INF 311

Introduction to Java Programming

Lecture 4: arrays and strings

Frank Nielsen

nielsen@lix.polytechnique.fr

Why do we need arrays?

- To handle many variables at once
- Processing many data or generating many results
- In mathematics, we are familiar with variables with indices:

$$S_n = x_1 + x_2 + \ldots + x_n = \sum_{i=1}^{n} x_i.$$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad A_{1,1} = 1,\ A_{1,2} = 2,\ \ldots,\ A_{3,2} = 8,\ A_{3,3} = 9$$

In most languages, indices start at zero (and not one).
...Just a convention

$(x_1, x_2, x_3, \ldots)$

$(x_0, x_1, x_2, \ldots)$

Declaring arrays in Java

For a given type, **TYPE[ ]** is the type of arrays storing elements of type TYPE.

- For arrays declared within the scope of functions:

- `int [ ] x;`
- `boolean [ ] prime;`
- `double [ ] coordinates;`
- `float [ ] [ ] matrix;`

- For arrays declared in the body of a class, use the keyword **static**:
  (array variables can be used by any function of the class)

- `static int [ ] x;`
- `static boolean [ ] prime;`

Building and initializing arrays

declarray.java

```
class declarray{


    static int digit [] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};

    static double x [] = {Math.PI, Math.E, 1.0, 0.0};

    static boolean prime[]={ false, true, true, true, false, true, false, true, false, false };


    static void MyFunction(int n)
    {
        int y [];

        // Allocate an array of size n
        y=new int[n];

    }


    pu
    {


}
```

Observe:
static boolean prime[ ]={ false, true, true, true, false, true, false, true, false, false };
but not
static boolean prime[10]={ false, true, true, true, false, true, false, true, false, false };

Building and initializing arrays

- Declare array with the reserved keyword **new**
- **Specify** the size of the array at built time
- Arrays can be declared and initialized **at once** too:
- `int [ ] x;`
- `x=new int [32];`
- `boolean [ ] prime = new boolean[16];`
- Arrays initialized by **enumerating** all its values:

```
int [ ] prime={2, 3, 5, 7, 11, 13, 17, 19};
```


Size of arrays

Size of arrays is given by the member function **length**:

```
prime.length;
System.out.println(prime.length);
```

Size of arrays **fixed for once**, **cannot be changed**:

```
array.length=23; // Generate an error
```


Size of arrays

declarray2.java

```
1  class declarray2{
2
3
4      public static void MyFunction(int n)
5      {
6          int array []=new int [n];
7          int i;
8
9          InformationArray(array);
10     }
11
12
13     public static void InformationArray(int [] t)
14     {
15         System.out.println("Size of array given in argument is:"+t.length);
16     }
17
18
19     public static void main (String[] args)
20     {
21
22         MyFunction(2312);
23
24         MyFunction(2008);
25
26     }
27
28
29
30 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Size of array given in argument is:2312
Size of array given in argument is:2008
Press any key to continue...
```


Index range of arrays and exceptions

Powerful mechanism of modern languages (Java, C++)

If index is out of range, an **exception** is raised on the fly:
ArrayIndexOutOfBounds

Out of range accesses **may not be detected** by the compiler:
Bug that yields termination or system crash

... However, fortunately, Java can **catch** exceptions too.

Size of arrays cannot be modified

declarray3.java

```
1  class declarray3{
2
3
4      public static void main (String[] args)
5      {
6
7
8          int x []=new int [12];
9
10         x.length=7;
11
12     }
13
14
15
16 }
```

```
Build Output
--------------------Configuration: <Default>--------------------
D:\Enseignements\INF311\Lectures2008\prog-inf311.4\declarray3.java:10: cannot assign a value to final variable length
                x.length=7;
                ^
1 error

Process completed.
```


Index range of arrays and exceptions

arraybound.java

```
1  class arraybound{
2
3
4      public static void main (String[] args)
5      {
6
7          int[] u = new int [16];
8
9          int [    ] v={0,1,2,3,4,5,6,7,8};
10
11
12         long l=v.length;
13
14         System.out.println("Size of v:"+l);
15
16         System.out.println(v[4]);
17
18         System.out.println(v[12]);
19
20     }
21
22
23
24 }
```

Observe the correct parsing

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Size of v:9
4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 12
        at arraybound.main(arraybound.java:18)
Press any key to continue...
```


The concept of references

An array is allocated as a single contiguous memory block

Java is managing memory so you do not have to free it once the array is not used anymore: **garbage collector**

An array variable is, in fact, a **reference** to the array

This reference of the array is the **symbolic address** of the first element (index 0)

Thus, when we write...

```
int [ ] v = {0, 1, 2, 3, 4};
int [ ] t =v;
t[2]++;
System.out.println(t[2]++);
```

... the elements of the array v are not copied verbatim to t. Only the reference!!!

Arrays & references

arrayref.java

```
1  class arrayref{
2
3
4      public static void main (String[] args)
5      {
6
7          int[] u = new int [5];
8
9          int [] v={0,1,2,3,4};
10
11
12         System.out.println("Reference of array u in memory:"+u);
13
14         System.out.println("Value of the 3rd element of array v:"+v[2]);
15
16         // Declare a new array
17         int [] t =v;
18
19         System.out.println(v[2]);
20         t[2]++;
21         System.out.println(v[2]);
22         v[2]++;
23         System.out.println(t[2]);
24
25     }
26
27
28
29 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Reference of array u in memory:[I@3e25a5
Value of the 3rd element of array v:2
2
3
4
Press any key to continue...
```

Functions & arrays

Functions and procedures can have arrays as arguments.
(remember that array types are: TypeElement[ ])

Example: Function that returns the minimum of an array of integers

arraymin.java

```
class arraymin{

    static int minArray(int [] t)
    {
        int m=t[0];

        for(int i=1;i<t.length; ++i)
            if (t[i]<m)
                m=t[i];

                return m;
    }

    public static void main(String[] args)
    {
    int [] v=new int [23];

    for(int i=0;i<23;i++)
        v[i]=25-i;

        System.out.println("The minimum of the array is :"+minArray(v));
    }
}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
The minimum of the array is :3
Press any key to continue...
```


Functions & arrays

Example: Function that returns the inner product of 2 vectors (produit scalaire)

$$\langle (x_1,\ldots,x_n),(y_1,\ldots,y_n)\rangle := \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \cdots + x_n y_n$$

innerproduct.java

```
class innerprod{

    static double innerproduct(int [] x, int [] y)
    {
        double sum=0.0;

            System.out.println("Dim of vector x:"+x.length+ " Dim of vector y:"+y.length);

        for(int i=0;i<x.length; ++i)
                sum=sum+x[i]*y[i];

                return sum;
    }

    public static void main(String[] args)
    {
    int dimension=30;

    int [] v1,  v2;

    v1=new int[dimension];
    v2=new int[dimension];

    for(int i=0;i<dimension;i++)
        {v1[i]=i;
         v2[i]=i+1;
        }


        System.out.println("The inner product of v1 and v2 is "+innerproduct(v1,v2));
    }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Dim of vector x:30 Dim of vector y:30
The inner product of v1 and v2 is 8990.0
Press any key to continue...
```


Array arguments in functions

A variable that has a type array is a **reference** to the array
(the memory address of the first element)

Therefore an argument of type array **does not copy**
all array elements in the memory allocated for the function,
but rather allocate a **single memory reference**:
a machine word.

