# Robust 3D-3D Registration

Jefry Tedjokusumo
HT045742U

# Problem Formulation

- Two objects $P$ and $Q$. $Q$ is fixed and $P$ will move to fit $Q$ .
- Two finite sets of points $\{ p_i \} \subseteq P$ and $\{ q_j \} \subseteq Q$ are selected.
- Compute the closest point in $\{ p_i \}$ to $\{ q_j \}$.

  $f(p) \rightarrow q$ where $|p-q| < |p-q'|$ $\forall q' \in \{ q_j \}$.

$$E = \sum_{i=0}^{n-1} \| RSp_i + T - f(p_i) \|^2$$

- Find $S$,$R$,$T$ such that $E$ minimum

# ICP Assumption

- Without lost of generalization we assume *P*'s centroid at the origin i.e. $\frac{1}{n}\sum_{i=0}^{n-1}\| p_i \| = 0$
- We will do scaling and rotation on *P* while translation on *Q*
- The new Error term : $\sum_{i=0}^{n-1}\| RSp_i - (f(p_i) + T) \|^2$

# ICP – Calculating optimum S

- $\{ p_i \}$'s centroid = $p'$
- $\{ q_i \}$.'s centroid = $q'$

- $S$ optimum when:

$$S = \frac{\dfrac{1}{n}\displaystyle\sum_{i=0}^{n-1} \left| p_i - p' \right|}{\dfrac{1}{m}\displaystyle\sum_{i=0}^{m-1} \left| q_j - q' \right|} = \frac{\text{p's variance}}{\text{q's variance}}$$

# ICP – Calculating optimum T

$$E = \sum_{i=0}^{n-1} \| RSp_i - (f(p_i) + T) \|^2$$

$$E = \sum_{i=0}^{n-1} \| RSp_i - f(p_i) \|^2 + \sum_{i=0}^{n-1} \left( 2\langle f(p_i), T \rangle + \langle T, T \rangle \right)$$

$$E = \sum_{i=0}^{n-1} \| RSp_i - f(p_i) \|^2 + \sum_{i=0}^{n-1} \left( \| f(p_i) + T \|^2 - \| f(p_i) \|^2 \right)$$

- $E$ minimized when $\sum_{i=0}^{n-1} \| f(p_i) + T \|^2 = 0$

$$T = -\frac{1}{n} \sum_{i=0}^{n-1} \| f(p_i) \|$$

# ICP Calculating optimum R

- Now we need to minimize this part:

$$\sum_{i=0}^{n-1} \| RSp_i - f(p_i) \|^2$$

- With the help of quaternion we have:

$$\sum_{i=0}^{n-1} \| q(Sp_i)q^* - f(p_i) \|^2 = \sum_{i=0}^{n-1} \left( \| Sp_i \|^2 - 2\langle q(Sp_i)q^*, f(p_i) \rangle + \| f(p_i) \|^2 \right)$$

- We want to maximize the following term

$$\sum_{i=0}^{n-1} \langle q(Sp_i)q^*, f(p_i) \rangle$$

# ICP Calculating optimum R

$$\sum_{i=0}^{n-1} \langle q(Sp_i)q^*, f(p_i) \rangle$$

from equation above, let $r_i = Sp_i$ *and* $t_i = f(p_i)$

- By quaternion properties we have:

$$\sum_{i=0}^{n-1} \langle q r_i q^*, t_i \rangle = \sum_{i=0}^{n-1} \langle q r_i, t_i q \rangle =$$

$$\sum_{i=0}^{n-1} \langle \overline{R_i} q, T_i q \rangle = \sum_{i=0}^{n-1} q^T \overline{R_i}^T T_i q =$$

$$q^T \left( \sum_{i=0}^{n-1} \overline{R_i}^T T_i \right) q$$

# ICP Calculating optimum R

- We want to maximize $q^T\left(\sum_{i=0}^{n-1} \overline{R_i}^T T_i\right)q$
- Let $C = \sum_{i=0}^{n-1} \overline{R_i}^T T_i$
- Since the matrix C is symmetric we can compute the eigenvalues and eigenvectors $Ce_i = \lambda_i e_i$ where $1 \le i \le 4$.
- $q^T C q$ is maximized when $q$ is the eigenvector that corresponds to largest eigenvalue

# Algorithm - Initialization

- If *P* has more than 1000 points, then
  $n$ = *number_of_point* / 1000
- { $p_i$ } = every $n^{th}$ point in *P*
- Do the same with *Q* (we have { $q_j$ } )

