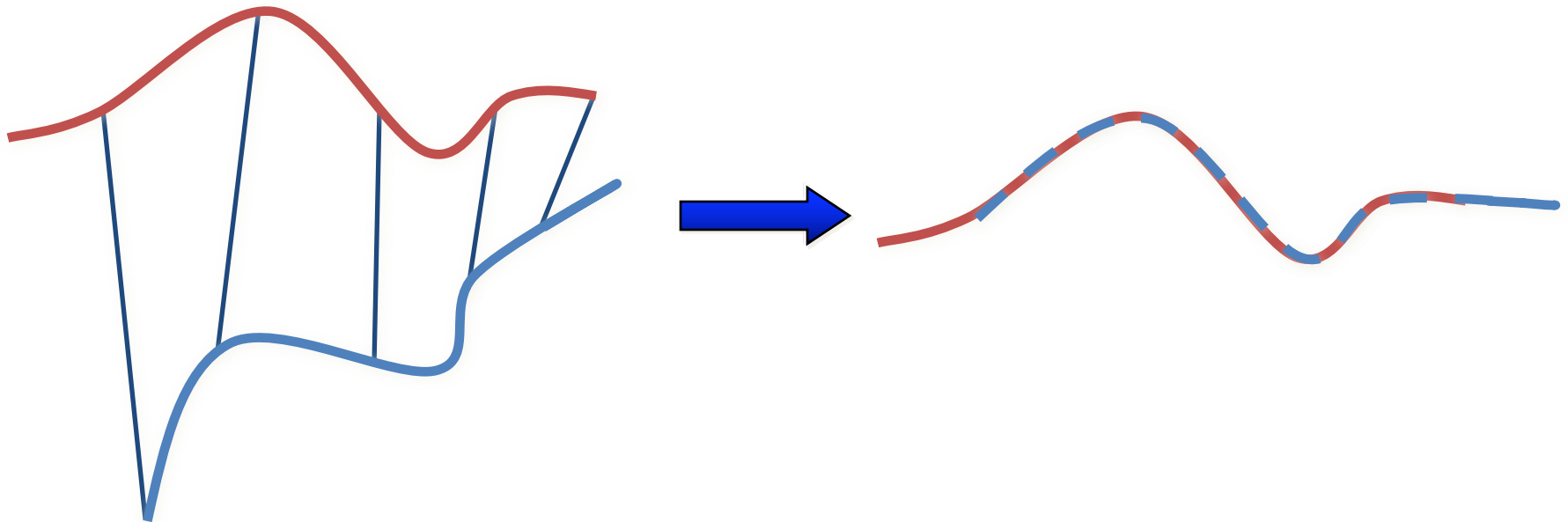
# Nomenclature

- Registration parameters.
  - $s$ : a scalar value which represents the scaling parameter
  - $t$ : a vector representing translation parameters. In 3D case  $t = [t_x, t_y, t_z]^T$
  - $R(\cdot)$ : an operator which applies rotation to its argument (a point).

Note:  $R(\cdot)$  will have a different definition according to the way it is used to represent rotation. E.g. Euler angles  $(\Theta_x, \Theta_y, \Theta_z)$ , rotation matrix  $\mathbf{R}$  (3x3 orthogonal matrix) or quaternion  $\mathbf{q}$  (rotation angle and axis of rotation)

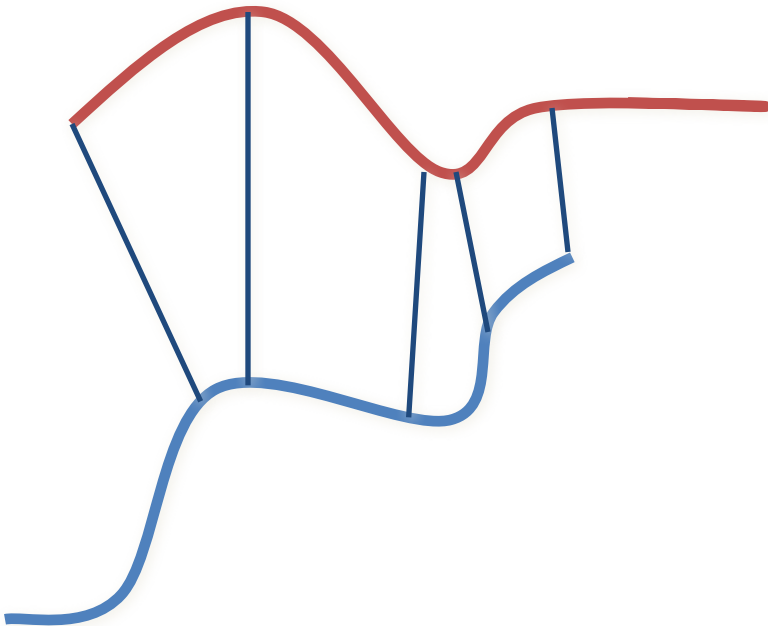
# Finding Correspondences

- If correct correspondences are known, we can find correct relative rotation/translation



# Finding Correspondences

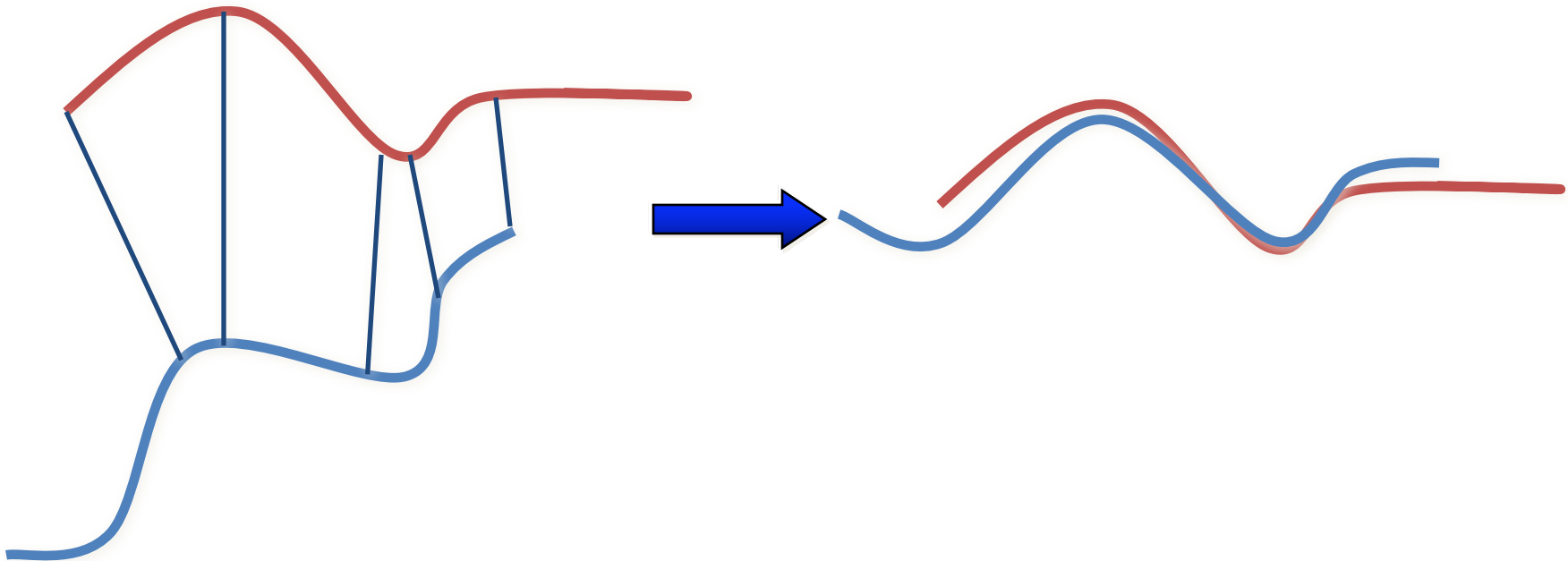
- How to find correspondences: User input? Feature detection? Signatures?
- Alternative: assume **closest** points correspond





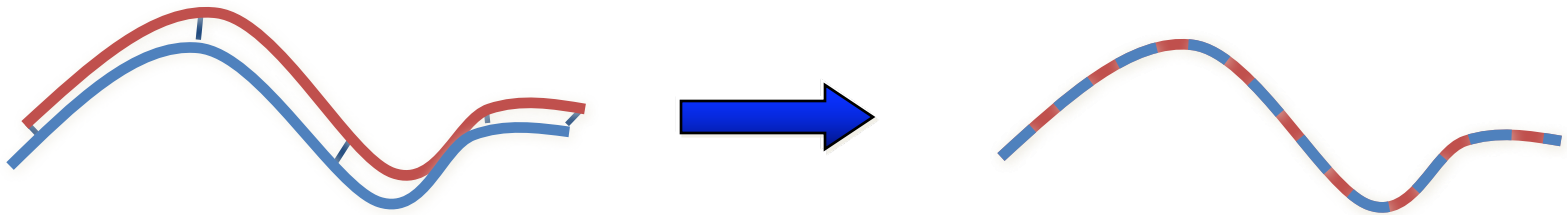
# Finding Correspondences

- How to find correspondences: User input? Feature detection? Signatures?
- Alternative: assume **closest** points correspond



# Finding Correspondences

- Converges if starting position “close enough”



# Finding Correspondences

- For every point,  $p_i$ , in the scene shape  $P = \{p_i\}$ ,  $i=1, \dots, N_P$ , we search for the closest point  $m_j$  in the model shape  $M$  to the scene point  $p_i$  using the Euclidean distance.
- Given two points  $p_i$  and  $m_i$ , the Euclidean distance can be computed as follows:

$$\begin{aligned} d(p_i, m_k) &= \|p_i - m_k\| \\ &= \sqrt{(p_{xi} - m_{xk})^2 + (p_{yi} - m_{yk})^2 + (p_{zi} - m_{zk})^2} \end{aligned}$$

- Given a scene point  $p_i$  and the model point set  $M$ , the Euclidean distance between  $p_i$  and  $M$  can be found as:

$$d(p_i, M) = \min_{k=1, \dots, N_M} d(p_i, m_k) = \min_{k=1, \dots, N_M} \|p_i - m_k\|$$

# Finding Correspondences

- The closest point  $m_j \in M$  (model point set) satisfies the equality:  
i.e.

$$d(p_i, m_j) = d(p_i, M)$$

- In other words,  $j$  is the index of the closest point  $p_i$

$$j = \arg \min_{k=1, \dots, N_M} d(p_i, m_k)$$

- The closest point in the model set  $M$  that yields the minimum distance will be denoted by  $y_i$ . That is  $y_i = m_j$  such that  $d(p_i, y_i) = d(p_i, M)$ . Now  $p_i$  corresponds to  $y_i$ .

- Let  $Y$  denote the resulting set of closest points and  $C$  be the closest point operator such that  $Y = C(P, M)$

$$\{y_i\} \leftarrow \{p_i\} \xrightarrow{\quad} \{m_k\}$$

# Alignment Calculation

- **Given:** A set of point correspondences between the scene shape  $P$  and the model shape  $M$ , where  $Y \subseteq M$  denotes the set of closest points to  $P$ , such that  $y_i$  is the corresponding closest point to  $p_i$ , where  $i=1,2,\dots,N_P$ .
- **Required:** Find the optimal registration parameters (scaling, rotation and translation) which brings the scene points  $P$  to the closest model points  $Y$ .
- **Approach:** Quaternion based method

# Alignment Calculation

- Let  $N_p$  be the number of points correspondences.
- The measurement coordinates in the scene and model coordinate systems will be denoted by  $P=\{p_i\}$  and  $Y=\{y_i\}$ , respectively, where  $i$  ranges from 1 to  $N_p$ .
- We are looking for a transformation of the form:

$$Y = sR(P) + t \quad (1)$$

which registers (aligns) the scene points  $P$  to the corresponding model points  $Y$ .

where:  $s$ : is a scale factor

$t$ : is the translational offset

$R(P)$  denotes the rotated version of the points  $P$ .

Note: At this time we do not use any particular notation for rotation.

# Alignment Calculation

- Rotation is a linear operation and it preserves lengths such that:

$$\|R(p)\|^2 = \|p\|^2$$

Where  $\|p\|^2 = p \cdot p$  is the square of the length of the vector point  $p$ .