```
static void MyFunction(int [ ] x)
MyFunction(v);
```

Only the *reference of v* is copied to the memory allocated for the function MyFunction.

Array arguments in functions

Thus we can modify the inside a function the contents of the array: the values of the elements of the array.

modifyarray.java

```
class modifyarray{

static void swap(int [] t, int i, int j)
{
    int tmp;

    tmp=t[i];
    t[i]=t[j];
    t[j]=tmp;
}

static void DisplayArray(int [] x)
{
    for(int i=0;i<x.length;i++)
    System.out.print(x[i]+" ");

    System.out.println();
}

    public static void main(String[] args)
    {

        System.out.println("This program shows how to modify inside a function the contents of an array");

        int [] t={1,2,3,4,5,6,7,8,9};

        DisplayArray(t);

        swap(t,2,3);

        DisplayArray(t);
    }
}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
This program shows how to modify inside a function the contents of an array
1 2 3 4 5 6 7 8 9
1 2 4 3 5 6 7 8 9
Press any key to continue...
```

Functions returning an array

addvector.java

```
1  class addvector{
2
3      static int [] addvector(int [] u, int [] v)
4      {
5          int[] result;
6
7          result=new int[u.length];
8
9          for(int i=0;i<u.length;i++)
10             result[i]=u[i]+v[i];
11
12             return result;
13     }
14
15
16     public static void main(String[] args)
17     {
18         int [] x={1, 2, 3};
19         int [] y={4, 5, 6};
20
21         int [] z= addvector(x,y);
22
23     for(int i=0;i<z.length;i++)
24         System.out.print(z[i]+" ");
25     }
26
27 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE20
5 7 9 Press any key to continue...
```

Arrays of arrays...

So far, we described **linear array** (1D).
What about matrices (2D arrays)?

A bidimensional array (n,m) consists of **n lines**,
each of which is an array of **m elements**

```
int [ ] [ ] matrix;
matrix=new int[n][m];
```

By default, at initialization, the array is **filled up with zero**
Change the contents of 2D arrays using 2 **nested loops**:

```
for(int i=0; i<n; i++)
  for(int j=0; j<m;j++)
      matrix[i][j]=i*j+1;
```


2D Arrays: Matrix x vector product

matrixvector.java

```
1  class matrixvector{
2
3      static int [] MultiplyMatrixVector(int [][] mat, int [] v)
4      {
5          int[] result;
6
7          result=new int[mat.length];
8
9          for(int i=0;i<result.length;i++)
10         {
11             result[i]=0;
12
13             for(int j=0;j<v.length;j++)
14
15             result[i]+= mat[i][j]*v[j];
16         }
17             return result;
18     }
19
20
21     public static void main(String[] args)
22     {
23         int [][] M={{1, 2, 3}, {4,5,6}, {7,8,9}};
24         int [] v={1,2,3};
25
26         int [] z= MultiplyMatrixVector(M,v);
27
28     for(int i=0;i<z.length;i++)
29         System.out.print(z[i]+" ");
30     }
31
32 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
14 32 50 Press any key to continue...
```

Dichotomic search

Also called binary search algorithm

Assume we are given a **sorted array** of size n:

array[0] < array[1] < ... < array[n-1]

and a **query key** p

Seek whether there is an element in the array that has value p

That is, give a function that return the **index** of the element in the array with value p, or that returns -1 otherwise.

Dichotomic search: Think recursion!

- Start with a search interval [left, right] with left=0 and right=n-1
- Let m denote the **middle** of this interval: m=(left+right)/2
- If array[m]=p then we are done, and we return m;
- If array[m] <a, then if the solution exists it is in [m+1,right]
- If array[m]>a, then if the solution exists it is in [left,m+1]
- The search algorithm terminates if left>right, we return -1;

Dichotomic search: Recursion

dichotomysearch.java

```
class dichotomysearch{


    static int Dichotomy(int [] array, int left, int right, int key)
    {

        if (left>right) return -1;

        int m=(left+right)/2;

        if (array[m]==key) return m;
        else
        {
            if (array[m]<key) return Dichotomy(array,m+1, right, key);
                else  return Dichotomy(array,left,m-1, key);
        }

    }


    static int DichotomicSearch(int [] array, int key)
    {
        return Dichotomy(array,0,array.length-1, key);
    }

    public static void main (String[] args)
    {
    int [] v={1,6,9 ,12 ,45, 67, 76, 80, 95};

        System.out.println("Seeking for element 6: Position "+DichotomicSearch(v,6));
        System.out.println("Seeking for element 80: Position "+DichotomicSearch(v,80));
        System.out.println("Seeking for element 33: Position "+DichotomicSearch(v,33));
    }

}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Seeking for element 6: Position 1
Seeking for element 80: Position 7
Seeking for element 33: Position -1
Press any key to continue...
```


Strings: Basic objects in Java

- A string of character is **an object** with type **String**
- A variable of type String is a **reference** on that object:

String school= "Ecole Polytechnique";
String vars=school;

- Once built, a string object **cannot** be modified

- Beware: use only for moderate length strings, otherwise use the **class StringBuffer**

Class String: Some methods

A method is a function or procedure on an object class

Method Length() : gives the number of characters

```
String s= ''anticonstitutionnellement'';
System.out.println(s.length());
```

Method equals():

s1.equals(s2): Predicate that returns true **if and only if** the two strings s1 and s2 are made of the same sequence of characters.

```
String s1=''Poincare'';
String s2=TC.lireMotSuivant();
System.out.println(s1.equals(s2));
```

Beware: s1==s2 is different!
It compares the **reference** of the strings.
(Physical versus logical equality test)

Class String in action...

ngmethod.java

```
class stringmethod{


    public static void main(String[] args)
    {
        String name="Ecole Polytechnique";
        String promotion="2008";

        String fullname=name+" "+promotion;

        System.out.println(fullname);
        System.out.println("Length:"+fullname.length());


        System.out.println("Type a sentence so that I check whether it is Poincare or not");
        String pattern="Poincare";
        String query=TC.lireMotSuivant();

        System.out.println(pattern.equals(query));

    }

}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Ecole Polytechnique 2008
Length:24
Type a sentence so that I check whether it is Poincare or not
Poincare
true
Press any key to continue...
```


Class String: More methods

Method **charAt()**:
s.charAt(i) gives the character at the (i+1)th position in string s.

```
String s= ''3.14159265'';
System.out.println(s.charAt(1));
```

Method **compareTo()**:
u.compareTo(v) compares lexicographically the strings u with v.

```
String u=''lien'', v=''lit'', w=''litterie'';
System.out.println(u.compareTo(v));
System.out.println(v.compareTo(w));
```


Class String: More methods

stringmethod2.java

```
1  class stringmethod2
2  {
3
4      public static void main(String[] args)
5      {
6          String s="3.141559";
7
8          System.out.println(s.charAt(1));
9
10         String u="lien", v="lit", w="litterie";
11         System.out.println(u.compareTo(v));
12         System.out.println(v.compareTo(w));
13
14     }
15
16 }
```

```
C:\PROGRA~1\XINOXS~1\JCR
.
-15
-5
Press any key to continue
```


Demystifying the main function

```
class ClassName
{
  public static void main(String[ ] args)
  {
  ...
  }
}
```

Function main has an array of string of characters as arguments
These strings are stored in args[0], args[1], ...
... when calling java main s0 s1 s2 s3

Use Integer.parseInt() to convert a string into an integer

```
D:\J>javac main.java

D:\J>java main a small test to parse as a command line
0:a
1:small
2:test
3:to
4:parse
5:as
6:a
7:command
8:line

