ELEC 278 – Guest Lecture

Computer Vision

Kevin Hughes

Outline

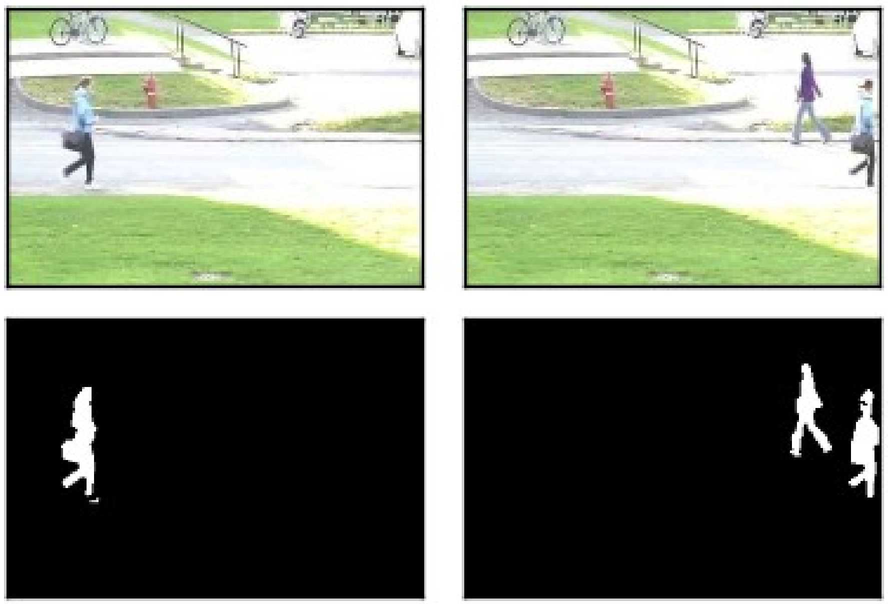
- Computer Vision Intro
- Computer Vision Applications
  - Motion Segmentation
  - Mapping
  - Object Recognition
  - Face Recognition
  - Deep Green
  - ARPool
- Image Data Structure
- Point Cloud Data Structure
- Itertative Closest Point and K-D-Trees