- Since correspondences are not perfect, we will not be able to find a scale factor, a translation and a rotation such that the transformation equation(1) is satisfied for each point.
- Instead there will be a residual error for each point pair (correspondence) defined as follows:

$$e_i = y_i - \underbrace{(sR(p_i) + t)}_{\text{Transformed version of the scene point } p_i} \quad (2)$$

# Alignment Calculation

- We will minimize the total error defined as the sum of squares of these errors:

$$E = \sum_{i=1}^{N_P} \| e_i \|^2$$

- We will consider the variation of the total error first with translation, then with scale and finally with respect to rotation.



# Finding the Translation

- It is useful to refer all points to their centroids which are defined as follows:

$$\mu_P = \frac{1}{N_P} \sum_{i=1}^{N_P} p_i \quad \text{and} \quad \mu_Y = \frac{1}{N_P} \sum_{i=1}^{N_P} y_i$$

- Lets denote the new points by:

$$p_i' = p_i - \mu_P \quad \text{and} \quad y_i' = y_i - \mu_Y$$

- After this transformation, the point sets become zero mean, i.e

$$\frac{1}{N_P} \sum_{i=1}^{N_P} p_i' = 0 \quad \text{and} \quad \frac{1}{N_P} \sum_{i=1}^{N_P} y_i' = 0$$

# Finding the Translation

- Now, the error term can be re-written as follows:

$$e_i = y_i' - (sR(p_i') + t') \quad (3)$$

$$\text{where} \quad t' = t - \mu_Y + sR(\mu_P) \quad (4)$$

$t'$  is the new translational offset after bringing the points to the origin (i.e. zero-mean).

Q: Prove equation (4)

# Finding the Translation

## Proof:

Set equation (2) equal to equation (3) since the residual errors of the points before and after bringing the points to the origin are equal.

$$\text{Thus } y_i - (sR(p_i) + t) = y_i' - (sR(p_i') + t')$$

$$\text{Since } y_i' = y_i - \mu_Y \quad \text{and} \quad p_i' = p_i - \mu_P$$

$$\text{Therefore } y_i - sR(p_i) - t = y_i - \mu_Y - sR(p_i - \mu_P) - t'$$

Since rotation is a linear operator

$$y_i - sR(p_i) - t = y_i - \mu_Y - sR(p_i) + sR(\mu_P) - t'$$

$$t' = t - \mu_Y + sR(\mu_P)$$



# Finding the Translation

Hence the sum of squares of errors becomes:

$$\begin{aligned} E &= \sum_{i=1}^{N_P} \| e_i \|^2 = \sum_{i=1}^{N_P} \| y_i' - sR(p_i') - t' \|^2 \\ &= \sum_{i=1}^{N_P} \left[ \| y_i' - sR(p_i') \|^2 - 2t' \| y_i' - sR(p_i') \| + \| t' \|^2 \right] \\ &= \sum_{i=1}^{N_P} \| y_i' - sR(p_i') \|^2 - 2t' \sum_{i=1}^{N_P} \| y_i' - sR(p_i') \| + N_P \| t' \|^2 \end{aligned} \quad (5)$$

Now the sum in the middle of the expression in (5) is zero since the points are referred to the centroid (i.e. zero-mean and rotation and scaling don't affect the mean.)

Hence, The first and third terms are left:

$$E = \sum_{i=1}^{N_P} \| y_i' - sR(p_i') \|^2 + N_P \| t' \|^2 \quad (6)$$

# Finding the Translation

- Remember that we are looking for the optimal translational offset  $t'$  which will minimize the total error  $E$ .
- Since the first term in (6) does not depend on the translation and the second term in (6) can not be negative since:

$$\|t'\|^2 \geq 0 \quad \text{and} \quad N_P > 0$$

- Thus the total error is obviously minimized with  $t' = 0$ .
- Therefore the translational offset can be found as follows:

$$\begin{aligned} \because t' = t - \mu_Y + sR(\mu_P) &= 0 \\ \therefore t &= \mu_Y - sR(\mu_P) \end{aligned} \quad (7)$$

That is, the translation is just the difference of the model points centroid and the scaled and rotated scene points centroid.

We return to this equation to find the translational offset once we have found the scale and rotation.

# Finding the Translation

- At this point, we note that the error term can be written as:

$$e_i = y_i' - sR(p_i')$$

- Since  $t' = 0$ , so the total error to be minimized can be re-written as follows:

$$E = \sum_{i=1}^{N_P} \| y_i' - sR(p_i') \|^2 \quad (8)$$

# Finding the Scale

- Expanding the total error defined in (8), we will get:

$$\begin{aligned} E &= \sum_{i=1}^{N_P} \left[ \|y_i'\|^2 - 2y_i' s R(p_i') + \|s R(p_i')\|^2 \right] \\ &= \sum_{i=1}^{N_P} \|y_i'\|^2 - 2s \sum_{i=1}^{N_P} y_i' R(p_i') + s^2 \sum_{i=1}^{N_P} \|R(p_i')\|^2 \end{aligned}$$

- Since rotation preserves length, i.e.

$$\|R(p_i')\|^2 = \|p_i'\|^2$$

- Therefore we obtain:

$$E = \sum_{i=1}^{N_P} \|y_i'\|^2 - 2s \sum_{i=1}^{N_P} y_i' R(p_i') + s^2 \sum_{i=1}^{N_P} \|p_i'\|^2$$

- Let

$$S_y = \sum_{i=1}^{N_P} \|y_i'\|^2, \quad S_P = \sum_{i=1}^{N_P} \|p_i'\|^2 \quad \text{and} \quad \mathbf{D} = \sum_{i=1}^{N_P} y_i' R(p_i')$$

# Finding the Scale

- Hence;

$$E = S_y - 2sD + s^2 S_P \quad (9)$$

where  $S_y$  and  $S_P$  are the sums of the squares of the points length relative to their centroids, while  $D$  is the sum of the dot products of the corresponding points in the model with the rotated points in the scene.

- Since we are looking for the scaling factor  $s$ , complete the square in (9) with respect to  $s$ , we will get:

$$E = \left( s\sqrt{S_P} - \frac{D}{\sqrt{S_P}} \right)^2 + \frac{S_y S_P - D^2}{S_P} \quad (10)$$

Lets prove (10) ☹...



# Finding the Scale

## Proof:

**Recall :**  $(s - A)^2 = s^2 - 2As + A^2$

**To solve for  $s$ , equate the total error to zero.**

$$E = S_P s^2 - 2Ds + S_Y = 0$$

**Move the constant term to the other side :**

$$S_P s^2 - 2Ds = -S_Y$$

**Divide by the coefficient of  $s^2$  :**

$$s^2 - \frac{2Ds}{S_P} = \frac{-S_Y}{S_P}$$

**Take half of the coefficient of the  $s$  - term and square it :**

$$-\frac{D}{S_P} \longrightarrow \left( \frac{D^2}{S_P^2} \right)$$

**Add this to both sides :**

$$s^2 - \frac{2Ds}{S_P} + \left( \frac{D^2}{S_P^2} \right) = \frac{-S_Y}{S_P} + \left( \frac{D^2}{S_P^2} \right)$$

# Finding the Scale

## Proof: cont

*Convert the left hand side to the squared form and simplify the right hand side :*

$$\left(s - \frac{D}{S_P}\right)^2 = \frac{1}{S_P} \left(\frac{D^2}{S_P} - S_Y\right)$$

*Multiply both sides by  $S_P$*

$$S_P \left(s - \frac{D}{S_P}\right)^2 = \left(\frac{D^2}{S_P} - S_Y\right)$$

*Since  $S_P = \sum_{i=1}^{N_P} \|p_i\|^2 \geq 0$ , Hence, we can take the square root*

$$\left(\sqrt{S_P}\right)^2 \left(s - \frac{D}{S_P}\right)^2 - \left(\frac{D^2}{S_P} - S_Y\right) = 0$$

$$\left(s\sqrt{S_P} - \frac{D\sqrt{S_P}}{S_P}\right)^2 + S_Y - \frac{D^2}{S_P} = 0$$

$$\text{Therefore : } E = \left(s\sqrt{S_P} - \frac{D\sqrt{S_P}}{S_P}\right)^2 + \frac{S_Y S_P - D^2}{S_P}$$



# Finding the Scale

- The total error can be minimized with respect to the scaling factor  $s$  when the first term in (10) is set to zero, since the second term doesn't depend on  $s$ .

$$\textit{Therefore} : \left( s\sqrt{S_P} - \frac{D}{\sqrt{S_P}} \right)^2 = 0,$$

$$s\sqrt{S_P} - \frac{D}{\sqrt{S_P}} = 0 \textit{ thus } \sqrt{S_P} = \frac{D}{\sqrt{S_P}}$$

$$\textit{Hence} : s = \frac{\sum_{i=1}^{N_P} y_i' R(p_i')}{\sum_{i=1}^{N_P} \|p_i'\|^2} \quad (11)$$

# Symmetry in Scale

- If we exchange the roles of the scene and the model points as recommended by Horn[2], we will be finding the best fit instead of to the transformation:

$$y_i = (sR(p_i) + t)$$

- We will find to the inverse transformation:

$$p_i = \left( s^T R^T (y_i) + \bar{t} \right)$$

- The scale factor equation in this case becomes:

$$s = \sqrt{\left( \frac{\sum_{i=1}^{N_P} \| y'_i \|^2}{\sum_{i=1}^{N_P} \| p'_i \|^2} \right)}$$

# Finding the Rotation

- Recall (10):

$$E = \left( s\sqrt{S_P} - \frac{D}{\sqrt{S_P}} \right)^2 + \frac{S_y S_P - D^2}{S_P}$$

- To minimize the error with respect to scaling, we set the first term to zero, since the second term doesn't depend on scaling.

$$\left( s\sqrt{S_P} - \frac{D}{\sqrt{S_P}} \right)^2 = 0$$

- Hence, the error term can now be re-written as:

$$E = \frac{S_y S_P - D^2}{S_P} \quad (12)$$

$$\text{where: } S_y = \sum_{i=1}^{N_P} \|y_i'\|^2, \quad S_P = \sum_{i=1}^{N_P} \|p_i'\|^2 \text{ and } D = \sum_{i=1}^{N_P} y_i' R(p_i')$$

- Hence  $S_y \geq 0$  and  $S_P \geq 0$ , therefore  $S_y S_P \geq 0$  and doesn't depend on the rotation, i.e. constant with respect to  $R(\cdot)$ , and  $D^2 \geq 0$  (self evident) is the only part in the error expression that depends on  $R(\cdot)$ .

# Finding the Rotation

- The total error can thus be re-written as:

$$E = \frac{S_y S_P - D^2}{S_P} = 0$$

- Thus  $E = S_y S_P - D^2$  is needed to be minimized.

- Recall:

$$D = \sum_{i=1}^{N_P} y_i' R(p_i')$$

which is the sum of the dot products of the model points and the rotated scene points.

# Finding the Rotation

- This maximization can be interpreted geometrically as follows:

$$y_i' \cdot R(p_i') = \|y_i'\| \|R(p_i')\| \cos \theta$$

where  $\theta$  is the angle between the model points and the rotated scene points.

- To obtain the optimal rotation,  $\theta$  should be zero, therefore  $\cos \theta = 1$  which is the maximum value obtained by the cosine function.
- Since  $\|y_i'\| \geq 0$  and  $\|R(p_i')\| \geq 0$ , having  $\theta = 0$  will lead to the maximum value of  $y_i' \cdot R(p_i')$ .
- Therefore, maximizing  $D$  implicitly means minimizing the angle between the model points and the rotated scene points.