D:\J>
```

Parsing arguments in the main function

parsingarg.java

```
1  class parsingarg{
2
3
4
5      public static void main(String[] args)
6      {
7
8          String first=args[0];
9
10         for(int i=1; i<args.length;i++)
11         if (first.compareTo(args[i])>0)
12         first=args[i];
13
14         System.out.println("Lexicographically maximum string is:"+first);
15     }
16
17 }
```

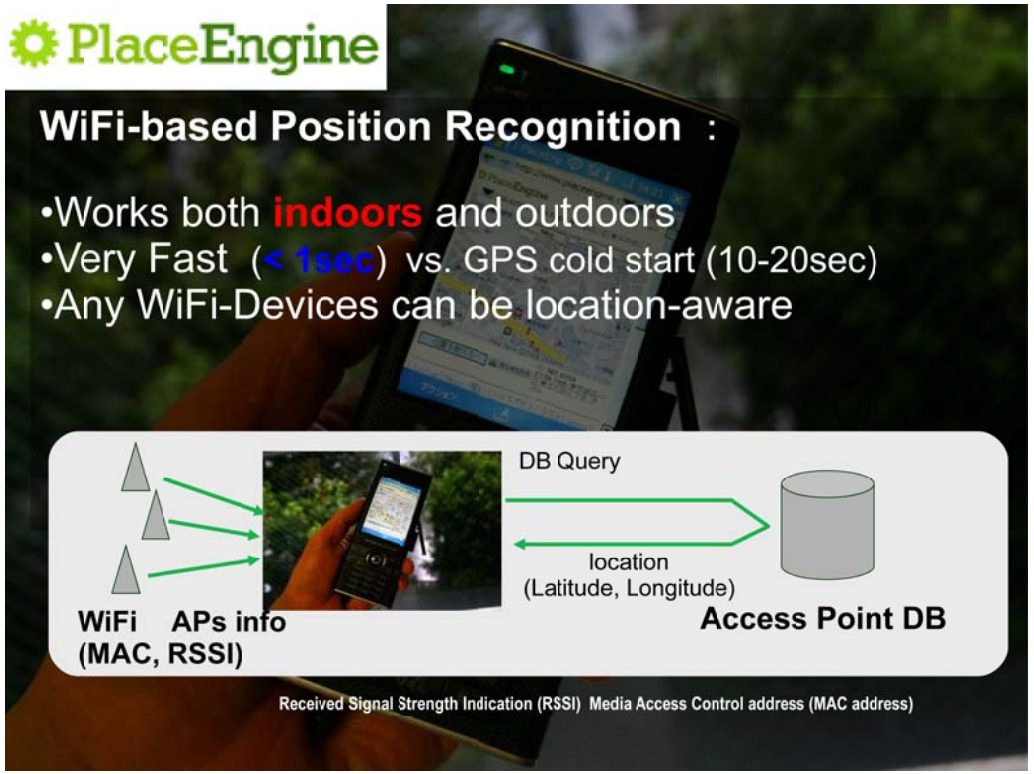
```
D:\J>java parsingarg lit lien litterie
Lexicographically maximum string is:lien

D:\J>
```


Parsing arguments in the main functior

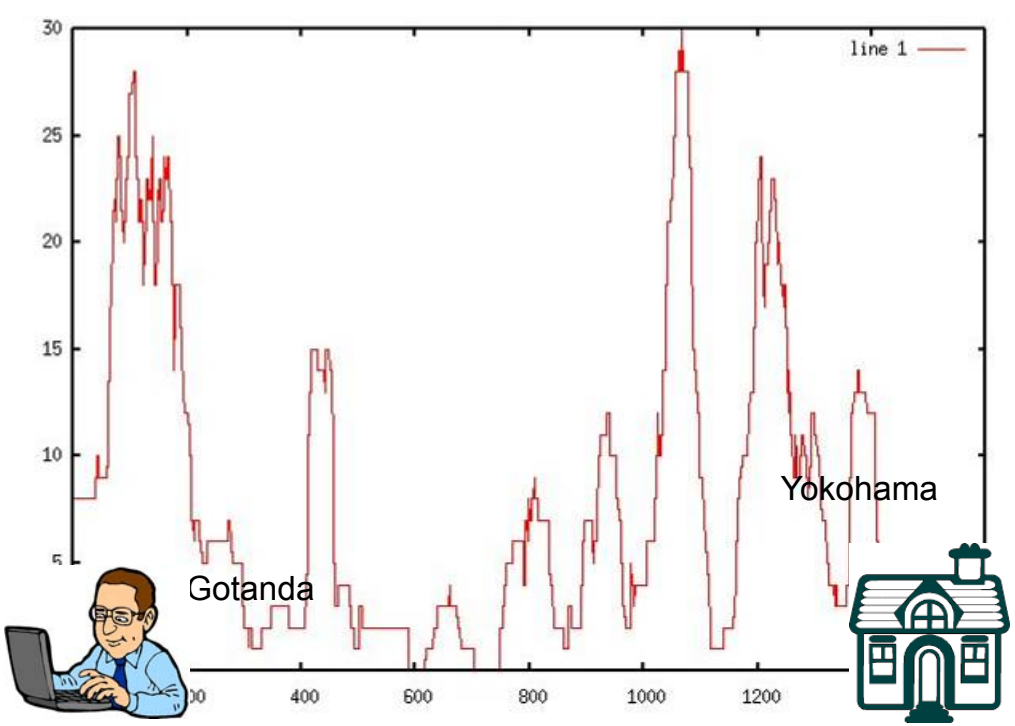
parsingarg2.java

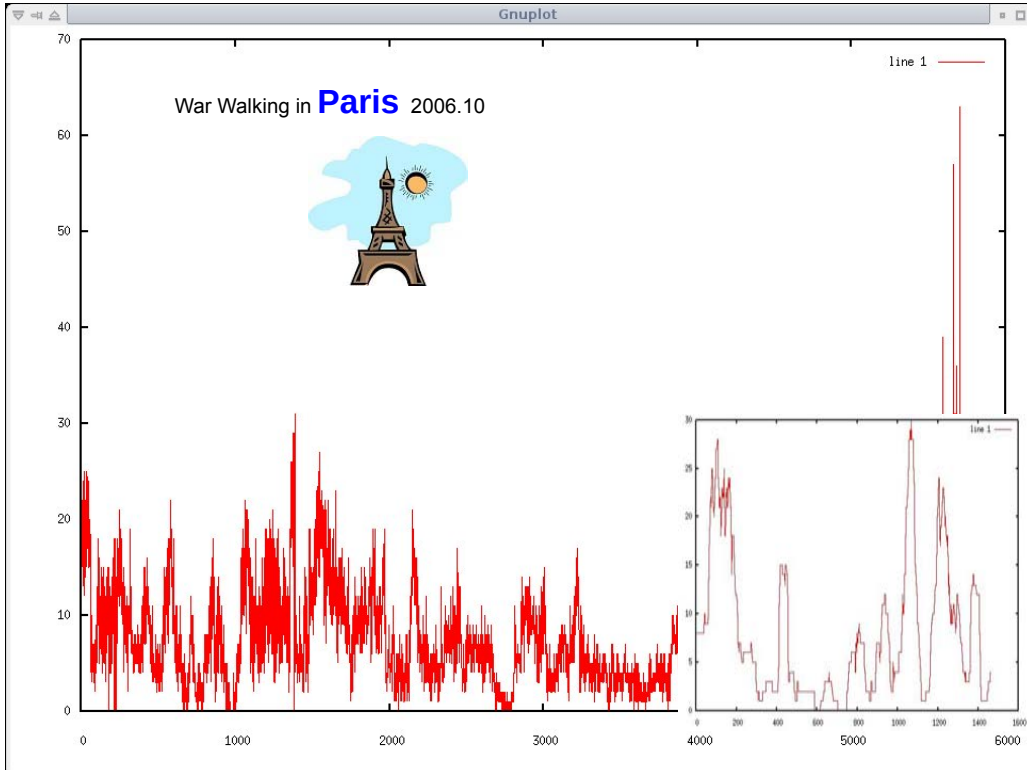
```
1  class parsingarg2{
2
3
4
5      public static void main(String[] args)
6      {
7
8          int first=0;
9
10         for(int i=1; i<args.length;i++)
11         if  (Integer.parseInt(args[first])>Integer.parseInt(args[i]))
12         first=i;
13
14         System.out.println("Location of minimum argument:"+first);
15     }
16
17 }
```

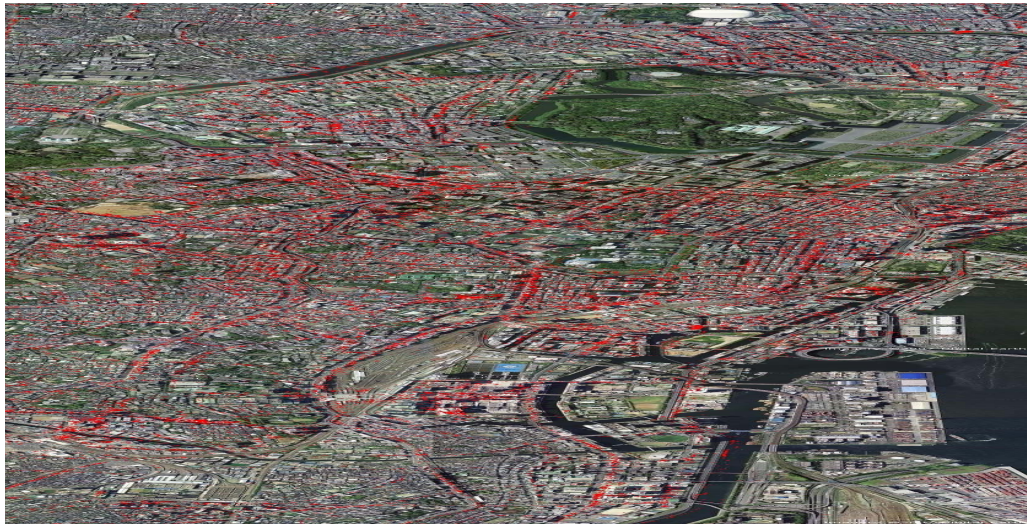
```
D:\J>javac parsingarg2.java