- Moves *P*'s centroid to origin (0,0,0)
- Scale *P* according to the formula

- Calculate *P*'s and *Q*'s principal axis
- Rotate *P* such that their principal axis aligned together

- Translate *Q*'s centroid to origin (0,0,0)

# Algorithm - Loop

- Compute the closest point in { $p_i$ } to { $q_j$ }.
  { $q_i'$ } = $f$ ( $p_i$ ) where { $q_i'$ } $\subseteq$ { $q_j$ }

- Find { $q_i'$ } centroid
- Translate Q such that { $q_i'$ } centroid at origin (0,0,0) → optimum translation

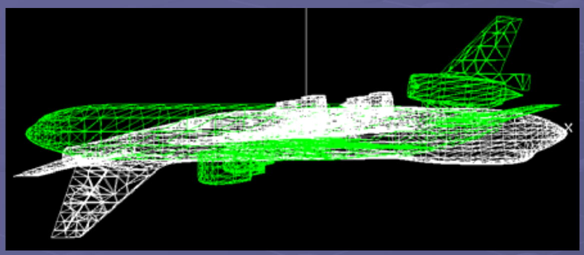
- Calculate optimum rotation based on { $p_i$ } and { $q_i'$ }
- Rotate P

- Repeat this process until optimum rotation and optimum translation not significant (close to 0)
- If optimum rotation and optimum translation close to 0, go to next initial guess

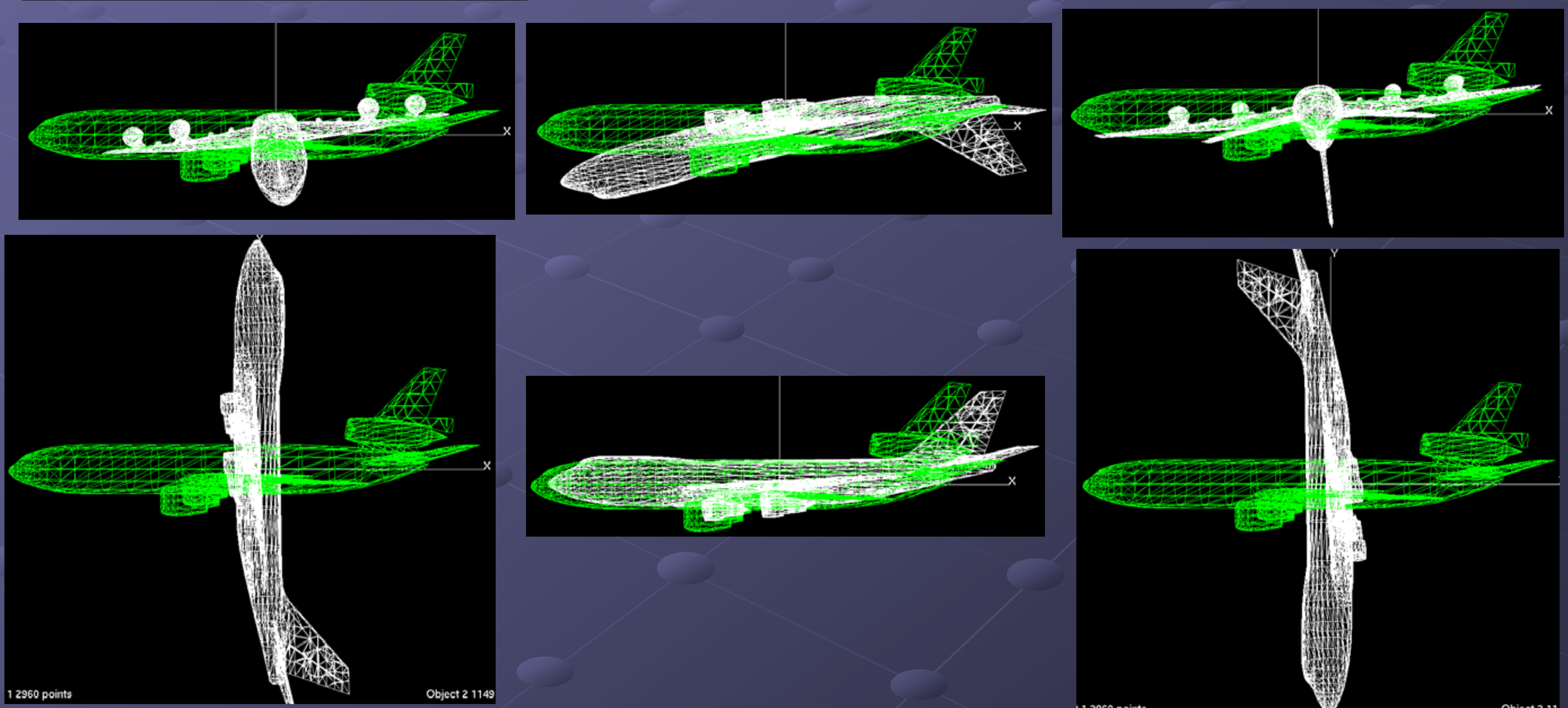
# Algorithm – Initial Guess (1)