# Finding the Rotation

## Representation of rotation:

- There are many ways to represent rotation, including Euler angles, axis and angle, orthonormal matrices and Hamilton's quaternion's.
- Orthonormal matrices have been used most often in photogrammetry and robotics. However, there are a number of advantages to the unit-quaternion notation.
- Also, unit-quaternions are closely allied to the geometrically intuitive axis and angle notation.
- Here we solve the problem of finding the rotation that maximizes  $D = \sum_{i=1}^{N_p} y_i' R(p_i')$  by using unit-quaternions



# Finding the Rotation

- We have to find the unit quaternion  $\mathbf{q}$  that maximizes:

$$\begin{aligned}
 D &= \sum_{i=1}^{N_p} y'_i \cdot R(p'_i) = \sum_{i=1}^{N_p} y'_i \cdot (\mathbf{q} p'_i \mathbf{q}^*) \\
 &= \sum_{i=1}^{N_p} (\mathbf{q} p'_i \mathbf{q}^*) \cdot y'_i = \sum_{i=1}^{N_p} (\mathbf{q} p'_i) \cdot (y'_i \mathbf{q})
 \end{aligned}$$

- Supposed that

$$p'_i = [p'_{xi}, p'_{yi}, p'_{zi}]^T \quad \text{and} \quad y'_i = [y'_{xi}, y'_{yi}, y'_{zi}]^T$$

- Then

$$\mathbf{q} p'_i = \overline{\mathbf{P}_i} \mathbf{q}$$

$$= \begin{bmatrix} 0 & -p'_{xi} & -p'_{yi} & -p'_{zi} \\ p'_{xi} & 0 & p'_{zi} & -p'_{yi} \\ p'_{yi} & -p'_{zi} & 0 & p'_{xi} \\ p'_{zi} & p'_{yi} & -p'_{xi} & 0 \end{bmatrix} \mathbf{q}$$

while

$$p'_i \mathbf{q} = \mathbf{Y}_i \mathbf{q}$$

$$= \begin{bmatrix} 0 & -y'_{xi} & -y'_{yi} & -y'_{zi} \\ y'_{xi} & 0 & -y'_{zi} & y'_{yi} \\ y'_{yi} & y'_{zi} & 0 & -y'_{xi} \\ y'_{zi} & -y'_{yi} & y'_{xi} & 0 \end{bmatrix} \mathbf{q}$$

# Finding the Rotation

- Note that  $\overline{\mathbf{P}}_i$  and  $\mathbf{Y}_i$  are skew symmetric as well as orthogonal since they are associated to purely imaginary quaternions.
- The sum that we have to maximize can now be written as:

$$\begin{aligned} D &= \sum_{i=1}^{N_p} (\mathbf{q} \mathbf{P}'_i) (\mathbf{y}'_i \mathbf{q}) = \sum_{i=1}^{N_p} (\overline{\mathbf{P}}_i \mathbf{q}) (\mathbf{Y}_i \mathbf{q}) = \sum_{i=1}^{N_p} (\overline{\mathbf{P}}_i \mathbf{q})^T (\mathbf{Y}_i \mathbf{q}) \\ &= \sum_{i=1}^{N_p} \mathbf{q}^T \overline{\mathbf{P}}_i^T \mathbf{Y}_i \mathbf{q} = \mathbf{q}^T \left( \sum_{i=1}^{N_p} \overline{\mathbf{P}}_i^T \mathbf{Y}_i \right) \mathbf{q} = \mathbf{q}^T \left( \sum_{i=1}^{N_p} \mathbf{N}_i \right) \mathbf{q} \\ &= \mathbf{q}^T \mathbf{N} \mathbf{q} \end{aligned}$$

where  $\mathbf{N}_i = \overline{\mathbf{P}}_i^T \mathbf{Y}_i$  and  $\mathbf{N} = \sum_{i=1}^{N_p} \mathbf{N}_i$

# Finding the Rotation

$$\mathbf{N}_i = \bar{\mathbf{P}}_i^T \mathbf{Y}_i$$

$$= \begin{bmatrix} 0 & p'_{xi} & p'_{yi} & p'_{zi} \\ -p'_{xi} & 0 & -p'_{zi} & p'_{yi} \\ -p'_{yi} & p'_{zi} & 0 & -p'_{xi} \\ -p'_{zi} & -p'_{yi} & p'_{xi} & 0 \end{bmatrix} \begin{bmatrix} 0 & -y'_{xi} & -y'_{yi} & -y'_{zi} \\ y'_{xi} & 0 & -y'_{zi} & y'_{yi} \\ y'_{yi} & y'_{zi} & 0 & -y'_{xi} \\ y'_{zi} & -y'_{yi} & y'_{xi} & 0 \end{bmatrix}$$

$$= \begin{bmatrix} p'_{xi}y'_{xi} + p'_{yi}y'_{yi} + p'_{zi}y'_{zi} & p'_{yi}y'_{zi} - p'_{zi}y'_{yi} & -p'_{xi}y'_{zi} + p'_{zi}y'_{xi} & p'_{xi}y'_{yi} - p'_{yi}y'_{xi} \\ -p'_{zi}y'_{yi} + p'_{yi}y'_{zi} & p'_{xi}y'_{xi} - p'_{zi}y'_{zi} - p'_{yi}y'_{yi} & p'_{xi}y'_{yi} + p'_{yi}y'_{xi} & p'_{xi}y'_{zi} + p'_{zi}y'_{xi} \\ p'_{zi}y'_{xi} - p'_{xi}y'_{zi} & p'_{yi}y'_{xi} + p'_{xi}y'_{yi} & p'_{yi}y'_{yi} - p'_{zi}y'_{zi} - p'_{xi}y'_{xi} & p'_{yi}y'_{zi} + p'_{zi}y'_{yi} \\ -p'_{yi}y'_{xi} + p'_{xi}y'_{yi} & p'_{zi}y'_{xi} + p'_{xi}y'_{zi} & p'_{zi}y'_{yi} + p'_{yi}y'_{zi} & p'_{zi}y'_{zi} - p'_{yi}y'_{yi} - p'_{xi}y'_{xi} \end{bmatrix}$$

# Finding the Rotation

$$\mathbf{N} = \sum_{i=1}^{N_p} \mathbf{N}_i$$

$$= \begin{bmatrix} S_{xx} + S_{yy} + S_{zz} & S_{yz} - S_{zy} & -S_{xz} + S_{zx} & S_{xy} - S_{yz} \\ -S_{zy} + S_{yz} & S_{xx} - S_{zz} - S_{yy} & S_{xy} + S_{yx} & S_{xz} + S_{zx} \\ S_{zx} - S_{xz} & S_{yx} + S_{xy} & S_{yy} - S_{zz} - S_{xx} & S_{yz} + S_{zy} \\ -S_{yx} + S_{xy} & S_{zx} + S_{xz} & S_{zy} + S_{yz} & S_{zz} - S_{yy} - S_{xx} \end{bmatrix}$$

where  $S_{xx} = \sum_{i=1}^{N_p} p'_{xi} y'_{xi}$ ,  $S_{xy} = \sum_{i=1}^{N_p} p'_{xi} y'_{yi}$ ,  $S_{xz} = \sum_{i=1}^{N_p} p'_{xi} y'_{zi} \dots$  etc

# Finding the Rotation

- Hence we can define the S-matrix whose elements are sums of products of coordinates measured in the scene shape with coordinates measured in the model shape, such that:

$$S = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix}$$

- Note that the scene and model points were brought to the origin by subtracting their centroids.

# Finding the Rotation

## Eigenvector Maximizes Matrix Product:

- It can be shown that the unit quaternion that maximizes  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  is the eigenvector corresponding to the most positive eigenvalue of the matrix  $\mathbf{N}$ .

## Proof:

- To find the rotation that minimizes the sum of squares of errors, we have to find the quaternion  $\mathbf{q}$  that maximizes  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  subject to the constraint that  $\mathbf{q} \cdot \mathbf{q} = 1$  (unit quaternion).
- The symmetric 4x4 matrix  $\mathbf{N}$  will have four real eigenvalues, say  $\lambda_1, \lambda_2, \lambda_3$  and  $\lambda_4$ .
- A corresponding set of orthogonal unit eigenvectors  $v_1, v_2, v_3$  and  $v_4$  can be constructed such that  $\mathbf{N}v_i = \lambda_i v_i$  where  $i = 1, 2, 3, 4$ .

# Finding the Rotation

## Proof: cont

- The eigenvectors span the 4D space, so an arbitrary quaternion  $\mathbf{q}$  can be written as a linear combination in the form:

$$\mathbf{q} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4$$

- Since the eigenvectors are orthogonal, we have:

$$\mathbf{q} \cdot \mathbf{q} = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2$$

- We know that this has to be equal to one since we are looking for a unit quaternion.

- Now  $\mathbf{Nq} = \lambda \mathbf{q} = \alpha_1 \lambda_1 \mathbf{v}_1 + \alpha_2 \lambda_2 \mathbf{v}_2 + \alpha_3 \lambda_3 \mathbf{v}_3 + \alpha_4 \lambda_4 \mathbf{v}_4$

- We conclude that

$$\mathbf{q}^T \mathbf{Nq} = \mathbf{q} \cdot (\mathbf{Nq}) = \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \alpha_4^2 \lambda_4$$

# Finding the Rotation

## Proof: cont

- Now, suppose we have arranged the eigenvalues in order such that  $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$ .
- Then we have:

$$\begin{aligned} \mathbf{q}^T \mathbf{N} \mathbf{q} &= \mathbf{q} \cdot (\mathbf{N} \mathbf{q}) = \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \alpha_4^2 \lambda_4 \\ &\leq \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_1 + \alpha_3^2 \lambda_1 + \alpha_4^2 \lambda_1 \\ &= (\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2) \lambda_1 \\ &= (\mathbf{q} \cdot \mathbf{q}) \lambda_1 \\ &= \lambda_1 \end{aligned}$$



# Finding the Rotation

## Proof: cont

- Since we need to maximize  $\mathbf{q}^T \mathbf{N} \mathbf{q}$ , but we have  $\mathbf{q}^T \mathbf{N} \mathbf{q} \leq \lambda_1$ , hence  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  is bounded above by  $\lambda_1$  which is the largest eigenvalue.
- Thus the maximum of  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  is attained when  $\mathbf{q}^T \mathbf{N} \mathbf{q} = \lambda_1$ .
- This only happens if  $\alpha_1 = 1$  and  $\alpha_2 = \alpha_3 = \alpha_4 = 0$ , i.e. :

$$\begin{aligned}\mathbf{q} &= \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4 \\ &= \mathbf{v}_1\end{aligned}$$

which is the eigenvector corresponding to the largest positive eigenvalue  $\lambda_1$ .



# Finding Alignment - Summary

- We can now summarize the algorithm of finding the rotation, scaling factor and the translational offset given pairs of points correspondences as follows:

1. Find the centroids of the two point sets in the scene and model shapes.

$$\mu_P = \frac{1}{N_P} \sum_{i=1}^{N_P} p_i \quad \text{and} \quad \mu_Y = \frac{1}{N_P} \sum_{i=1}^{N_P} y_i$$

2. Compute the points coordinates relative to their centroids.

$$p_i' = p_i - \mu_p \quad \text{and} \quad y_i' = y_i - \mu_Y$$

3. For each pair of points  $\{p_i', y_i'\}$ , compute the nine possible products of the two vectors  $p_i'$  and  $y_i'$ . Then add them up to obtain  $S_{xx}$ ,  $S_{xy}$ , ...  $S_{zz}$ . These nine totals contain all the information needed to find the solution.

$$\text{where } S_{xx} = \sum_{i=1}^{N_p} p_{xi}' y_{xi}', \quad S_{xy} = \sum_{i=1}^{N_p} p_{xi}' y_{yi}', \quad S_{xz} = \sum_{i=1}^{N_p} p_{xi}' y_{zi}' \dots \text{ etc}$$

# Finding Alignment - Summary

4. Compute the ten independent elements of the 4x4 symmetric matrix  $\mathbf{N}$  by combining the sums obtained in (3) as follows:

$$\mathbf{N} = \begin{bmatrix} S_{xx} + S_{yy} + S_{zz} & S_{yz} - S_{zy} & -S_{xz} + S_{zx} & S_{xy} - S_{yx} \\ -S_{zy} + S_{yz} & S_{xx} - S_{zz} - S_{yy} & S_{xy} + S_{yx} & S_{xz} + S_{zx} \\ S_{zx} - S_{xz} & S_{yx} + S_{xy} & S_{yy} - S_{zz} - S_{xx} & S_{yz} + S_{zy} \\ -S_{yx} + S_{xy} & S_{zx} + S_{xz} & S_{zy} + S_{yz} & S_{zz} - S_{yy} - S_{xx} \end{bmatrix}$$

5. Find the eigenvalues and eigenvectors of  $\mathbf{N}$ .
6. The quaternion  $\mathbf{q}$  representing the rotation is the eigenvector corresponding to the largest positive eigenvalue of  $\mathbf{N}$ .