D:\J>java parsingarg2 9 4 6 2 6 4 1 3 5 4 6
Location of minimum argument:6

D:\J>
```

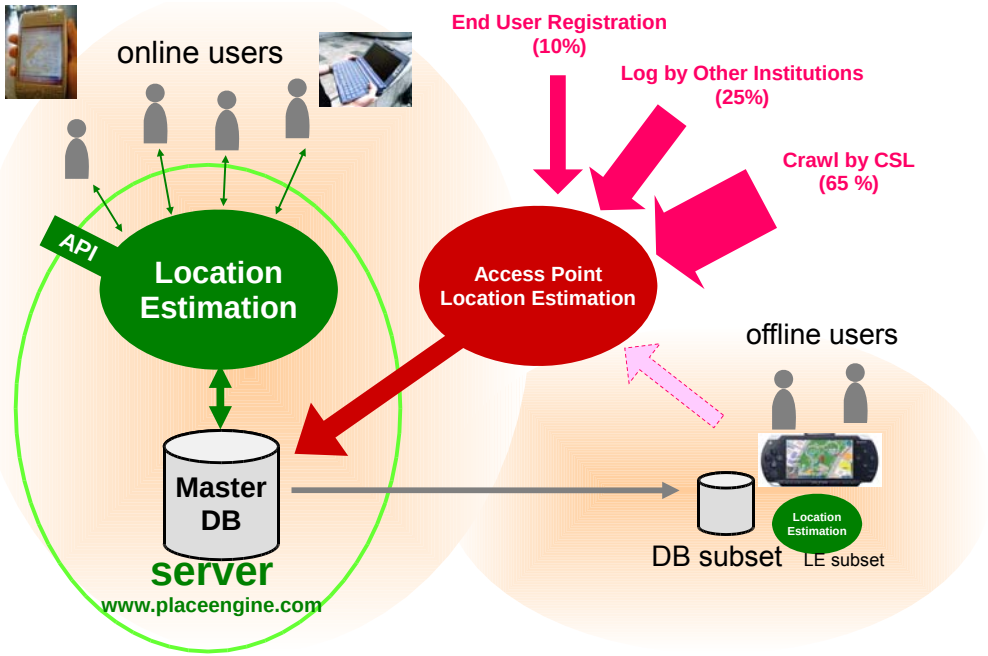
Rekimoto War Walking 2005





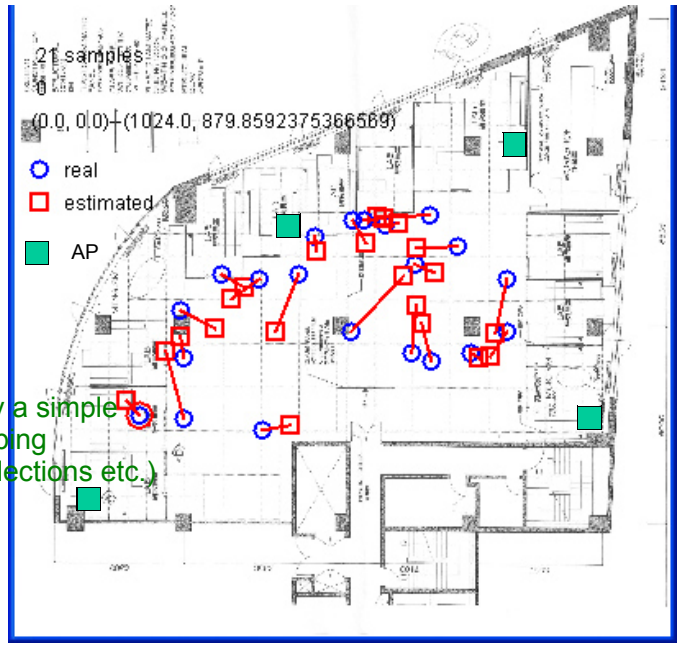


PlaceEngine Architecture



Indoor Estimation Results

(Rekimoto 2003 SonyCSL OpenHouse)



Location Estimated by a simple non-linear mapping
(to compensate wall reflections etc.)

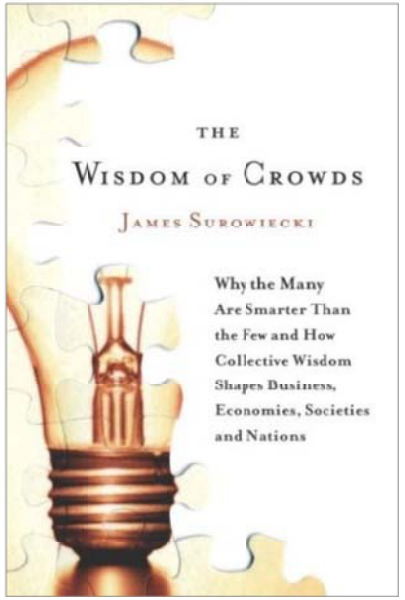
PlaceEngine Location Database

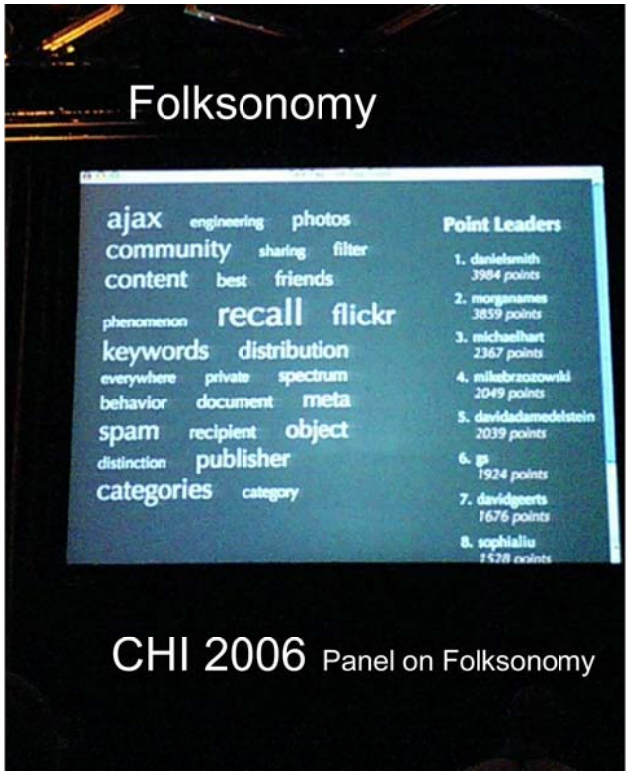
- Huge Realworld Database (> 700,000)
  - no one has total information
- How to gather data?
  - mega merging of (user participating) sensing information
- **Folksonomy → "Sensonomy"**

Sensonomy

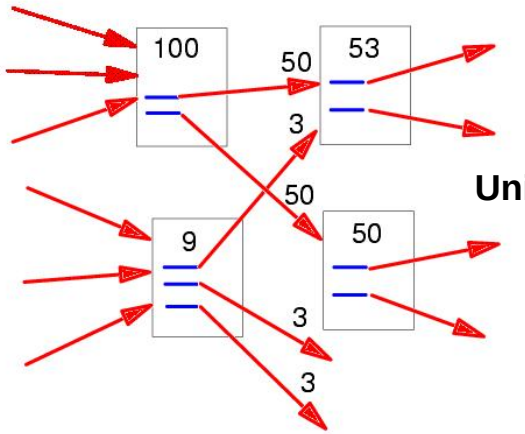
- Geographically Distributed
- Large (City to Planet) Scale