- Restore P and Q to initial position
- Do the “algorithm – initiation”
- Choose the “correct” principal axis as the rotational axis
- Rotate the “correct” degree
- Do the Algorithm - Loop

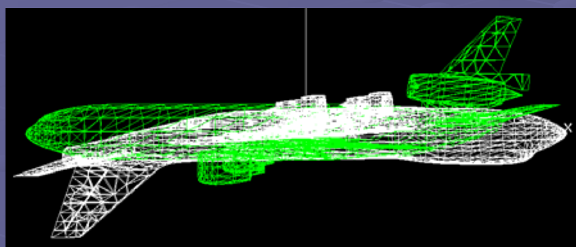
# Algorithm – Initial Guess (2)



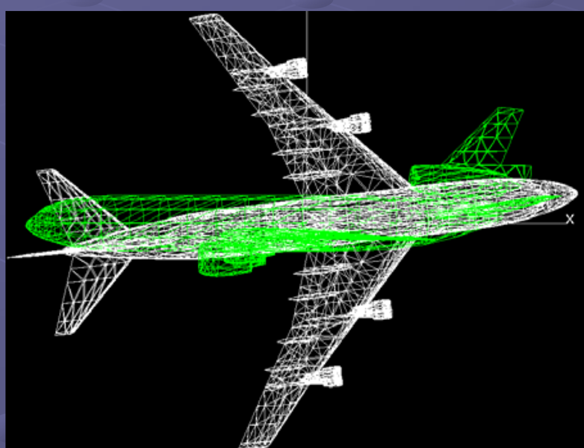
Initial position

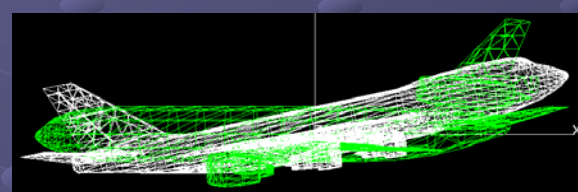


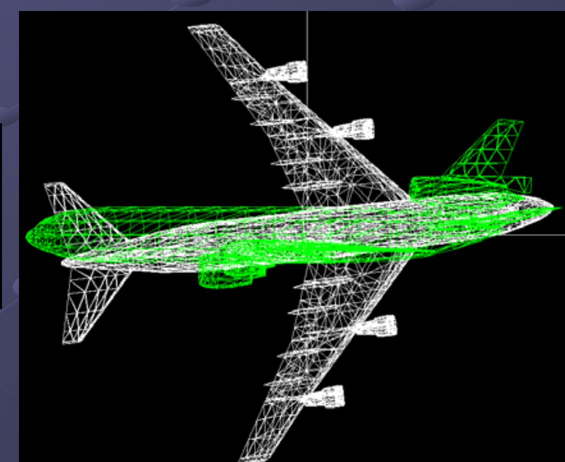
# Algorithm – Initial Guess (3)