# Finding Alignment - Summary

7. Compute the scaling factor as follows:

$$s = \frac{\sum_{i=1}^{N_p} y_i' (\mathbf{q} p_i' \mathbf{q}^*)}{\sum_{i=1}^{N_p} \|p_i'\|^2}$$

where  $\mathbf{q} p_i' \mathbf{q}^* = \left( \overline{\mathbf{Q}}^T \mathbf{Q} \right) p_i'$

$$\overline{\mathbf{Q}}^T \mathbf{Q} = \begin{bmatrix} \mathbf{q} \cdot \mathbf{q} & 0 & 0 & 0 \\ 0 & (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix}$$

where  $\mathbf{q} \cdot \mathbf{q} = q_0^2 + q_x^2 + q_y^2 + q_z^2$

8. Compute the translational offset as follows:

$$t = \mu_Y - s(\mathbf{q} \mu_P \mathbf{q}^*) = \mu_Y - s \left( \overline{\mathbf{Q}}^T \mathbf{Q} \right) \mu_P$$

Let's do it ...

## Iterative Closest Point Algorithm

- Initialization
- Find correspondences
- Find alignment
- Apply alignment
- Compute residual error



# Function Prototype

```
function [s, R, t, err,newP] = icp(M,P)
% ICP Iterative Closest Point Algorithm.
%
% [s,R,t,err] = icp(M,P)
%
% ICP fit scene points P to the model points M.
% Fit with respect to minimize the sum of square
% errors with the closest model points and scene points.
%
% Parameters:   M           3xNm matrix representing the Nm model 3D points
%              P           3xNp matrix representing the Np scene 3D points
%
% Return:      s           The scaling factor (uniform scaling)
%              R           The 3x3 rotation matrix
%              t           The 3x1 translation vector
%              err         Residual error defined as the mean square error
%                          between the model points and the transformed
%                          scene points.
%              newP        3xNp matrix representing the Np scene 3D points
%                          after applying the estimated rigid registration
%                          parameters (s,R,t), where newP = s*R*P + t
```

# Initializations ...

```
%% Initialization
% initiate a starting scaling factor, rotation matrix and starting
% translational offset
s = 1;
R = eye(size(M,1));
t = zeros(size(M,1),1);
newP = P;

% setting algorithm parameters
max_iter = 200;      % max number of icp iterations
thresh    = 1e-5;   % threshold to icp iterations

% number of points
Np = size(P,2);
Nm = size(M,2);
dim = size(P,1);
```

# ICP Loop: Finding Correspondences

```
%% start ICP loop
for iter = 1 : max_iter
    % finding correspondences
    % for each point in the scene points set P we want to get the closest
    % model point in the model points set M
    Y = zeros(dim,Np);    % set of closest points
    for i = 1 : Np
        % current point
        pi = newP(:,i);
        % get the distance to all model points
        d = zeros(1,Nm);
        for k = 1 : Nm
            mk = M(:,k);
            d(k) = sqrt(sum((pi - mk).^2)); % euclidean distance
        end
        % the closest point will be ...
        [minD,j] = min(d);
        Y(:,i) = M(:,j);
    end
end
```

$$d(p_i, m_k) = \sqrt{(p_{xi} - m_{xk})^2 + (p_{yi} - m_{yk})^2 + (p_{zi} - m_{zk})^2}$$

$$j = \arg \min_{k=1, \dots, N_M} d(p_i, m_k)$$

Loop to be continued ...



# ICP Loop: Finding/Applying Alignment

Explanation to follow

```
% finding alignment
```

```
[s, R, t, err] = find_alignment(newP, Y);
```

```
% apply alignment and compute residual error
```

```
for i = 1:Np
```

```
    newP(:,i) = s*R*newP(:,i) + t;
```

```
    e = Y(:,i) - newP(:,i);
```

```
    err = err + e'*e;
```

```
end
```


```
err = err/Np;
```

```
if err < thresh
```

```
    break;
```

```
end
```

```
end
```


$$e_i = y_i - (s\mathbf{R}p_i + t) \quad i = 1, 2, \dots, N_P$$
$$E = \sum_{i=1}^{N_P} \|e_i\|^2$$

Let's do it ...

## Finding Alignment

- Zero mean point sets.
- Quaternion computation.
- Rotation matrix computation.
- Scaling factor computation.
- Translational offset computation.
- Residual error computation.



# Function Prototype

```
function [s, R, t, err] = find_alignment(P,Y)

% Computes the scaling factor, rotation and translational offset factor)
% for the transformation between two corresponding 3D point sets Pi
% and Yi such as they are related by:
%
%      $Y_i = sR \cdot P_i + t$ 
%
% Parameters:   P           3xN matrix representing the N scene 3D points
%               Y           3xN matrix representing the N model 3D points
%               which correspond to the scene points P
%
% Return:      s           The scaling factor (uniform scaling)
%               R           The 3x3 rotation matrix
%               t           The 3x1 translation vector
%               err        Residual error defined as  $err = \sum(Y_i - (sR \cdot P_i + t))$ 
%
% Notes: Minimum 3D point number is  $N > 4$ 
```

# Test Given Point Sets

```
%% Test the size of point sets
[dim_p Np] = size(P);
[dim_y Ny] = size(Y);

if(Np ~= Ny)
    error('Point sets need to have same number of points.');
```

end

```
if(dim_p ~= 3 || dim_y ~= 3)
    error('Need points of dimension 3');
```

end

```
if(Np<4)
    error('Need at least 4 point pairs');
```

end

```
%Number of points
N = Np;
```

# Zero Mean Point Sets

```
%% Compute the centroid of each point set
```

```
Mu_p = mean(P,2);
```

$$\mu_P = \frac{1}{N_P} \sum_{i=1}^{N_P} P_i$$

```
Mu_y = mean(Y,2);
```

$$\mu_Y = \frac{1}{N_P} \sum_{i=1}^{N_P} y_i$$

```
% Remove the centroid: points measured relative to their centroids
```

```
Pprime = P - repmat(Mu_p,1,N);
```

$$p_i' = p_i - \mu_p$$

```
Yprime = Y - repmat(Mu_y,1,N);
```

$$y_i' = y_i - \mu_Y$$

# Quaternion Computation

```
%% Compute the optimal quaternion
% matrix of sums of products of the points,
Px = Pprime(1,:);      Yx = Yprime(1,:);
Py = Pprime(2,:);      Yy = Yprime(2,:);
Pz = Pprime(3,:);      Yz = Yprime(3,:);

Sxx = sum(Px.*Yx);
Sxy = sum(Px.*Yy);
Sxz = sum(Px.*Yz);

Syx = sum(Py.*Yx);
Syy = sum(Py.*Yy);
Syz = sum(Py.*Yz);

Szx = sum(Pz.*Yx);
Szy = sum(Pz.*Yy);
Szz = sum(Pz.*Yz);

Nmatrix = [ Sxx + Syy + Szz      Syz-Szy      -Sxz + Szx      Sxy - Syx;
            -Szy + Syz      Sxx - Szz - Syy      Sxy + Syx      Sxz + Szx;
            Sxz - Sxz      Syx + Sxy      Syy - Szz - Sxx      Syz + Szy;
            -Syx + Sxy      Szx + Sxz      Szy + Syz      Szz - Syy - Sxx];

% Compute eigenvalues
[V,D] = eig(Nmatrix);

% the optimal quaternion is the one corresponding to the largest positive
% eigen value which is D(4,4).
q = V(:,4);
```

# Rotation Matrix Computation

```
%% Compute the rotation matrix
% individual components
q0 = q(1); q1 = q(2); q2 = q(3); q3 = q(4);

% matrices associated to the found quaternion
Qbar = [q0 -q1 -q2 -q3 ;
        q1  q0  q3 -q2 ;
        q2 -q3  q0  q1 ;
        q3  q2 -q1  q0];

Q = [q0 -q1 -q2 -q3 ;
     q1  q0 -q3  q2 ;
     q2  q3  q0 -q1 ;
     q3 -q2  q1  q0];

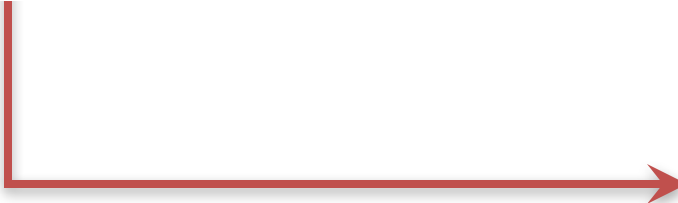
% The rotation matrix will be:
R = Qbar'*Q;
% Retrieve the 3x3 rotation matrix
R = R(2:4,2:4);
```

$$\bar{Q} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & q_z & -q_y \\ q_y & -q_z & q_0 & q_x \\ q_z & q_y & -q_x & q_0 \end{bmatrix}$$

$$Q = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix}$$

# Scaling Factor Computation

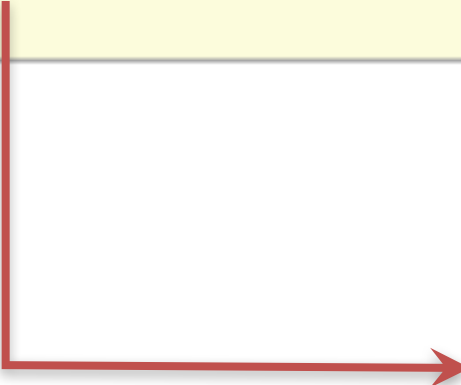
```
%% Compute the scaling factor  
Sp = 0;  
D = 0;  
for i=1:N  
    D = D + Yprime(:,i)' * Yprime(:,i) ;  
    Sp = Sp + Pprime(:,i)' * Pprime(:,i);  
end  
s = sqrt(D/Sp);
```


$$s = \sqrt{\frac{\sum_{i=1}^{N_P} \|y'_i\|^2}{\sum_{i=1}^{N_P} \|p'_i\|^2}}$$




# Translation Offset Computation

```
%% Compute the translational offset  
t = Mu_y - s*R*Mu_p;
```


$$t = \mu_Y - s \left( \overline{\mathbf{Q}}^T \mathbf{Q} \right) \mu_P = \mu_Y - s \mathbf{R} \mu_P$$

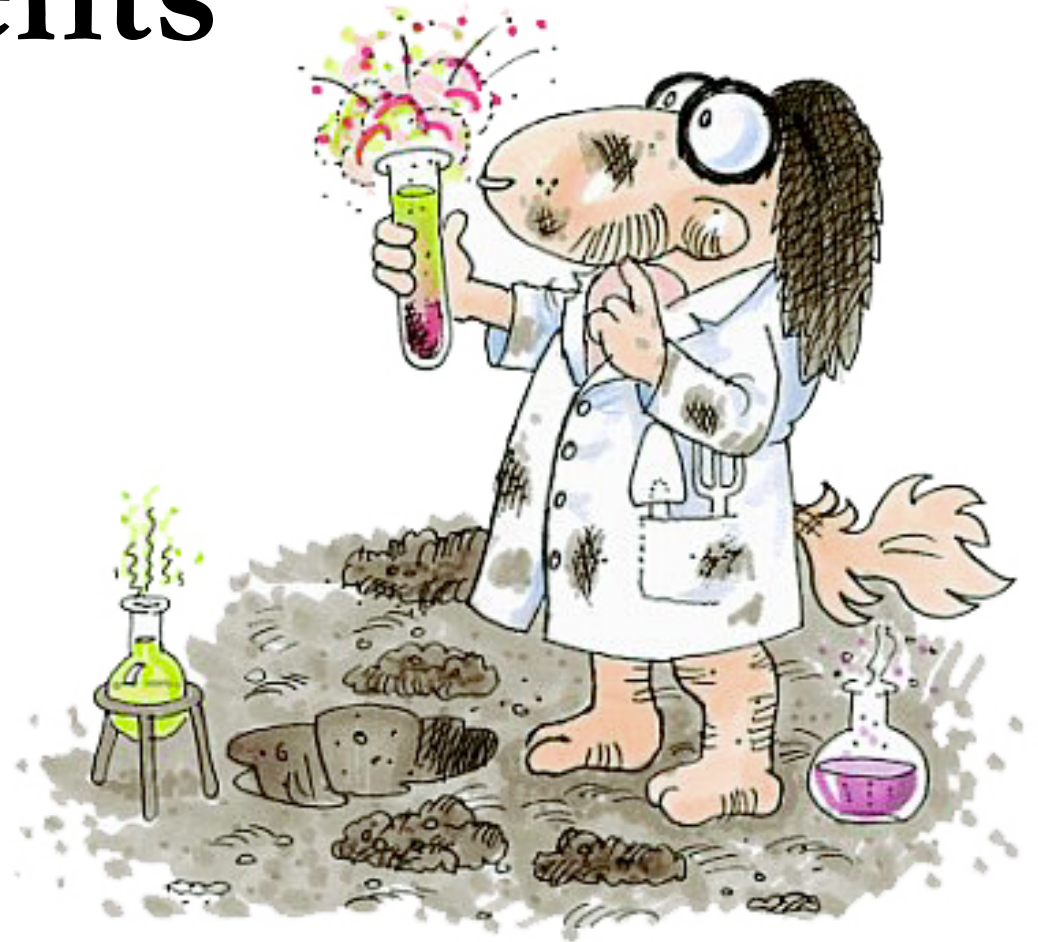
# Residual Error

```
%% Compute the residual error
err = 0;
for i = 1:N
    d = (Y(:,i) - (s*R*P(:,i) + t));
    err = err + d'*d;
end
```