- Participated by many users
- Sensing Network

Le pouvoir est dans la multitude anonyme





Google PageRank

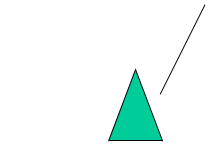


Unintentional folksonomy

Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, 'The PageRank Citation Ranking: Bringing Order to the Web', 1998,

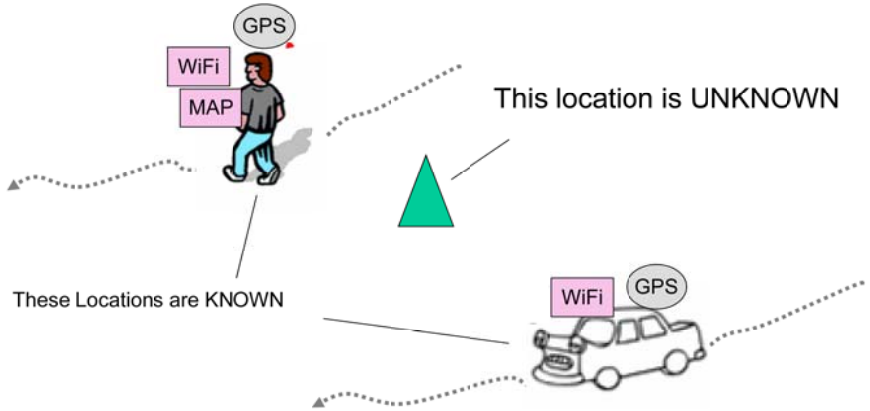
Access Point Position Estimation

This location is UNKNOWN
(except for Public Hotspot APs)



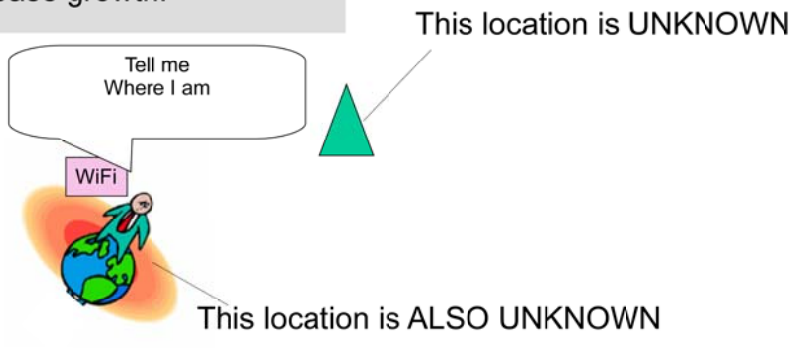
WiFi Access Point

Access Point Estimation (cont.)



Access Point Estimation (cont.)

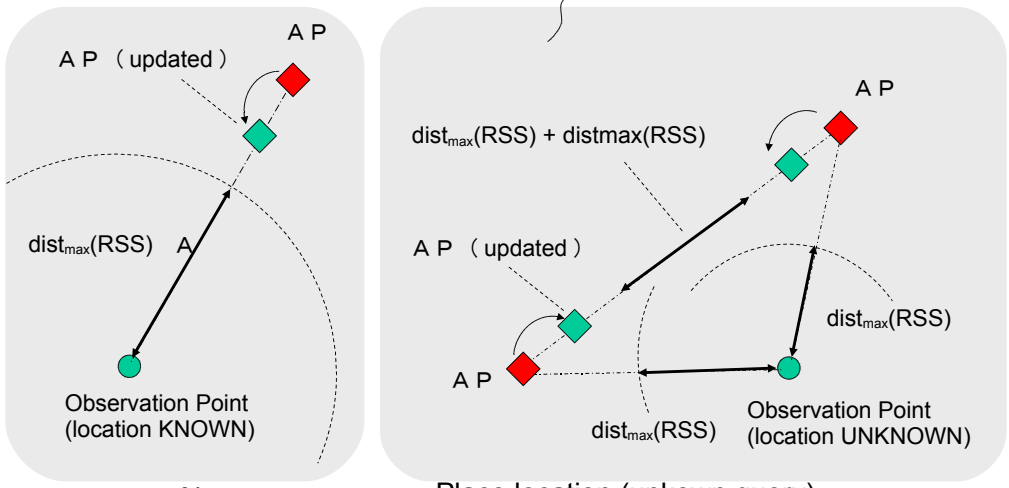
Queries from unknown location also contribute to Database growth.



Realworld Folksonomy

Database Improvement from accesses

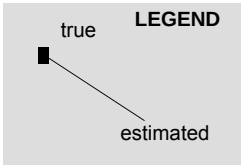
analogy of "co-occurrence (共起関係) in text-mining