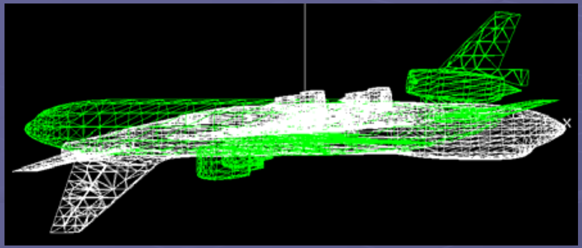
Used as reference



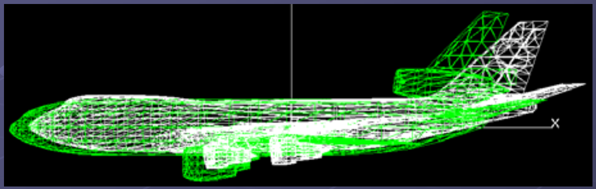


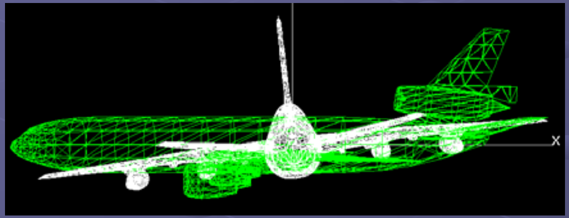


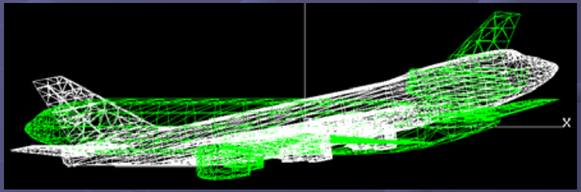
# Algorithm – Initial Guess (4)