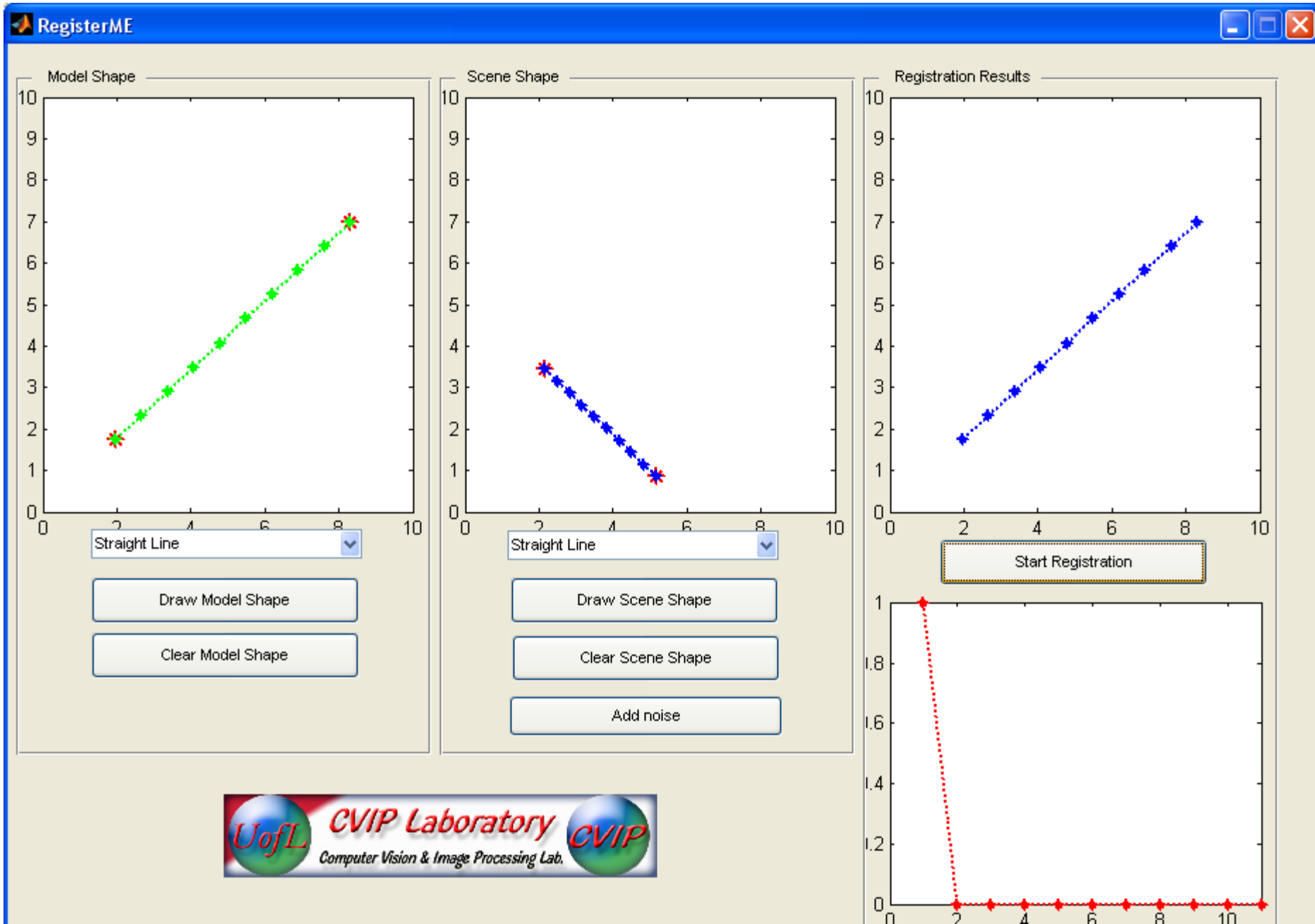

$$e_i = y_i - (s\mathbf{R}p_i + t) \quad i = 1, 2, \dots, N_P$$

$$E = \sum_{i=1}^{N_P} \|e_i\|^2$$

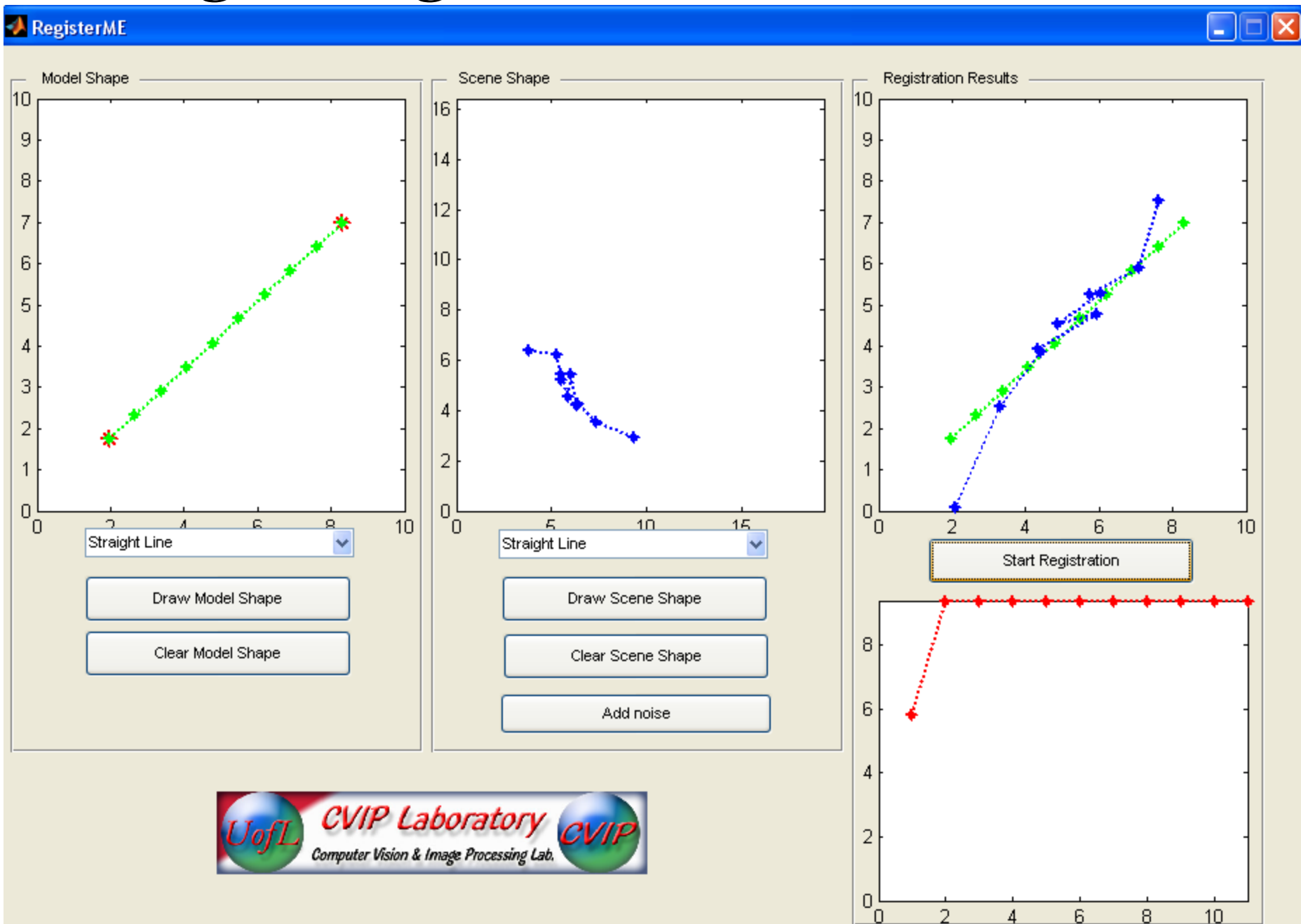
# Experiments



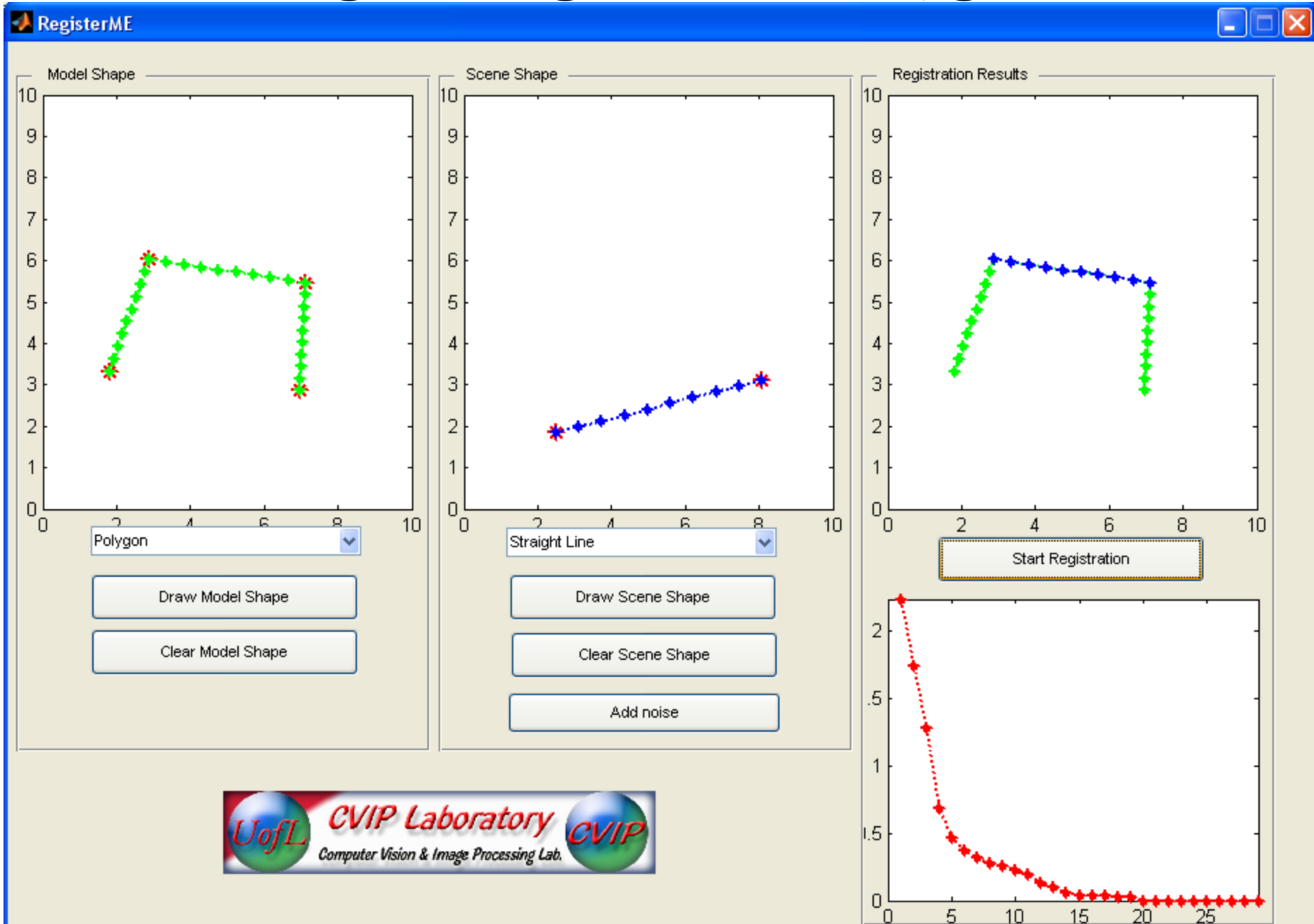
# Registering Line to Line



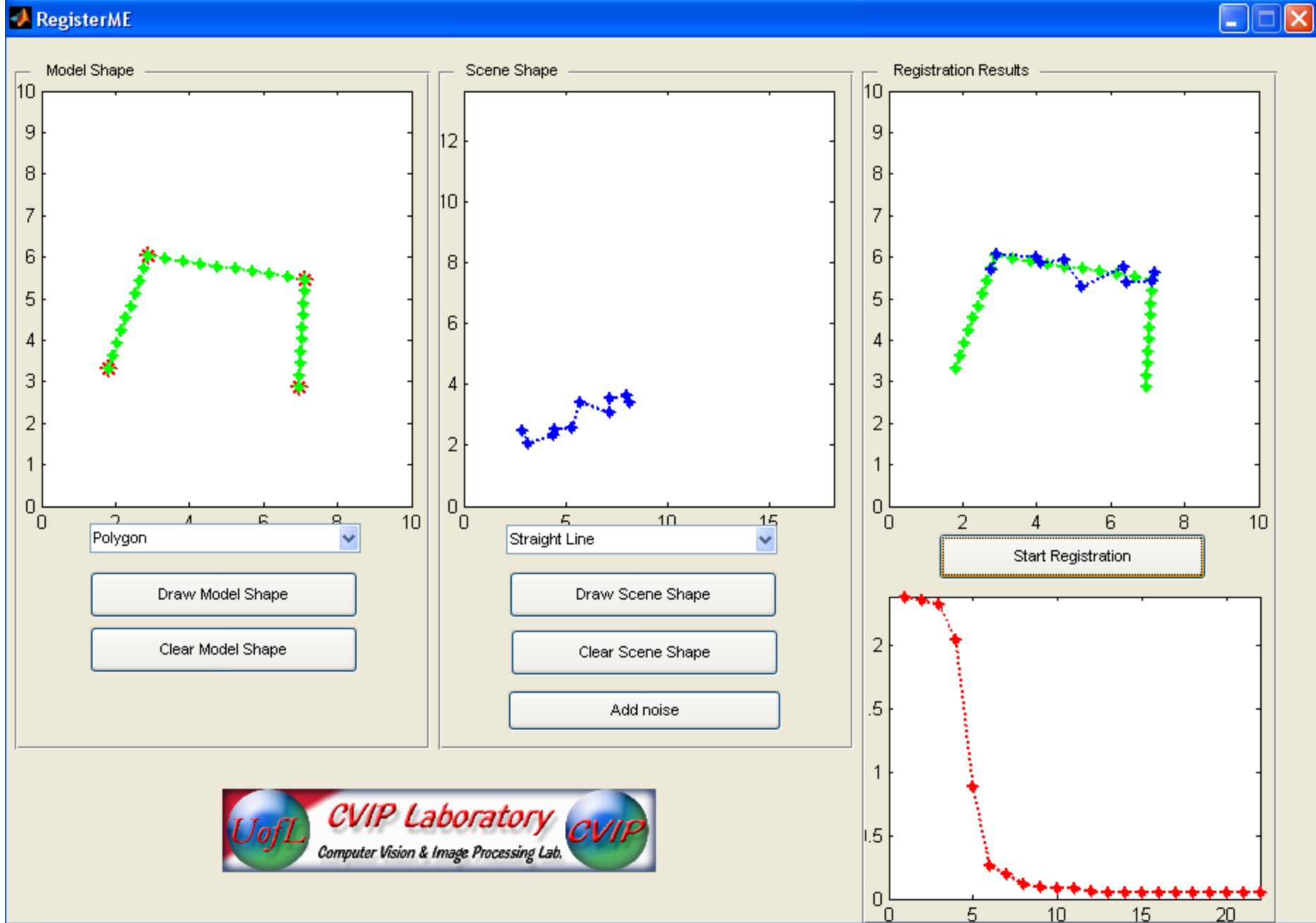
# Registering Line to Line – with noise



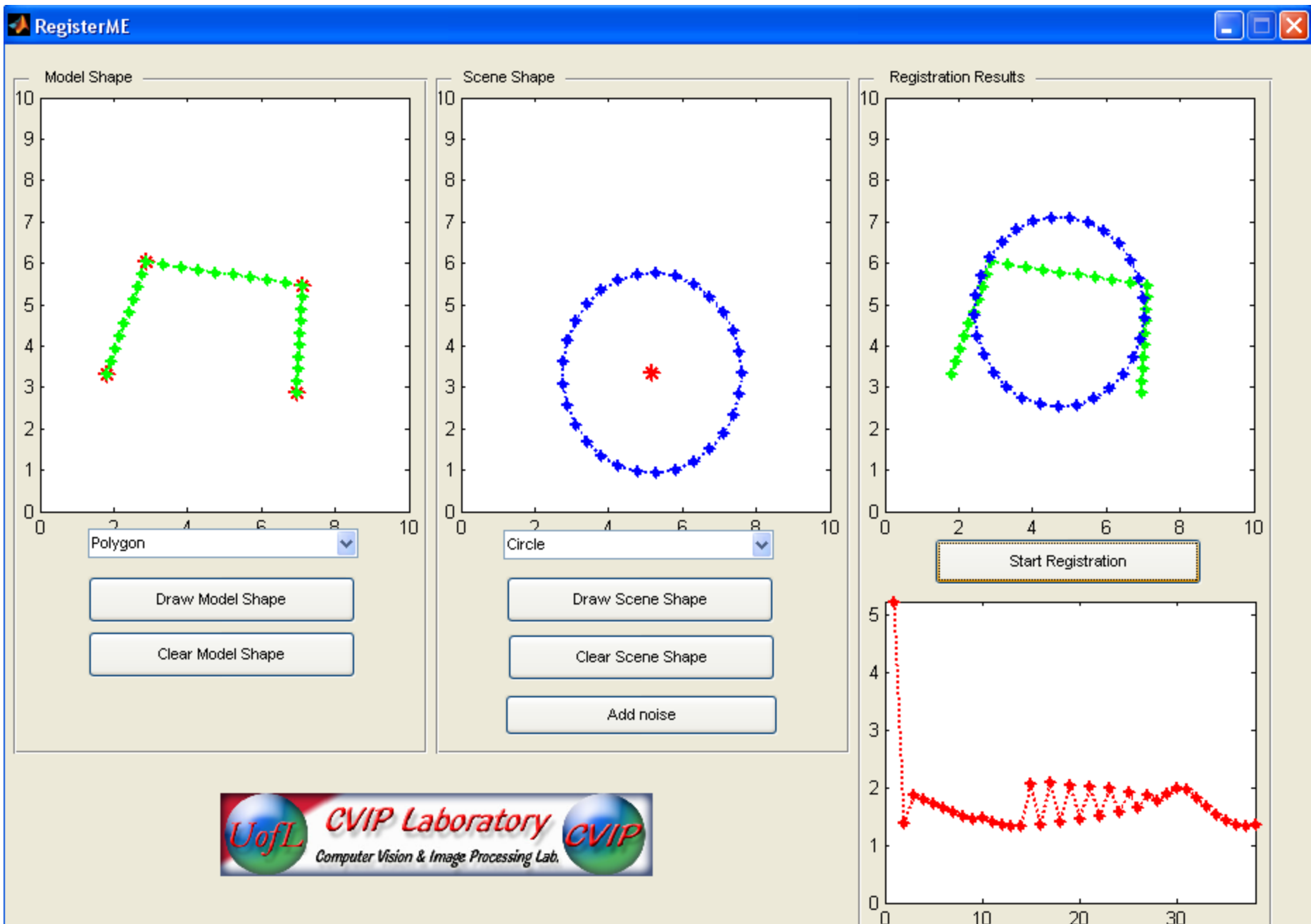
# Registering Line to Polygon



# Registering Line to Polygon – with noise

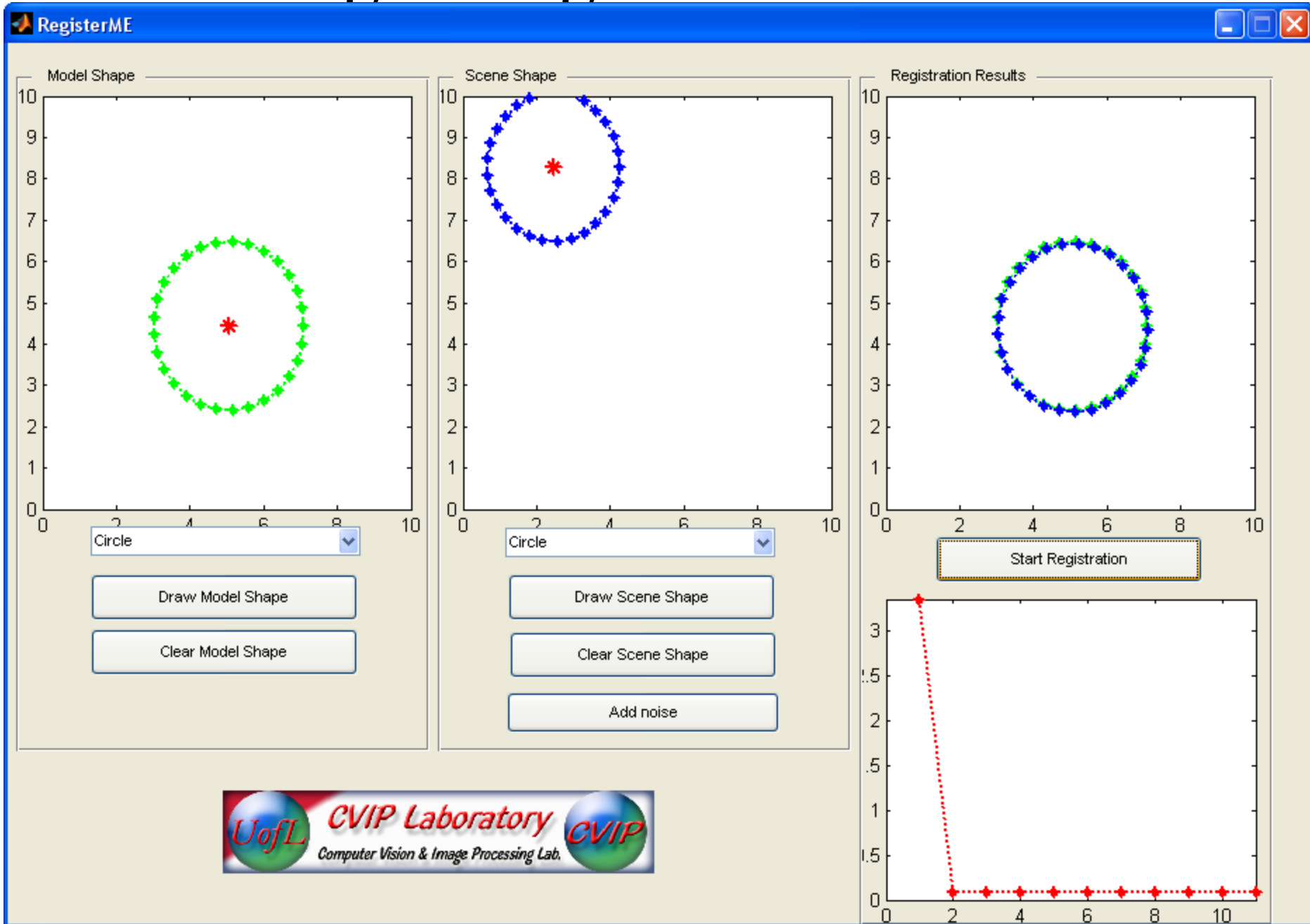


# Registering Circle to Polygon

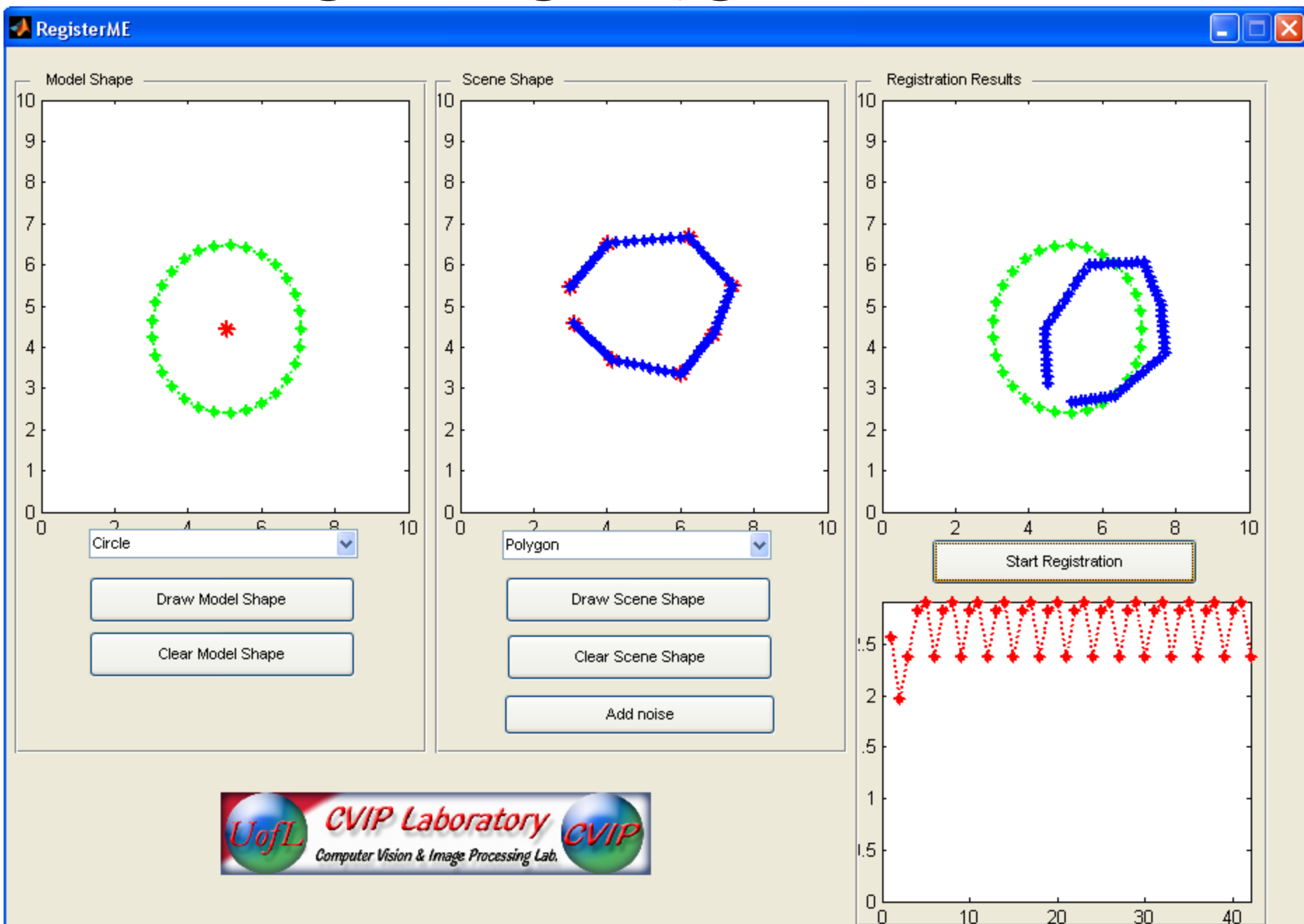




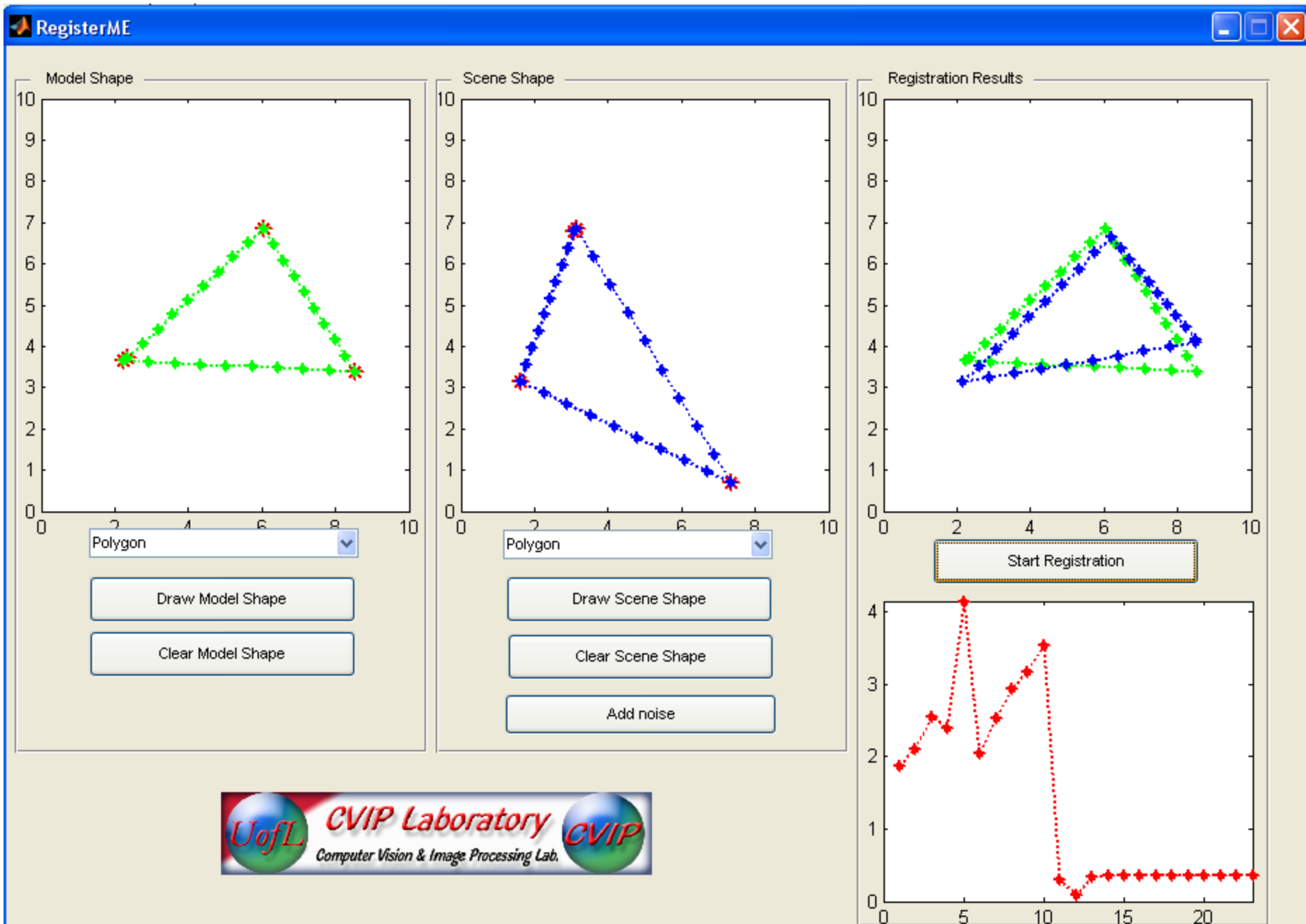
# Registering Circle to Circle



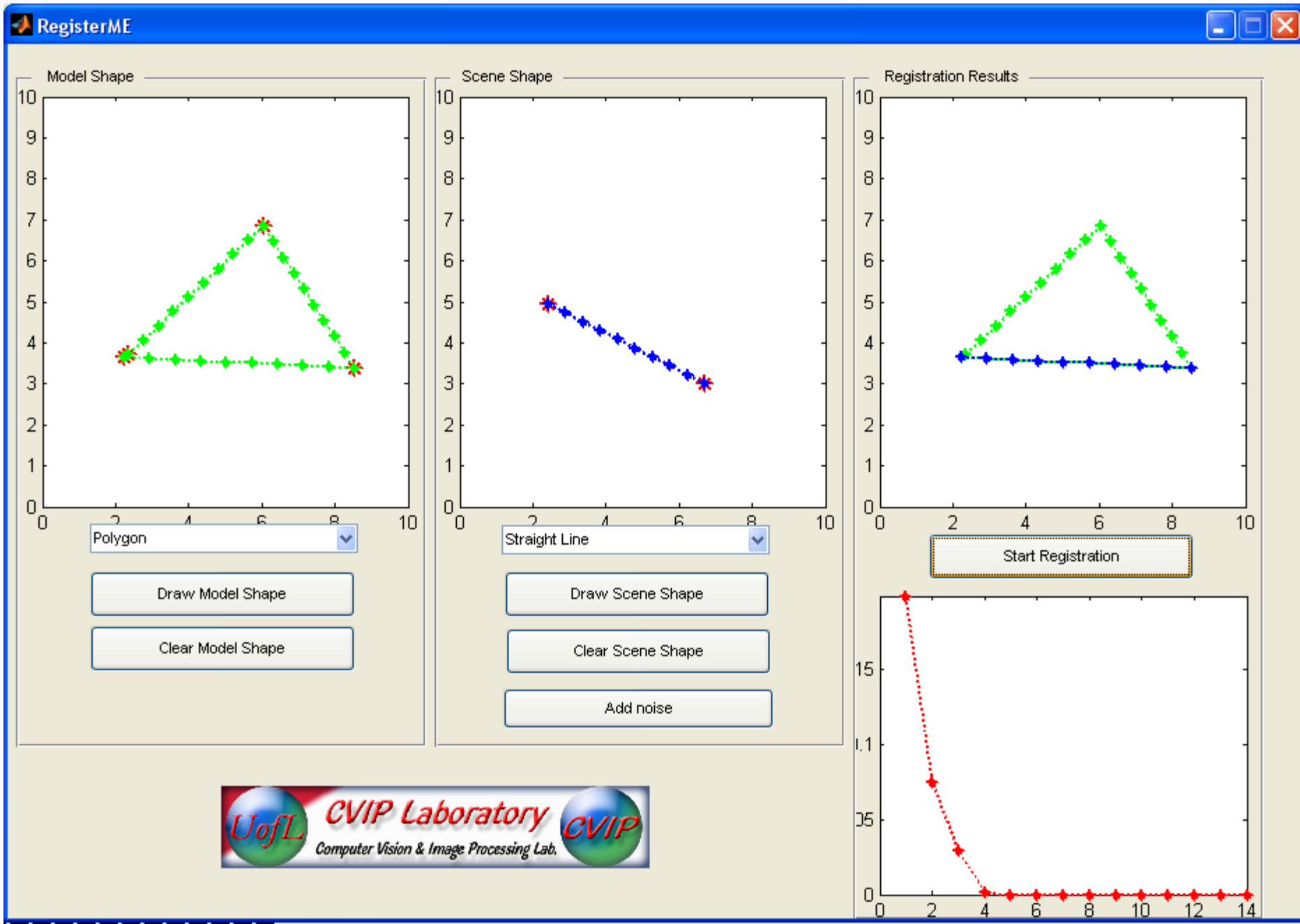
# Registering Polygon to Circle



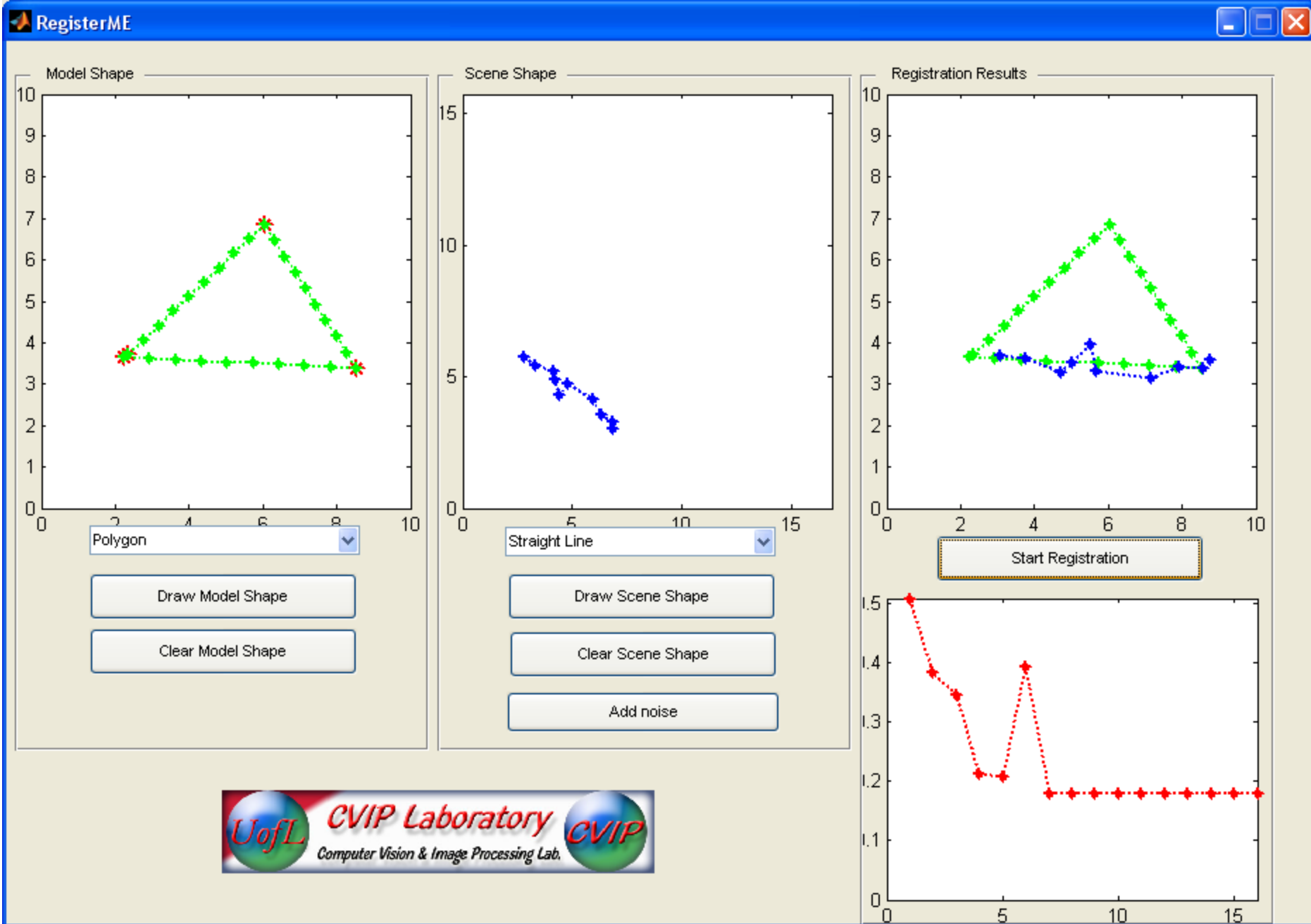
# Registering Triangle to Triangle



# Registering Line to Triangle



# Registering Line to Triangle – with noise



# Iterative Closest Point (ICP) Algorithm

Some Proofs ...



# A Tutorial on Rigid Registration

Iterative Closed Point (ICP)

Some Proofs ...

Aly A. Farag

University of Louisville

Acknowledgements:

Help with these slides were provided by Amal Farag and Shireen Elhabian

**Prove that quaternion multiplication preserves dot product, i.e.  $(qv).(qr) = (q.q)(v.r)$ . What happen in case  $q$  is a unit quaternion?**

- Proof:

$$\begin{aligned} (qv).(qr) &= (Qv).(Qr) = (Qv)^T (Qr) \\ &= \mathbf{v}^T \mathbf{Q}^T \mathbf{Q} \mathbf{r} = \mathbf{v}^T (\mathbf{q} \cdot \mathbf{q}) \mathbf{I} \mathbf{r} = (\mathbf{q} \cdot \mathbf{q})(\mathbf{v} \cdot \mathbf{r}) \end{aligned}$$

- In the case of a unit quaternion :  $(qv).(qr) = (v.r)$
- A special case follows immediately:

$$(\mathbf{qr}).(\mathbf{qr}) = (\mathbf{q} \cdot \mathbf{q})(\mathbf{r} \cdot \mathbf{r})$$

that is the magnitude of a product is just the product of the magnitudes.