1 %

Place location (unkown query) 9 9 %

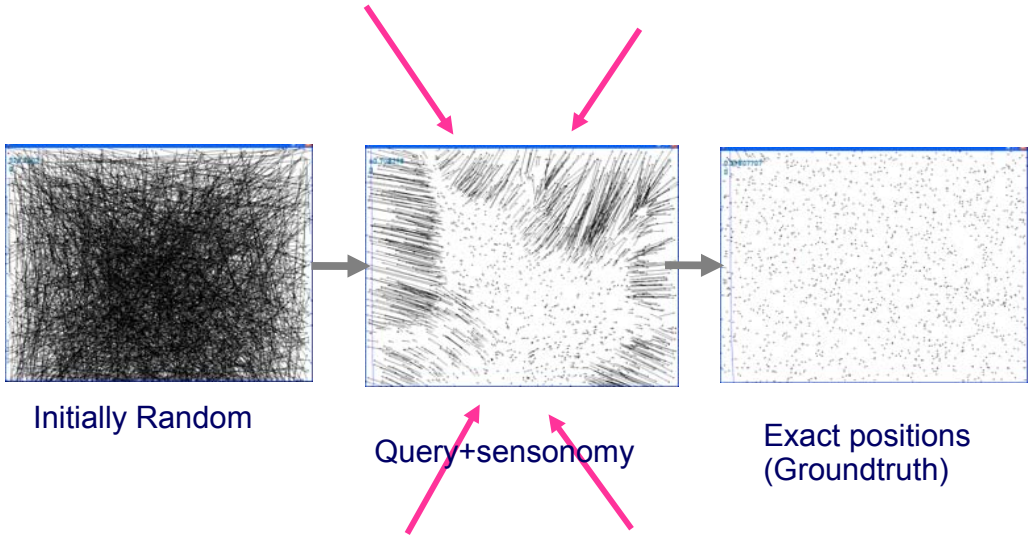
A Simulation

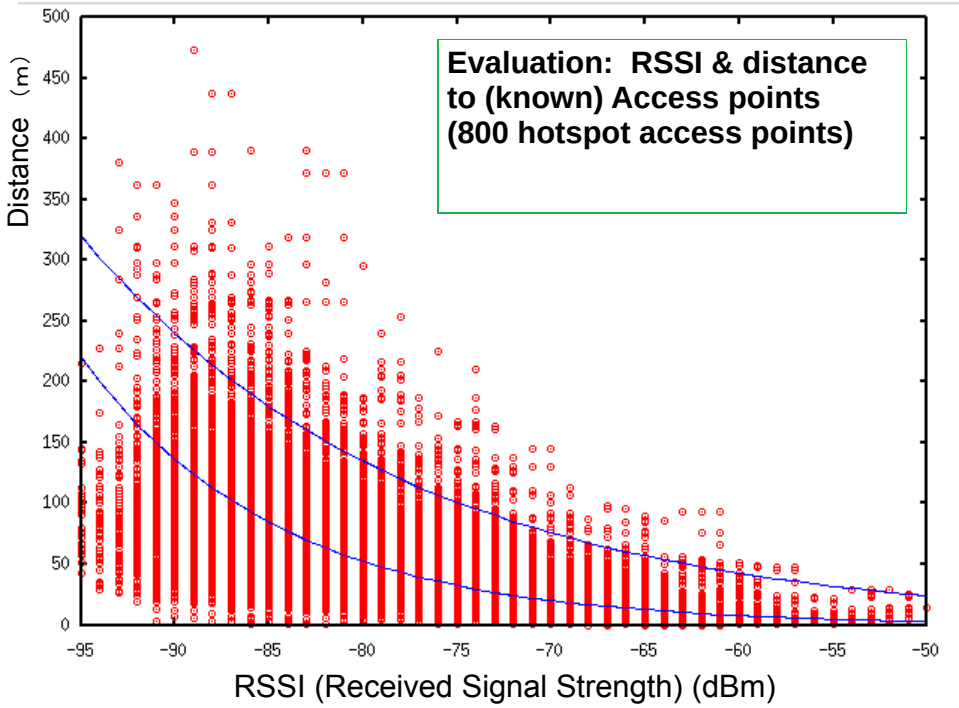


Real Data:

95.2% query
4.8% register

- 2000 APs
  - 1% known position (seeds)
  - 99% unknown positions
- User Info
  - 99% just query
  - 1% with current position
- Very Simple Update Algorithm
- Note:
  - Users don't know AP positions

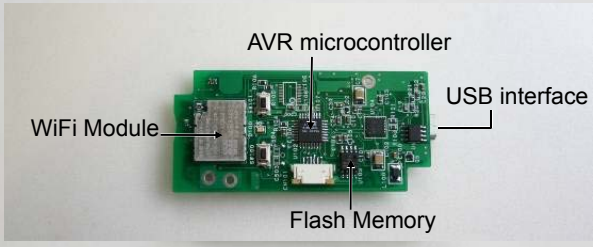




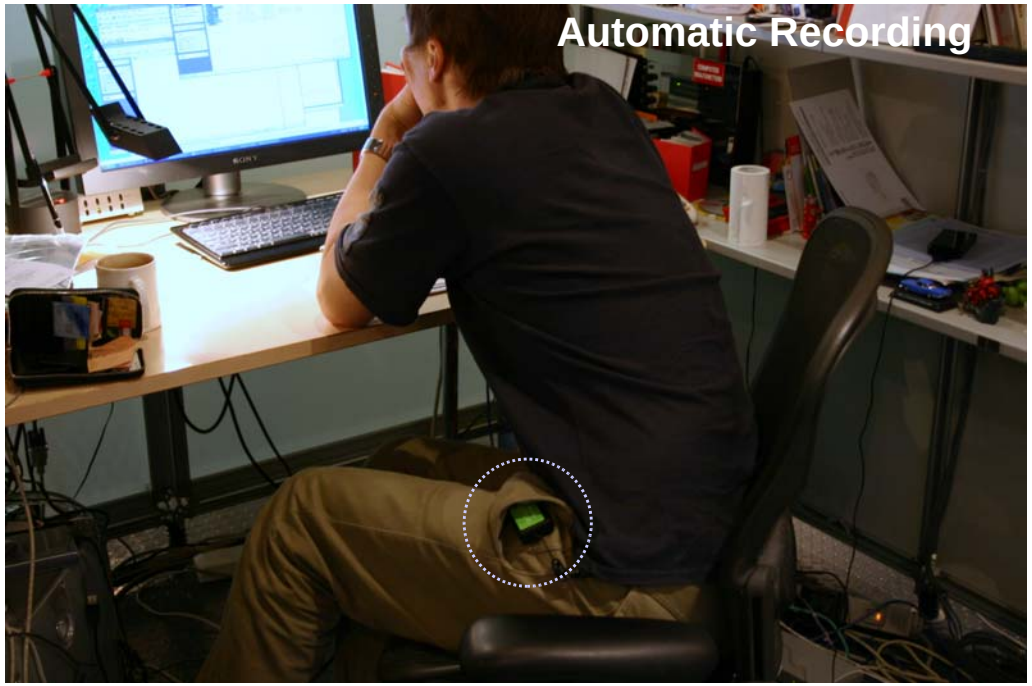
LifeTag

- **Location Life Logging**
  - **WiFi positioning**
  - no operation, just carry
  - works indoors & outdoors
- **WHEN = WHERE**
  - **memory support**
  - **life pattern analysis**
  - **activity prediction**

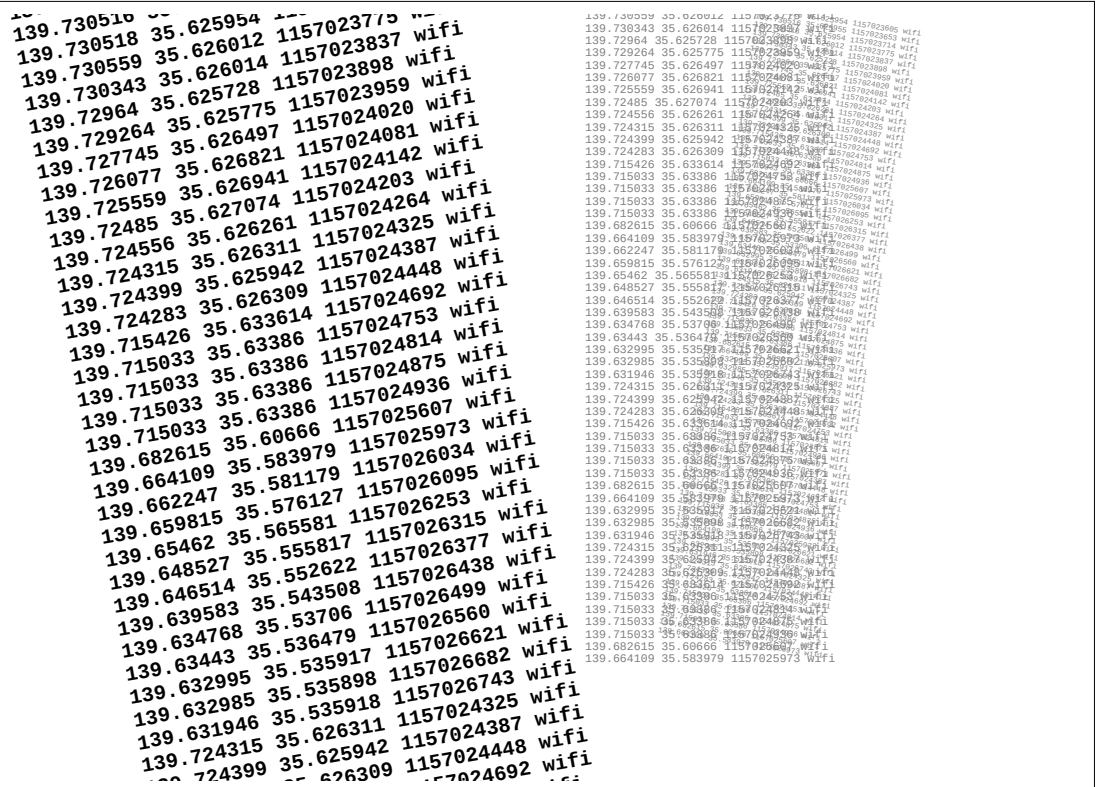
LifeTag

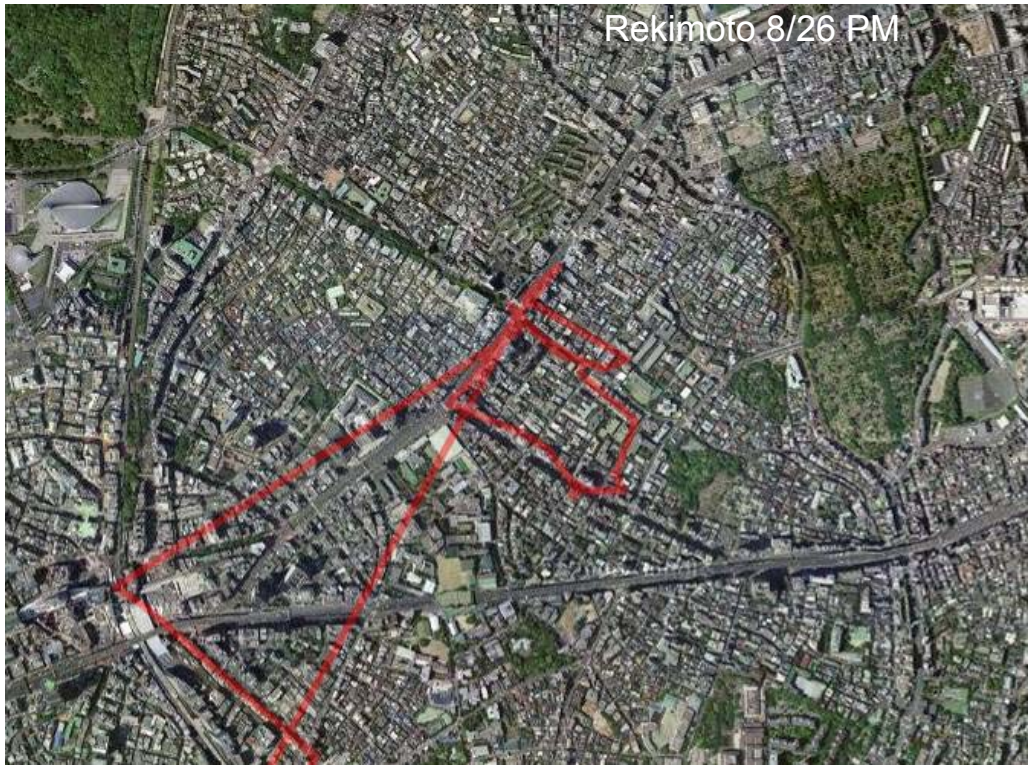