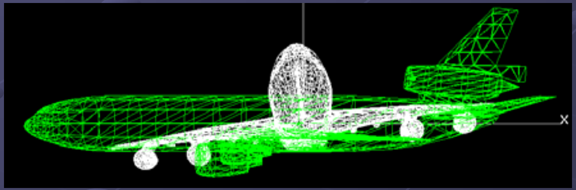


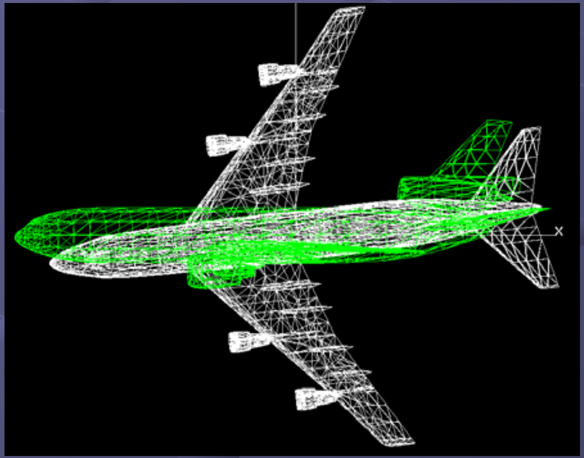
Change reference

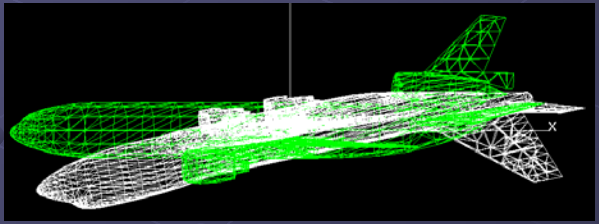


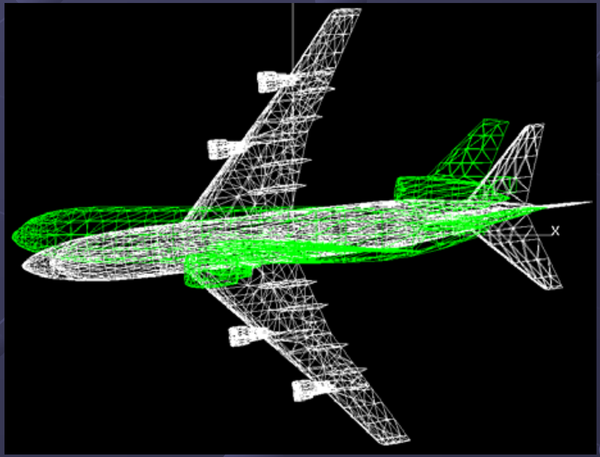




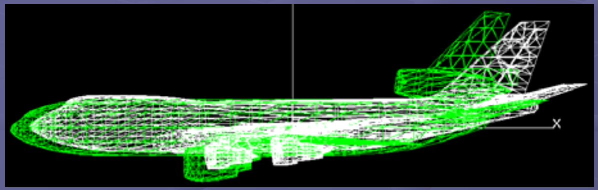




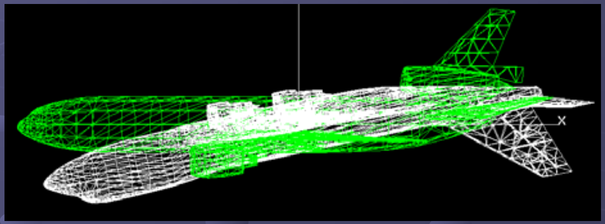


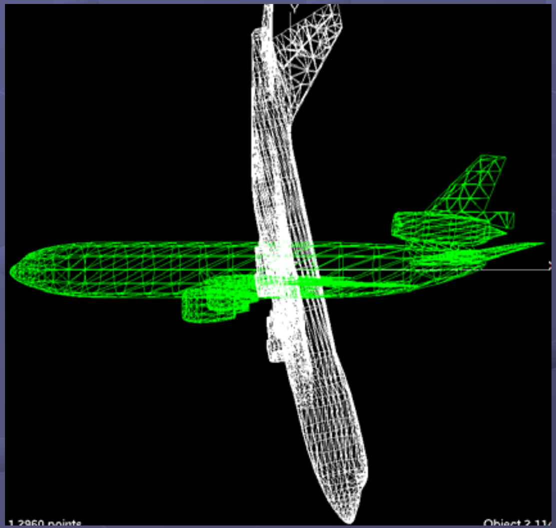


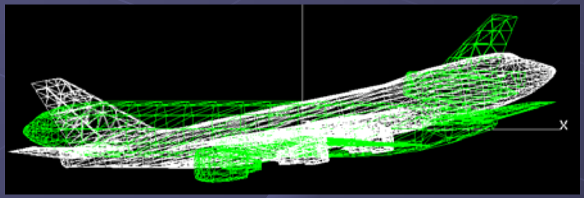
# Algorithm – Initial Guess (5)