Prove that the composite product leads to purely imaginary quaternion, hence it can be used to represent rotation, i.e. Rotating the vector (point)  $\mathbf{r}$  by a unit quaternion  $q$  can be defined as  $r' = qrq^*$ , where  $r'$  is a purely imaginary quaternion representing the vector  $\mathbf{r}$  after rotation by  $q$ .

Proof:

- The objective is representing the composite product as a matrix multiplied by a vector  $\mathbf{r}$ . let's find this matrix in terms of the matrices associated to the unit quaternion  $q$  and its conjugate.

$$qrq^* = (\mathbf{Q}\mathbf{r})q^* = \overline{\mathbf{Q}}^T (\mathbf{Q}\mathbf{r}) = (\overline{\mathbf{Q}}^T \mathbf{Q})\mathbf{r}$$

$$\text{where } \mathbf{Q} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} \text{ and } \overline{\mathbf{Q}} = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & q_z & -q_y \\ q_y & -q_z & q_0 & q_x \\ q_z & q_y & -q_x & q_0 \end{bmatrix}$$

Where  $\mathbf{Q}$  and  $\overline{\mathbf{Q}}$  are the 4x4 matrices corresponding to the unit quaternion  $q$ .

Proof: cont

$$\begin{aligned} \therefore \overline{\mathbf{Q}}^T \mathbf{Q} &= \begin{bmatrix} q_0 & q_x & q_y & q_z \\ -q_x & q_0 & -q_z & q_y \\ -q_y & q_z & q_0 & -q_x \\ -q_z & -q_y & q_x & q_0 \end{bmatrix} \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{q} \cdot \mathbf{q} & 0 & 0 & 0 \\ 0 & (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix} \end{aligned}$$

where  $\mathbf{q} \cdot \mathbf{q} = q_0^2 + q_x^2 + q_y^2 + q_z^2$

$\therefore \mathbf{q} \mathbf{r} \mathbf{q}^* = (\overline{\mathbf{Q}}^T \mathbf{Q}) \mathbf{r}$  is a purely imaginary quaternion if  $\mathbf{r}$  is a purely imaginary quaternion.

## Proof: cont

- Since  $\mathbf{q}$  is a quaternion, then  $\mathbf{Q}$  and  $\overline{\mathbf{Q}}$  are orthogonal matrices by definition.
- Since  $\mathbf{q}$  is a unit quaternion, then  $\mathbf{Q}$  and  $\overline{\mathbf{Q}}$  are orthonormal matrices,.