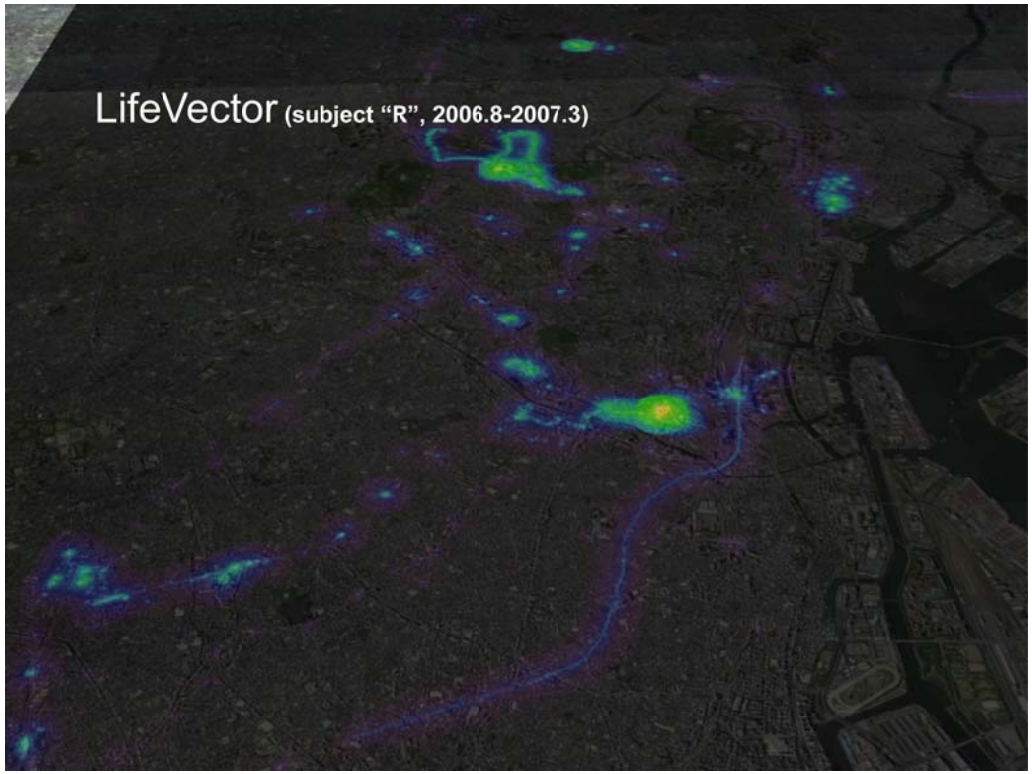
	GPS Logging	WiFi Logging
		
Indoor location	NO	YES
Intermittent operation (for power saving)	NO	YES
Battery lifetime	10 hours	1 week

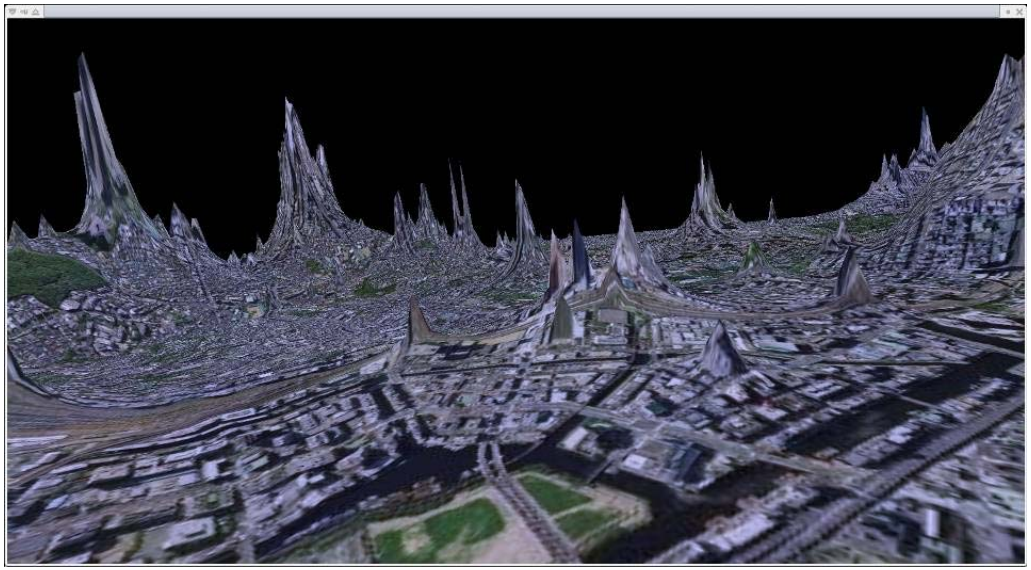


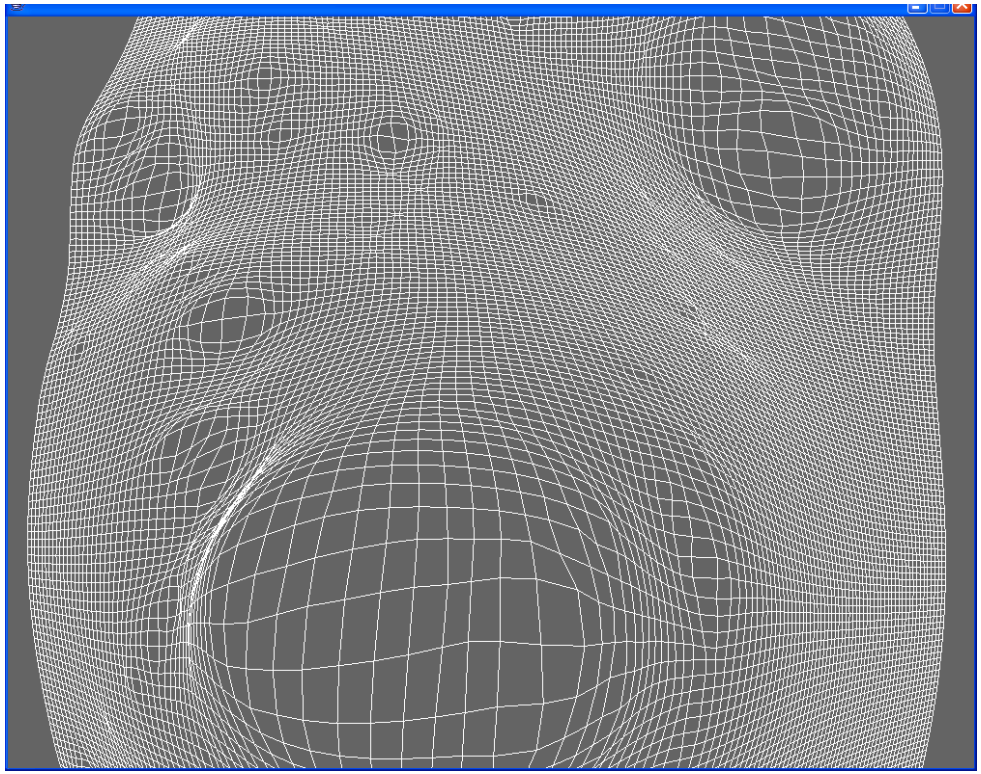

Life Bookmarking


139.730518 35.625954
139.730518 35.626012 1157023775 wifi
139.730559 35.626014 1157023837 wifi
139.730343 35.626014 1157023898 wifi
139.72964 35.625728 1157023959 wifi
139.729264 35.625775 1157024020 wifi
139.727745 35.626497 1157024081 wifi
139.726077 35.626821 1157024142 wifi
139.725559 35.626941 1157024203 wifi
139.72485 35.627074 1157024264 wifi
139.724556 35.626261 1157024325 wifi
139.724315 35.626311 1157024387 wifi
139.724399 35.625942 1157024448 wifi
139.724283 35.626309 1157024692 wifi
139.715426 35.633614 1157024753 wifi