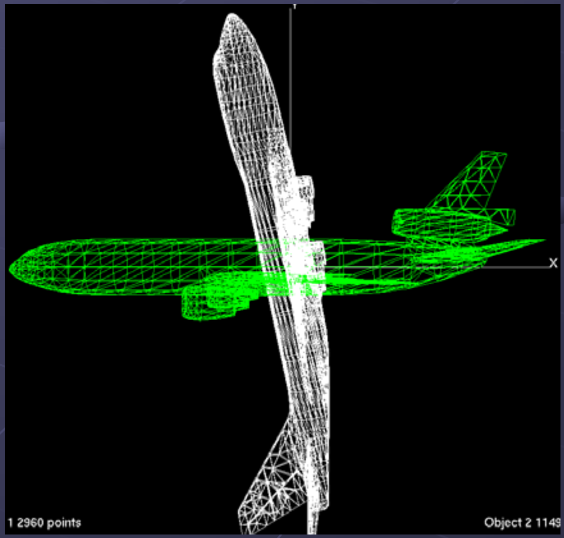



Change
reference



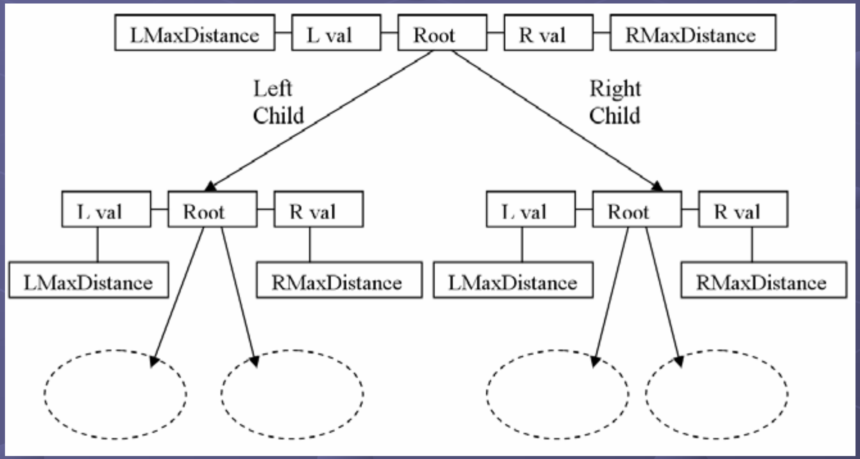


# Algorithm – Initial Guess (6)

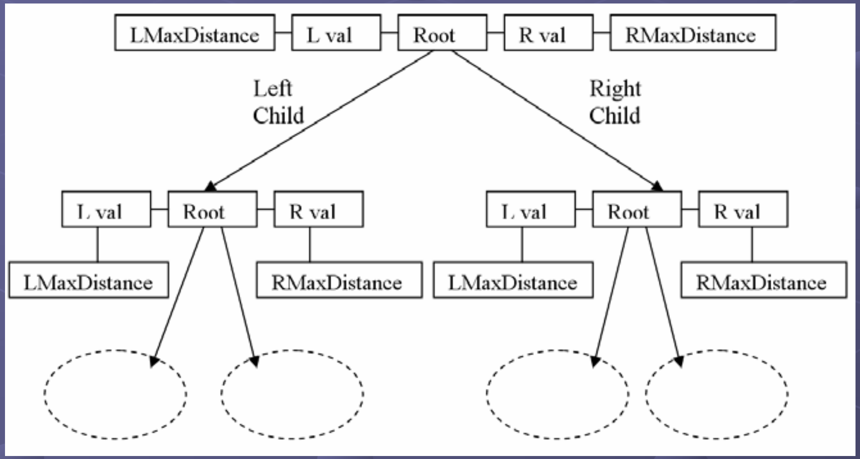
- Imagine the plane on the right is the search space, and the dot is the position where the ICP started
- The Initial guess helps the search process to be started uniformly in the search space

# Algorithm – Nearest Neighbor



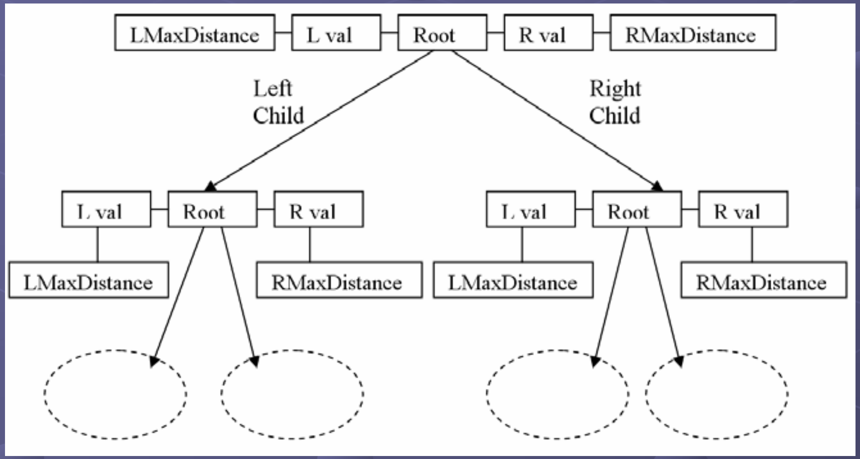

- A node of near tree consists of 4 elements:
  - Left Max Distance (scalar)
  - Left Value (3D point)
  - Right Max Distance (scalar)
  - Right Value (3D point)
- The tree it self has the following constraint:
  - $\forall$ points $p \in$ *Leftchild*,
    $| p - Lval | < LMaxDistance.$
  - $\forall$ points $p \in$ *Rightchild*,
    $| p - Rval | < RMaxDistance.$

# Algorithm – Nearest Neighbor




- Insertion of a point *p* is done as following:
- if | *p* - *Rval* | > | *p* – *Lval* |, update *RMaxDistance* and insert *p* to *Rightchild*
- if | *p* - *Rval* | < | *p* – *Lval* |, update *LMaxDistance* and insert *p* to *Leftchild*
- if current node is a leaf, insert *p* to either *Lval* or *Rval* as long as it is empty.

# Algorithm – Nearest Neighbor




- Finding nearest neighbor of point *p* is done as following:

- First test each of the *Lval* and *Rval* positions to see if one holds a point nearer than the nearest so far discovered.
  - if *d* = | *Lval* – *p* | < *currentNearest* then
    *currentNearest* = *d*
    *Lval* is current nearest point
  - if *d* = | *Rval* - *p* | < *currentNearest* then
    *currentNearest* = *d*
    *Rval* is current nearest point

- Now we test to see if the branches below might hold an object nearer than the best so far found. The triangle rule is used to test whether it is even necessary to descend.
  - if *currentNearest* + *LMaxDistance* > | *p* - *Lval* | then search *LeftChild*
  - if *currentNearest* + *RMaxDistance* > | *p* - *Rval* | then search *RightChild*

# Algorithm - output

$$Error = \frac{1}{n}\sum_{i=0}^{n-1} \| p_i - f(p_i) \|$$

$$\text{variance} = \frac{1}{n}\sum_{i=0}^{n-1} (Error - |(p_i - f(p_i))|)$$

Range x – y :