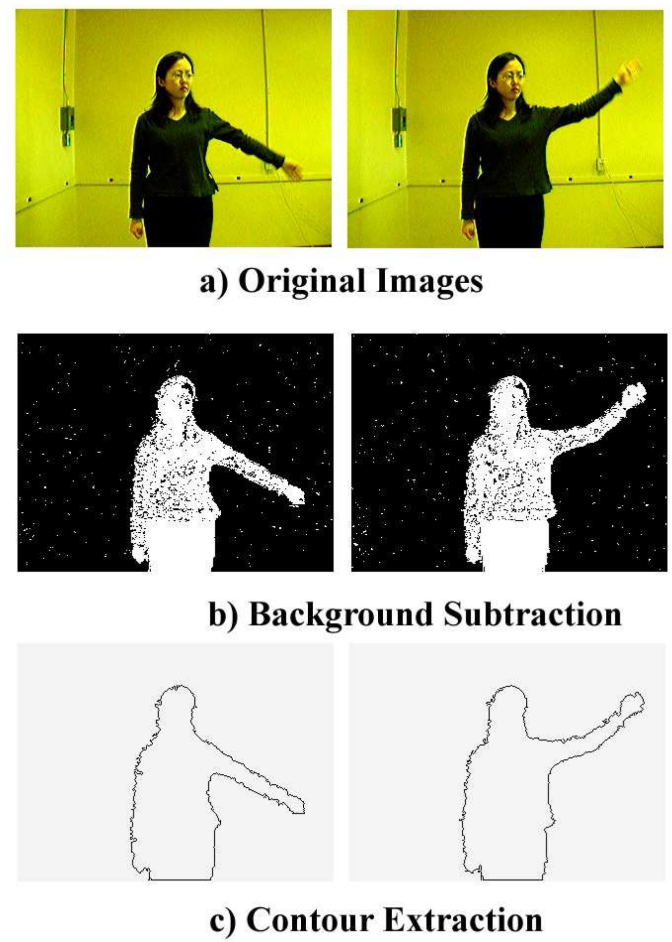
Computer Vision

The science of image processing and understanding

Motion Segmentation

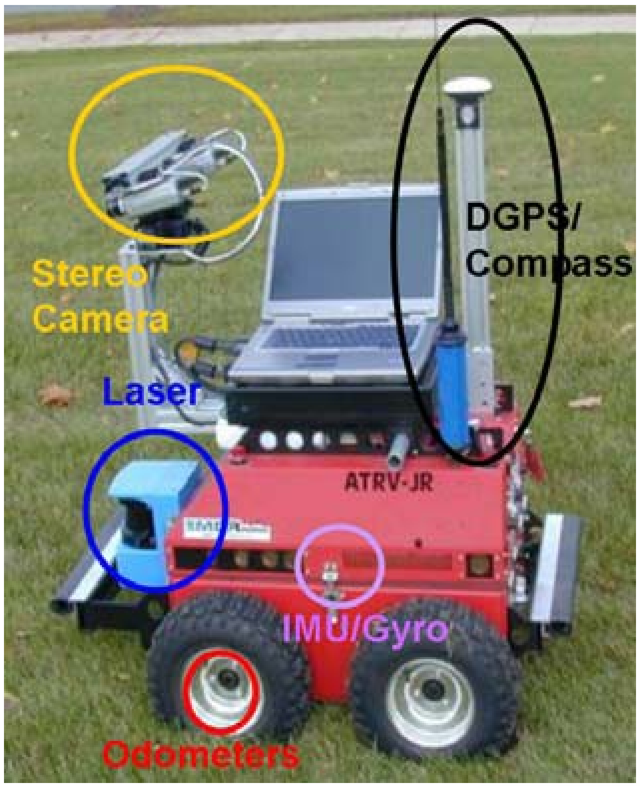
Background Subtraction

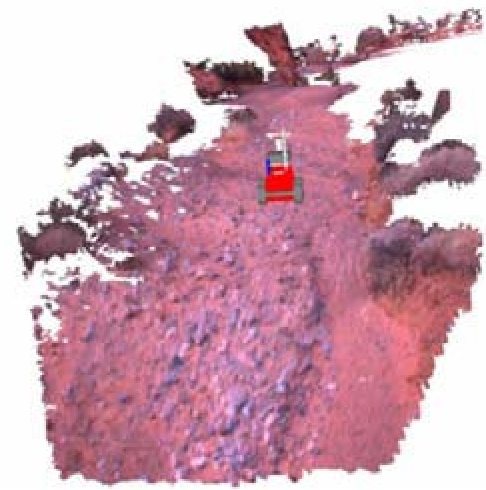


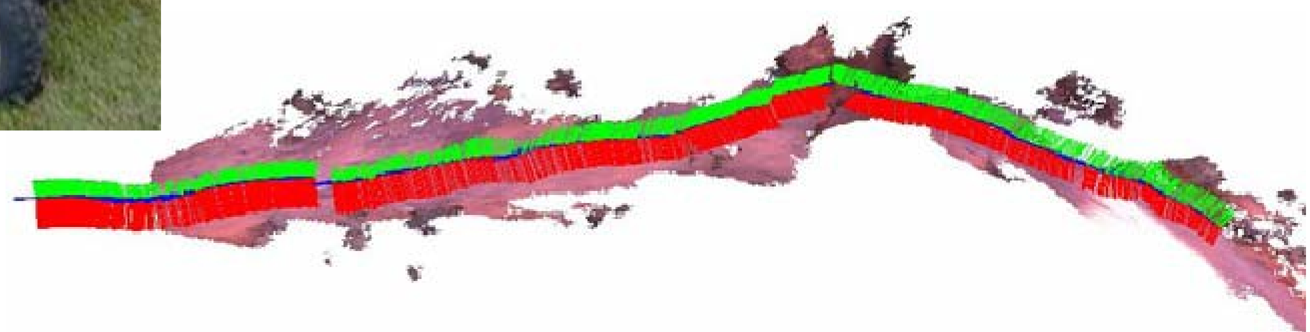


Mapping

SLAM

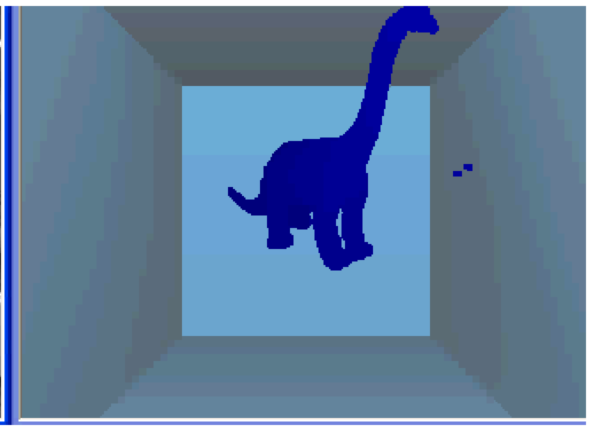






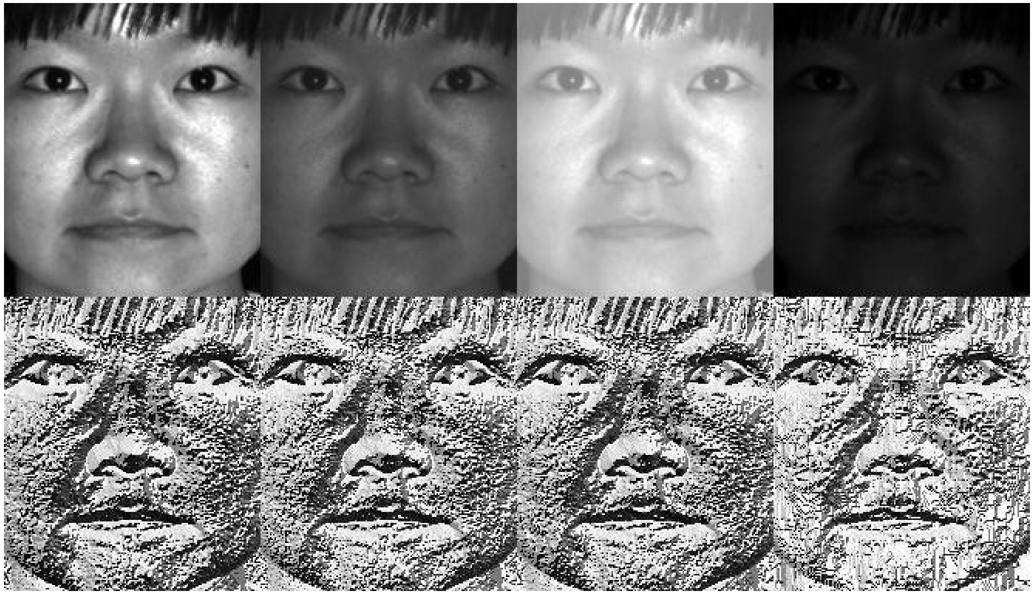
Object Recognition

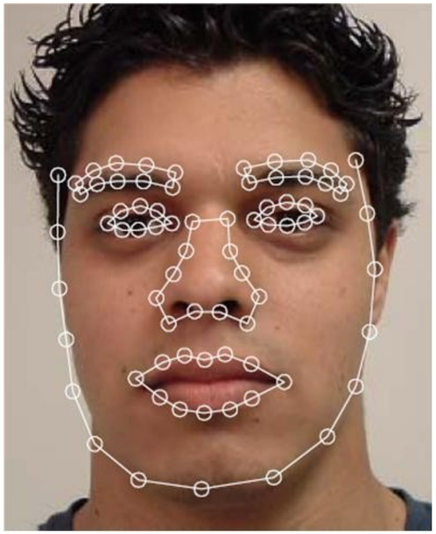
PWSE and BHT



Face Recognition

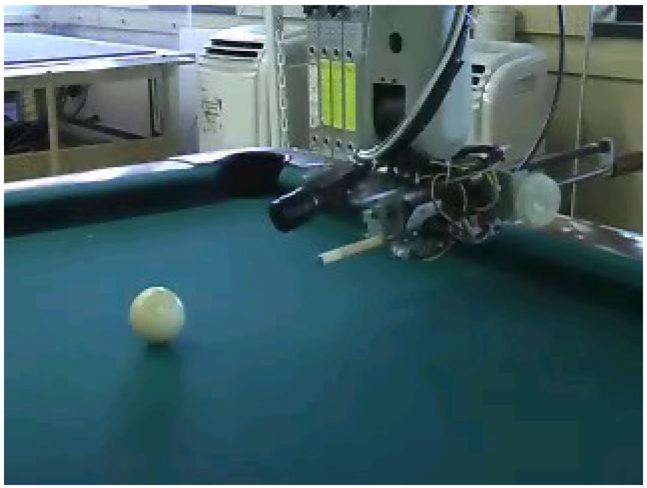