- Hence the lower-right-hand 3x3 sub-matrix of  $\overline{\mathbf{Q}}^T \mathbf{Q}$  must also be orthonormal , hence it is the rotation matrix  $\mathbf{R}$  that take  $\mathbf{r}$  to  $\mathbf{r}'$  such that  $\mathbf{r}' = \mathbf{R}\mathbf{r}$ .
- The expansion of  $\overline{\mathbf{Q}}^T \mathbf{Q}$  provides an explicit method for computing the orthonormal rotation matrix  $\mathbf{R}$  from the components of the unit quaternion  $q$ .



Prove that that the unit quaternion that maximizes  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  is the eigenvector corresponding to the most positive eigenvalue of the matrix  $\mathbf{N}$ .

Proof:

- To find the rotation that minimizes the sum of squares of errors, we have to find the quaternion  $\mathbf{q}$  that maximizes  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  subject to the constraint that  $\mathbf{q} \cdot \mathbf{q} = 1$  (unit quaternion).
- The symmetric 4x4 matrix  $\mathbf{N}$  will have four real eigenvalues, say  $\lambda_1, \lambda_2, \lambda_3$  and  $\lambda_4$ .
- A corresponding set of orthogonal unit eigenvectors  $v_1, v_2, v_3$  and  $v_4$  can be constructed such that  $\mathbf{N}v_i = \lambda_i v_i$  where  $i = 1, 2, 3, 4$ .

## Proof: cont

- The eigenvectors span the 4D space, so an arbitrary quaternion  $\mathbf{q}$  can be written as a linear combination in the form:

$$\mathbf{q} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4$$

- Since the eigenvectors are orthogonal, we have:

$$\mathbf{q} \cdot \mathbf{q} = \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2$$

- We know that this has to be equal to one since we are looking for a unit quaternion.
- Now  $\mathbf{Nq} = \lambda \mathbf{q} = \alpha_1 \lambda_1 \mathbf{v}_1 + \alpha_2 \lambda_2 \mathbf{v}_2 + \alpha_3 \lambda_3 \mathbf{v}_3 + \alpha_4 \lambda_4 \mathbf{v}_4$
- We conclude that

$$\mathbf{q}^T \mathbf{Nq} = \mathbf{q} \cdot (\mathbf{Nq}) = \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \alpha_4^2 \lambda_4$$