Number of ( $p_i$, $f$ ( $p_i$ ) ) pairs that the distance | $p_i - f ( p_i )$ |

Between
x/100 * r
and
y/100 *r

Where r is
max (| *pi – f ( pi )* |) –
min (| *pi – f ( pi )* |)

Error 0.217558
Variance 0.114638

range 0 - 10 = 280
range 10 - 20 = 393
range 20 - 30 = 281
range 30 - 40 = 254
range 40 - 50 = 184
range 50 - 60 = 45
range 60 - 70 = 6
range 70 - 80 = 8
range 80 - 90 = 4
range 90 - 100 = 23

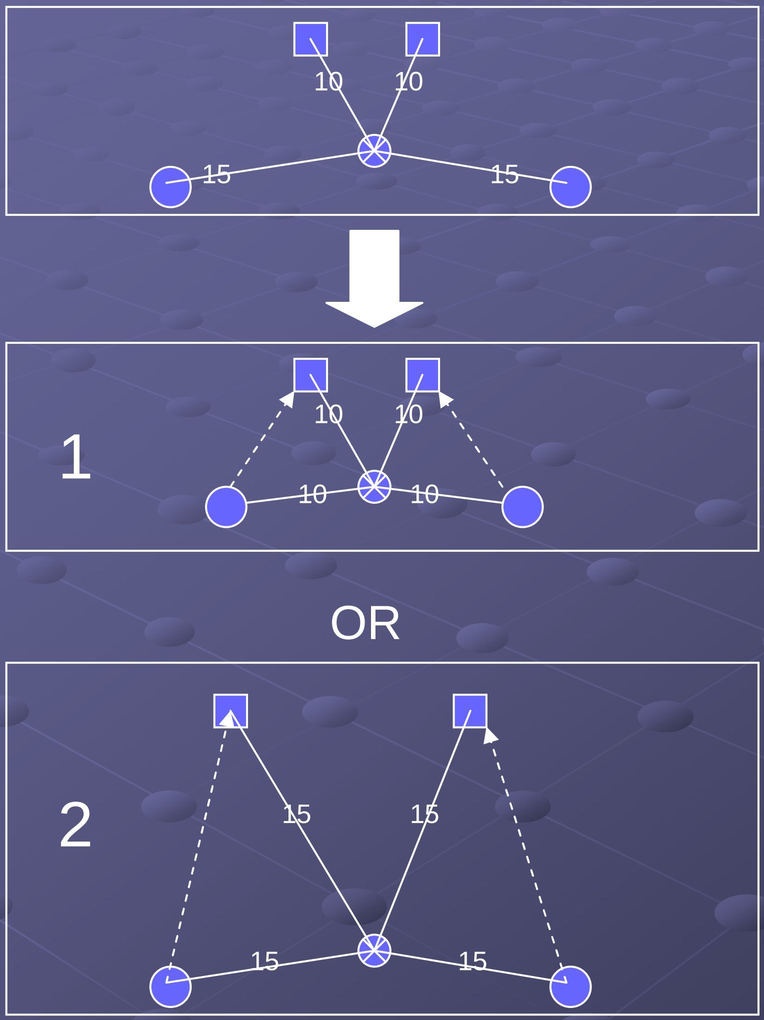
besIteration 63
bestGuess 8
rscale 0.714690

bestMatrix

| | | | |
|---|---|---|---|
| -1.000000 | 0.000445 | -0.000265 | 0.000000 |
| 0.000457 | 0.998742 | -0.050136 | 0.000000 |
| 0.000243 | -0.050136 | -0.998742 | 0.000000 |
| 0.000000 | 0.000000 | 0.000000 | 1.000000 |

bestVector
0.000062 -0.056674 -0.000111

# Error Normalization




- The squares are the points of object A
- The circles are the points of object B
- There are 2 possibilities, scale to A or scale to B
- In case 1 the "Error" obviously smaller (in distance) than the "Error" in case 2
- Case 1 and case 2 should have the same error term
- We normalize the error by divide the error value with object's variance

# Current ICP vs “Basic” ICP

- We cap the maximum point in the computation to be around 1000+

| **Experiment** | With PS | No PS |
| --- | --- | --- |
| Fish3 (19505) vs Fish4 (18878) | 64s | >5 minutes |
| Human3 (166776) vs Human4 (14603) | 48.67s | >5 minutes |