Eigenface

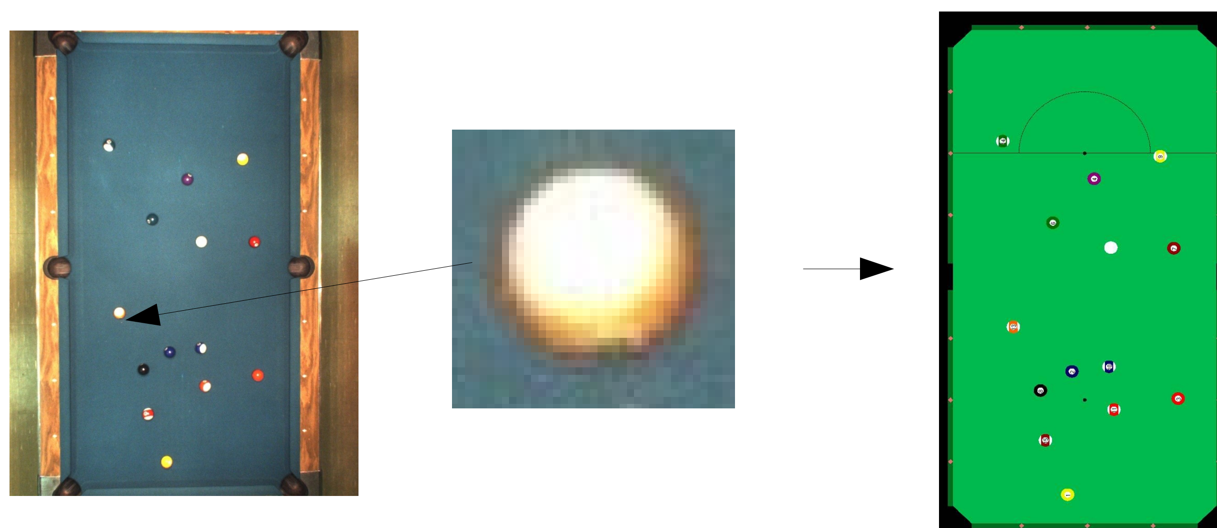




Robotic Pool

Deep Green





ARPool

Augmented Reality Pool

Image Data Structure

- Images are a lot of data!
  - A 640x480 image has 307200 pixels!
  - For a grayscale image each pixel is a single unsigned char
  - For a RGB image each pixel is an array of 3 unsigned chars

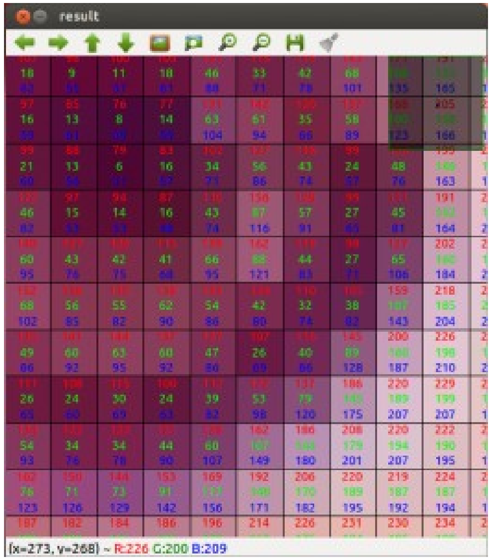


Image Data Structure

- Usually implemented as a 1D array with a header that contains the other important information