## Proof: cont

- Now, suppose we have arranged the eigenvalues in order such that  $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$ .
- Then we have:

$$\begin{aligned} \mathbf{q}^T \mathbf{N} \mathbf{q} &= \mathbf{q} \cdot (\mathbf{N} \mathbf{q}) = \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \alpha_4^2 \lambda_4 \\ &\leq \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_1 + \alpha_3^2 \lambda_1 + \alpha_4^2 \lambda_1 \\ &= (\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2) \lambda_1 \\ &= (\mathbf{q} \cdot \mathbf{q}) \lambda_1 \\ &= \lambda_1 \end{aligned}$$

## Proof: cont

- Since we need to maximize  $\mathbf{q}^T \mathbf{N} \mathbf{q}$ , but we have  $\mathbf{q}^T \mathbf{N} \mathbf{q} \leq \lambda_1$ , hence  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  is bounded above by  $\lambda_1$  which is the largest eigenvalue.
- Thus the maximum of  $\mathbf{q}^T \mathbf{N} \mathbf{q}$  is attained when  $\mathbf{q}^T \mathbf{N} \mathbf{q} = \lambda_1$ .
- This only happens if  $\alpha_1 = 1$  and  $\alpha_2 = \alpha_3 = \alpha_4 = 0$ , i.e. :

$$\begin{aligned}\mathbf{q} &= \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \alpha_4 \mathbf{v}_4 \\ &= \mathbf{v}_1\end{aligned}$$

which is the eigenvector corresponding to the largest positive eigenvalue  $\lambda_1$ .



# References

1. P.J. Besl, N.D. McKay, A method for registration of 3D shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (1992) 239–254.
2. B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," J. Opt. Soc. Amer. A vol. 4, no. 4, pp. 629-642, Apr. 1987.
3. [www.cs.virginia.edu/~gfx/Courses/2004/Intro.Spring.04/Lectures/lecture05.ppt](http://www.cs.virginia.edu/~gfx/Courses/2004/Intro.Spring.04/Lectures/lecture05.ppt)
4. [gmm.fsksm.utm.my/~graphics/files/Week9-3D\\_Transformations.ppt](http://gmm.fsksm.utm.my/~graphics/files/Week9-3D_Transformations.ppt)
5. [www.cs.tau.ac.il/~dcor/Graphics/adv-slides/ICP.ppt](http://www.cs.tau.ac.il/~dcor/Graphics/adv-slides/ICP.ppt)



Thank You



**Thank You**