Demo

# Current ICP vs "Basic" ICP

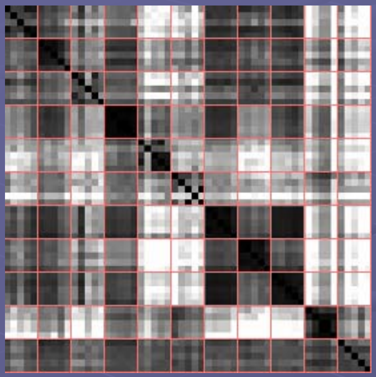
- Near Tree (n log n) vs "naive" nearest neighbor $n^2$

| **Experiment** | With NT | Naive |
|---|---|---|
| Fish3 (19505) vs Fish4 (18878) | 75s | 200s |
| Human3 (166776) vs Human4 (14603) | 53s | 198s |

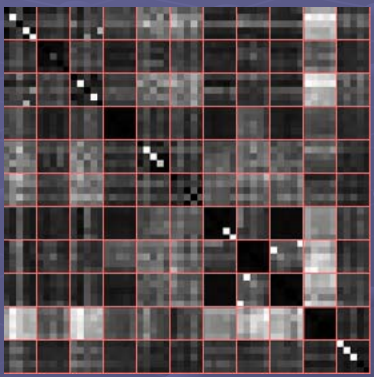
# Current ICP vs “Basic” ICP

- Initial Guess algorithm, helps to escape from local minima, and “guaranteed” to find global minima

# Experiments Result



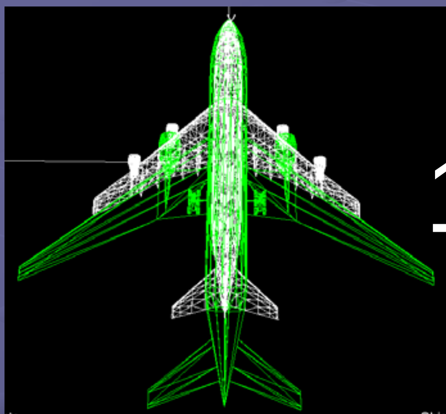
$Error_{ij} * Error_{ji}$



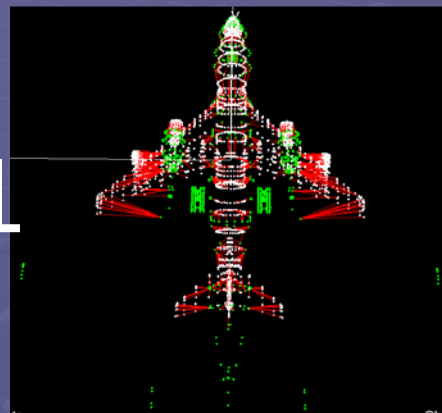
$Var_{ij} * Var_{ji}$

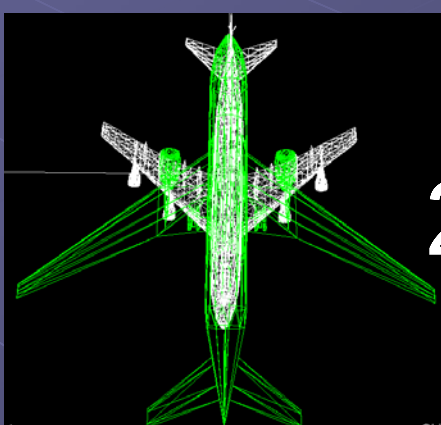
- 55 * 55 test
- 11 categories
  - Airplane
  - Animal
  - Bird
  - Car
  - Chair
  - Comp
  - Fish
  - Human-a
  - Human-b
  - Sphere
  - Tree

# Drawback

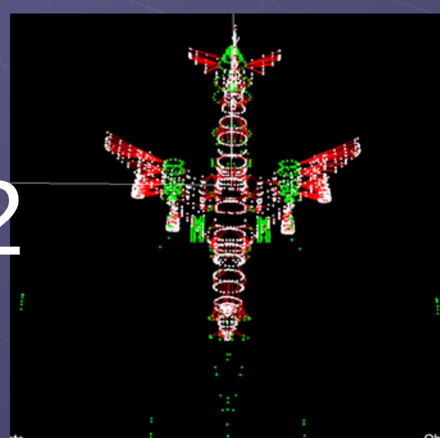


1





2



- Our algorithm considers case 2 as global optima
- We human will think case 1 as the global optima
- This because our algorithm works only in points. Case 2 has smaller error than case 1
- Solution:
  - Generate more points for green objects
  - Improve the algorithm from "point to point" to "point to triangle"

# Experiment Results