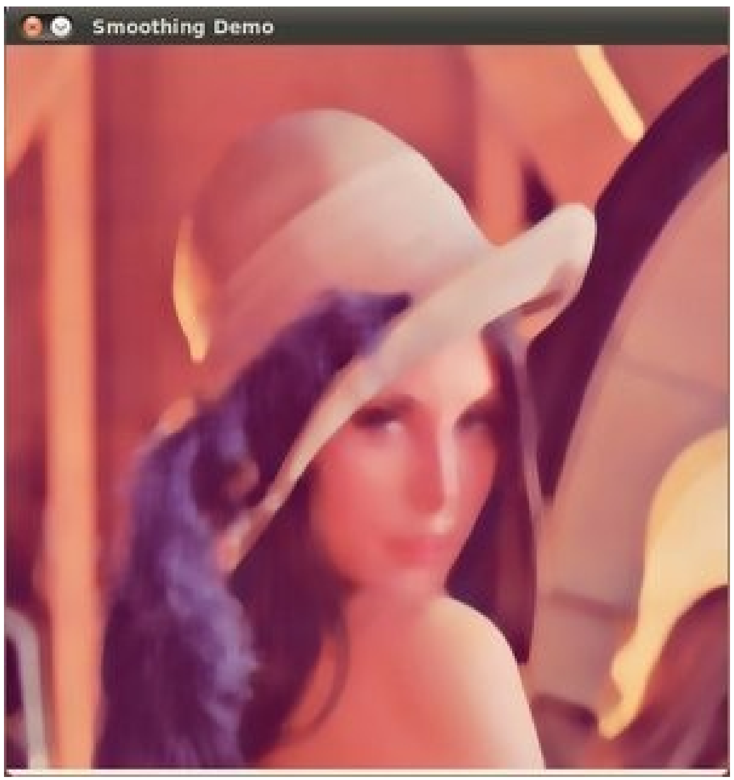
```
typedef struct Image
{
    int depth; // 1 for grayscale, 3 for RGB
    int width; // width of the image or cols of the matrix
    int height; // height of image or rows of the matrix
    unsigned char * data; // array
}
```

Image Data Structure

Iterating through an image:

```
Image img = ImageLoad("image.png"); // pretend function which inits the struct

for(int i = 0; i < img.width*img.height*img.depth; i++)
{
	unsigned char val = img.data[i];
}

/* or */

int step = img.width * img.depth * sizeof(unsigned char);
/* sometimes step is actually bigger then required to make better use
of memory - "padding" */

for(int r = 0; r < img.height; r++)
{
	for(int c = 0; c < img.width; c++)
	{
		unsigned char val = (img.data + step*r)[c];
	}
}
```

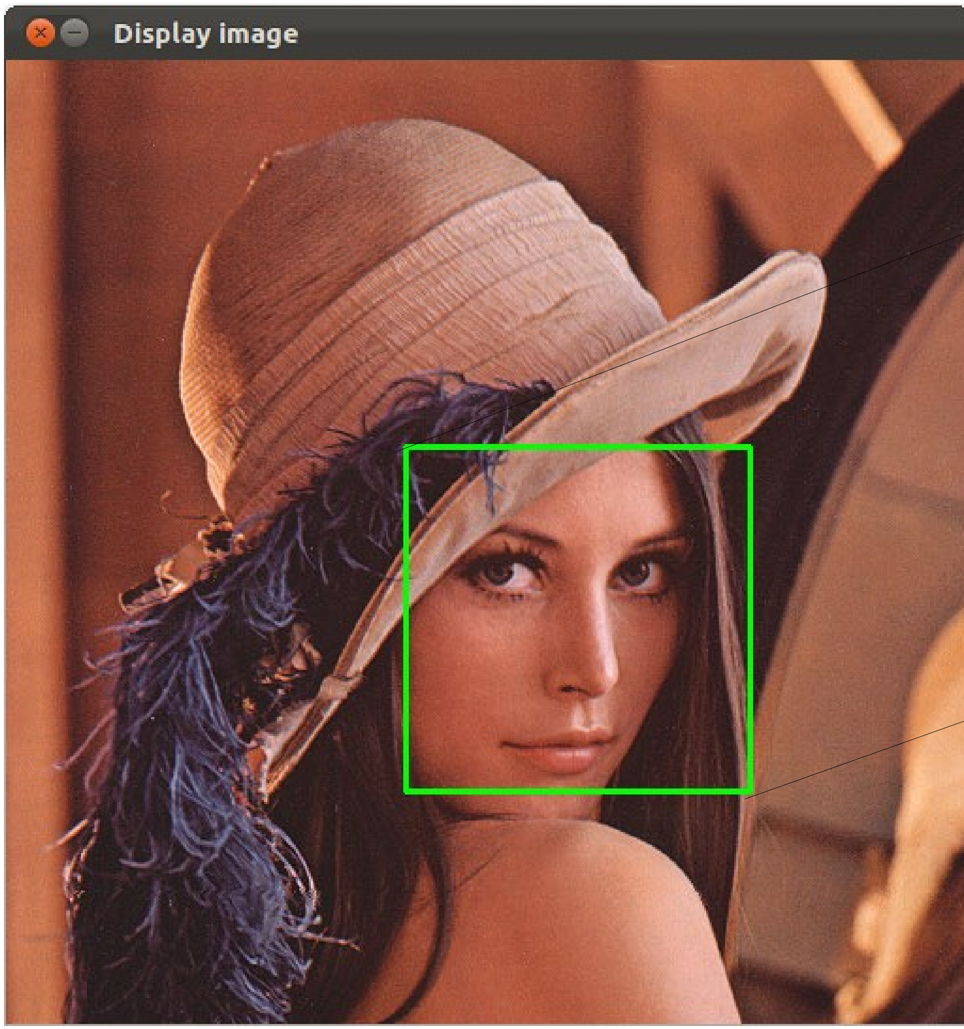
Image Data Structure



$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Image ROI

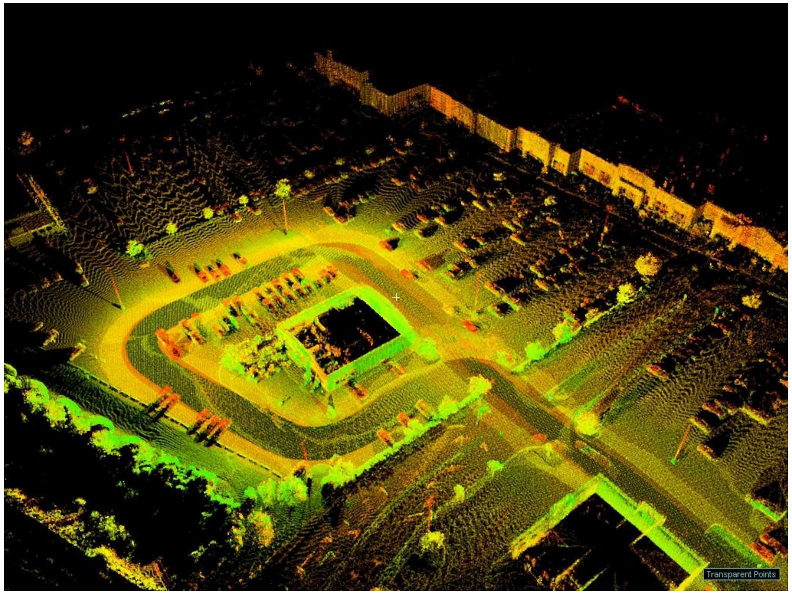


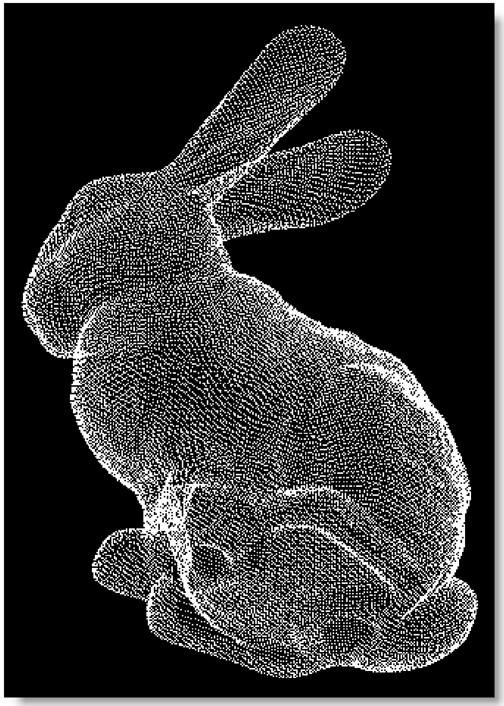
Same array in memory but 2 image headers

(not continous)

Point Clouds

3 Dimensional Image Data





Point Clouds

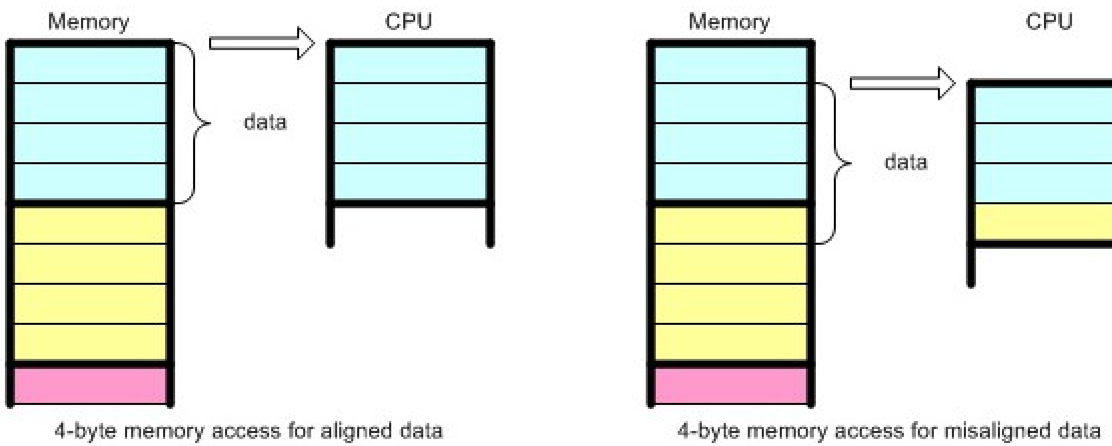
- Array of Points

```
struct PointXYZ
{
  float x;
  float y;
  float z;
  float padding;
};
```

pcl::PointCloud<pcl::PointXYZ>
Is essentially just a:
std::vector<PointT>

Data Alignment

- Data has 2 properties a value and a memory address
- Computers don't actually read a single address at a time but rather read chunks of 2,4,8,16 or 32 bytes



http://www.songho.ca/misc/alignment/dataalign.html

- Un-aligned data requires 2 reads compared to 1

The Nearest Neighbour Problem

Given a set of points S and a query point q find the closest point in S to q

- Complexity of O(Nd)
  - N is the number of points in S and d is the dimension of the space

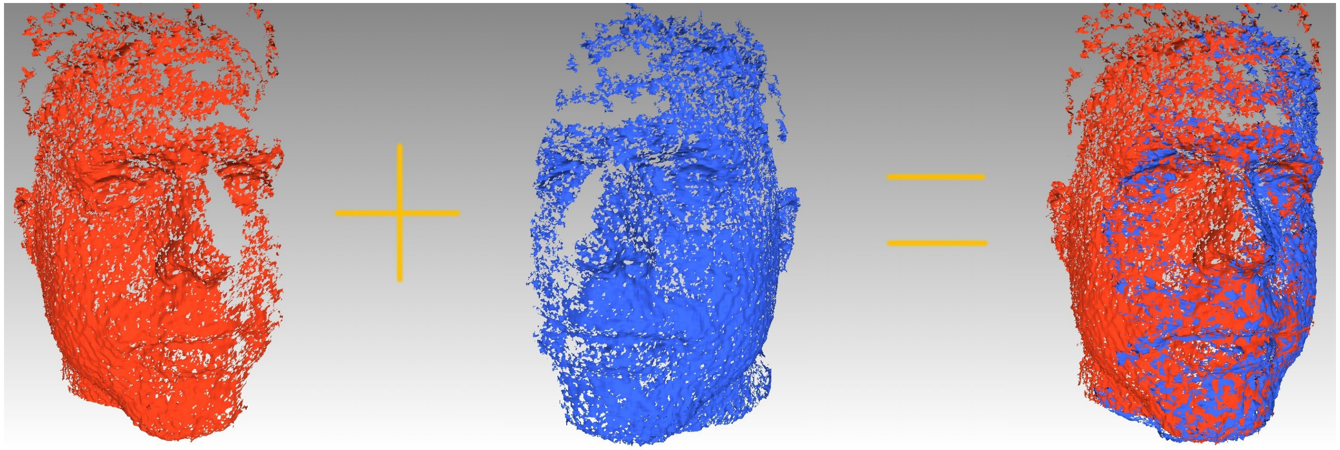
The Nearest Neighbour Problem

Important General problem in:

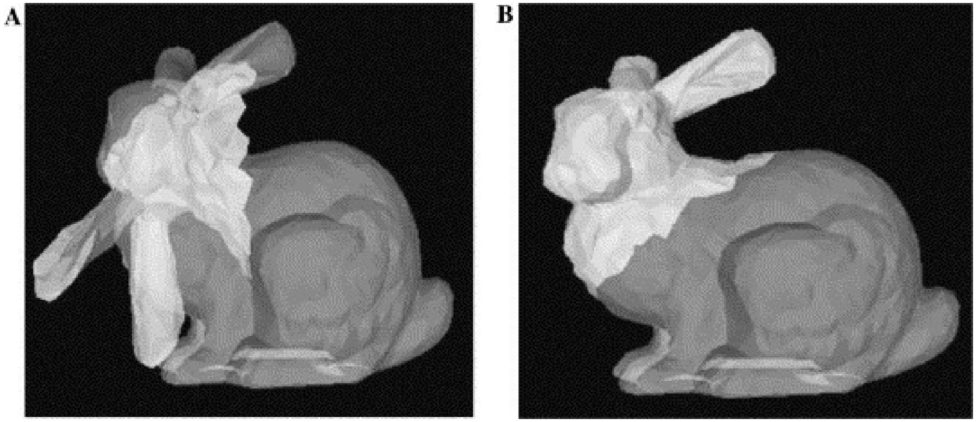
- Pattern Recognition
- Machine Learning
- Computer Vision
- Search

Iterative Closest Point

- An Algorithm for aligning 2 point clouds



http://dynface4d.isr.uc.pt/database.php



http://www.dlr.de/dlr/jobs/desktopdefault.aspx/tabid-10596/1003_read-6122/

Iterative Closest Point

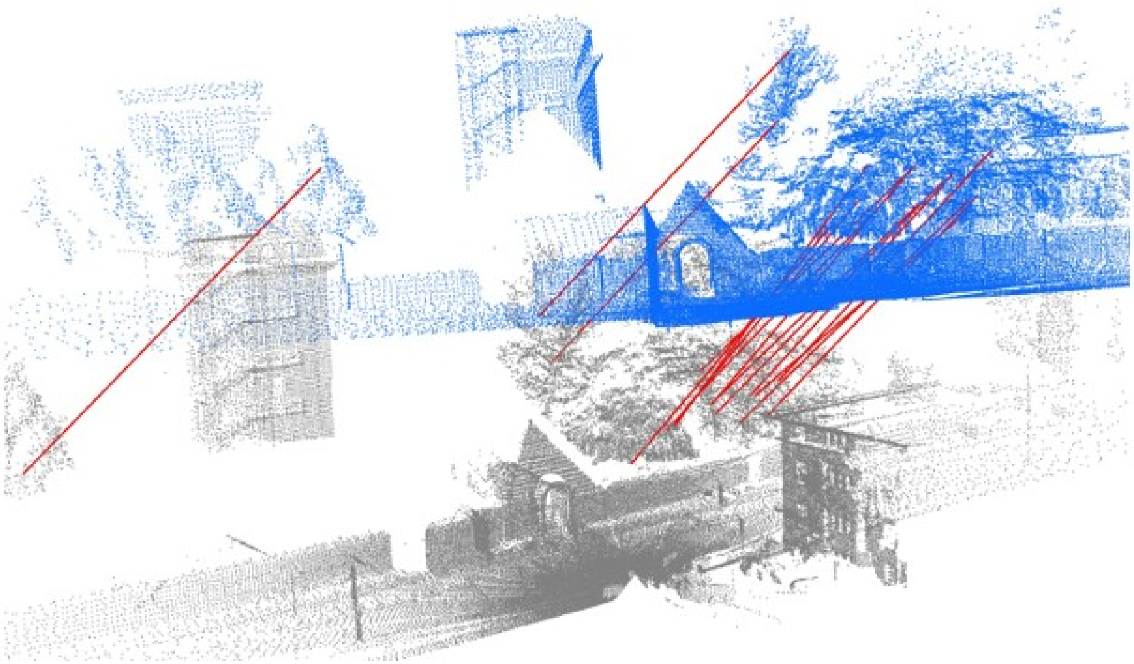
Essentially the algorithm steps are :

- Associate points by the nearest neighbor criteria.
- Estimate transformation parameters using a mean square cost function.
- Transform the points using the estimated parameters.
- Iterate (re-associate the points and so on).

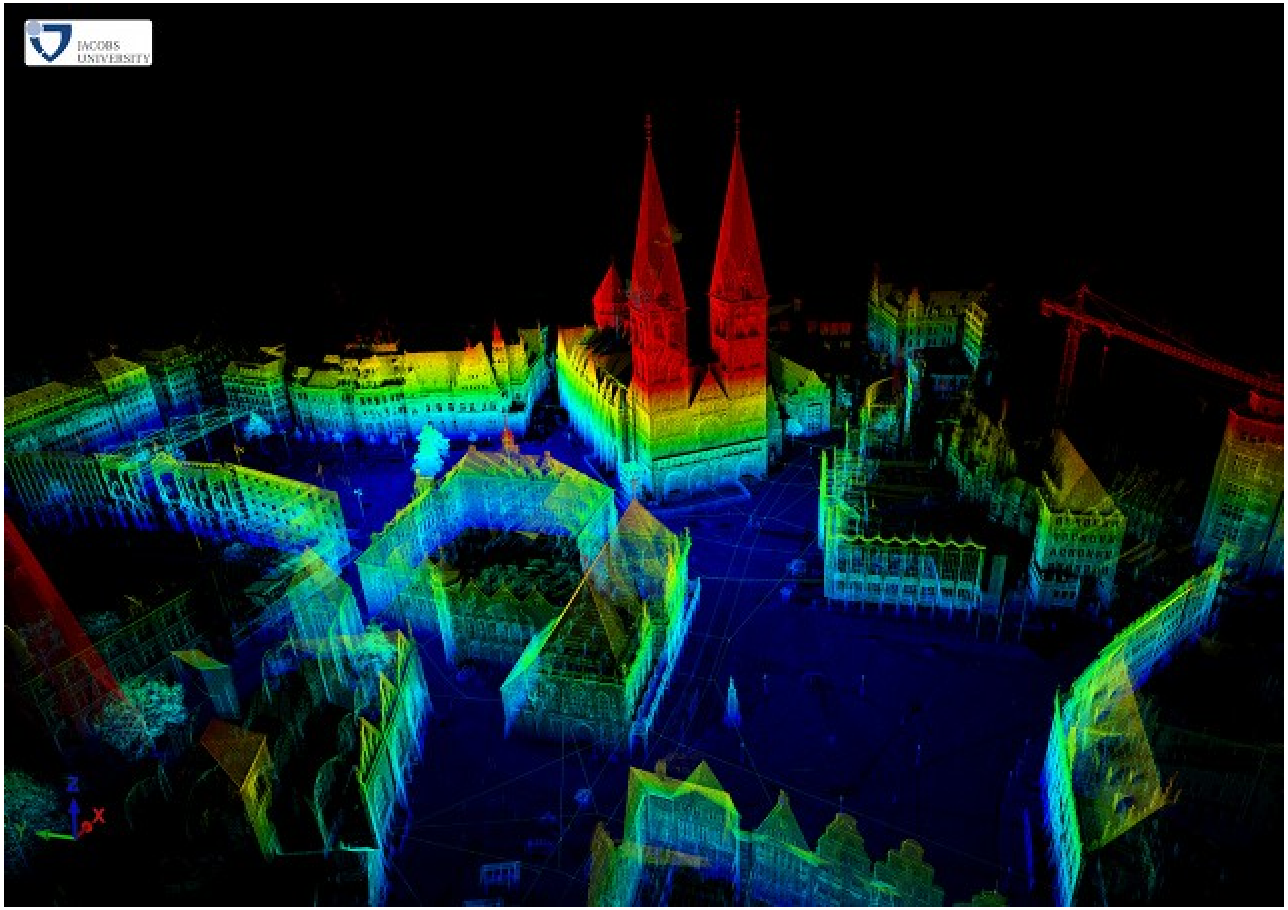
* (from wikipedia http://en.wikipedia.org/wiki/Iterative_closest_point)

Iterative Closest Point

- ICP requires N runs of finding the nearest neighbour and is by far the most computationally expensive part of the algorithm



Iterative Closest Point



N is often very large
for such problems

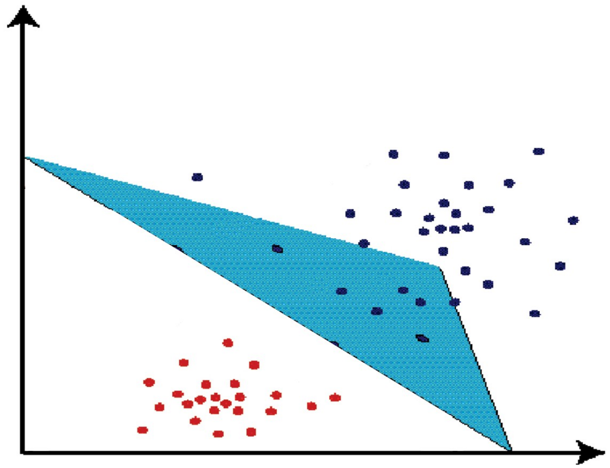
Iterative Closest Point

- How can nearest neighbour be made faster?
  - K-D-Trees!

K-D-Trees

The K-D-Tree is a binary tree where each node is a K-Dimensional point

We can think of each node as dividing the space with a hyperplane – all the points less than the plane are on one side while the points greater than are on the other side



K-D-Trees

K-D-Trees

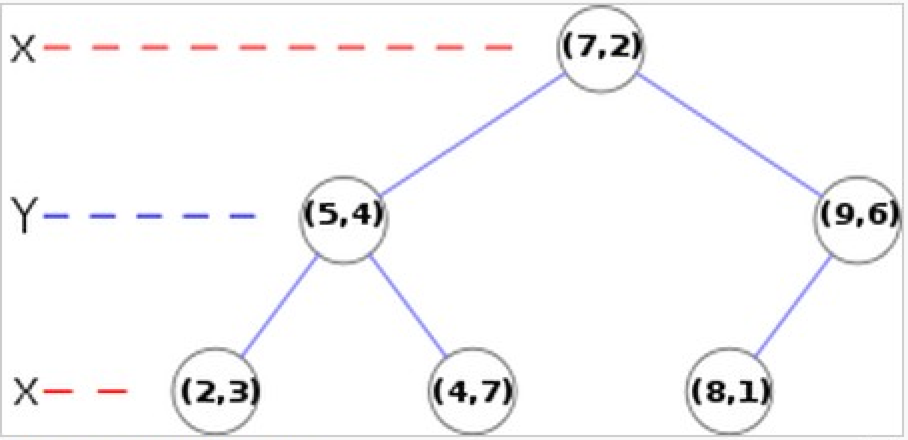
```python
def kdtree(point_list, depth=0):

    if not point_list:
        return None

    # Select axis based on depth so that axis cycles through all valid values
    k = len(point_list[0]) # assumes all points have the same dimension
    axis = depth % k

    # Sort point list and choose median as pivot element
    point_list.sort(key=lambda point: point[axis])
    median = len(point_list) // 2 # choose median

    # Create node and construct subtrees
    node = Node()
    node.location = point_list[median]
    node.left_child = kdtree(point_list[:median], depth + 1)
    node.right_child = kdtree(point_list[median + 1:], depth + 1)
    return node
```

http://en.wikipedia.org/wiki/K-d_tree

K-D-Trees

K-D-Trees

```
point_list = [(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)]
tree = kdtree(point_list)
```



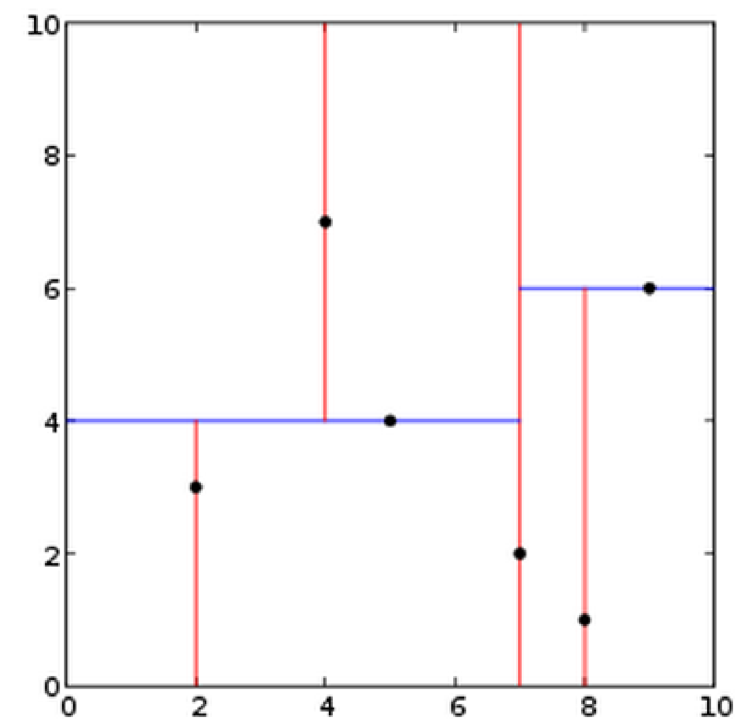
The resulting *k*-d tree.

http://en.wikipedia.org/wiki/K-d_tree

K-D-Trees

Searching:

- Tree is searched until a leaf node is found, this node is marked as the current best
- The alogrithm then travereses back up the tree checking if it is possible for there to be a closer point at each node
- The search finishes when the root node is reached



k-d tree decomposition for the point set (2,3), (5,4), (9,6), (4,7), (8,1), (7,2).

K-D-Trees

K-D-Trees - Michael Greenspan (2011)

```
Node buildKdTree( PointSet P )
{
   Node node = null ;

   if ( sizeof(P) <= minSize )
   {
          node = new LeafNode( P ) ;
   }
   else
   {
          int k = calcMaxSpreadD( P ) ;
          float mid = calcMid( P, k ) ;

          node = new InternalNode(k,mid) ;

          PointSet leftP = calcLE(P,k,mid) ;
          PointSet rightP = calcGT(P,k,mid) ;

          node.leftChild
                    = buildKdTree( leftP ) ;
          node.rightChild
                    = buildKdTree( rightP ) ;
   }

   return node ;
}
```

K-D-Trees

K-D-Trees - Michael Greenspan (2011)

```
Point searchKdTree( Point q, Node node )
{
   Point p ;
   if ( node.isInternal() )
   {
      int k = node.getK() ;
      float val = node.getVal() ;

          if ( q.getK( k ) <= val )
          {
               p = searchKdTree( q, node.leftChild ) ;
               if ( BOB ) search right subtree ;
          }
          else
          {
               p = searchKdTree( q, node.rightChild ) ;
               if ( BOB ) search left subtree ;
          }
   }
   else   // leaf node
   {
      p = findP( q, node ) ;

      if ( BWB ) done ;
      else return p ;
   }
}
```

K-D-Trees

- Using K-D-Trees the nearest neighbour search is reduced to an average complexity of O(log n)

Thanks!