139.715033 35.63386 1157024814 wifi
139.715033 35.63386 1157024875 wifi
139.715033 35.63386 1157024936 wifi
139.715033 35.63386 1157025607 wifi
139.682615 35.60666 1157025973 wifi
139.664109 35.583979 1157026034 wifi
139.662247 35.581179 1157026095 wifi
139.659815 35.576127 1157026253 wifi
139.65462 35.565581 1157026315 wifi
139.648527 35.555817 1157026377 wifi
139.646514 35.552622 1157026438 wifi
139.639583 35.543508 1157026499 wifi
139.634768 35.53706 1157026560 wifi
139.63443 35.536479 1157026621 wifi
139.632995 35.535917 1157026682 wifi
139.632985 35.535898 1157026743 wifi
139.631946 35.535918 1157024325 wifi
139.724315 35.626311 1157024387 wifi
139.724399 35.625942 1157024448 wifi

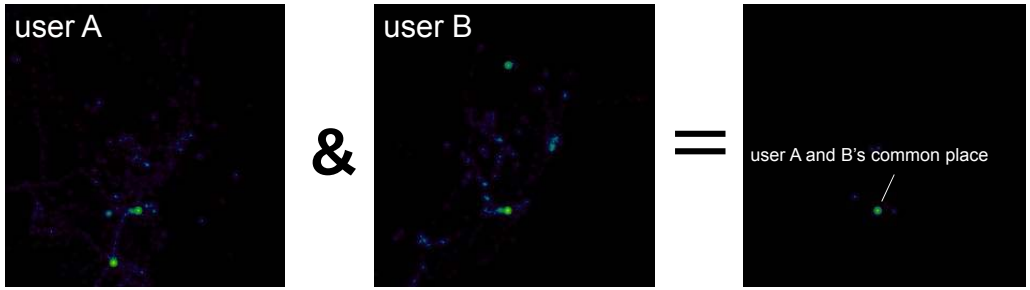

Rekimoto 8/26 PM


LifeVector (subject “R”, 2006.8-2007.3)

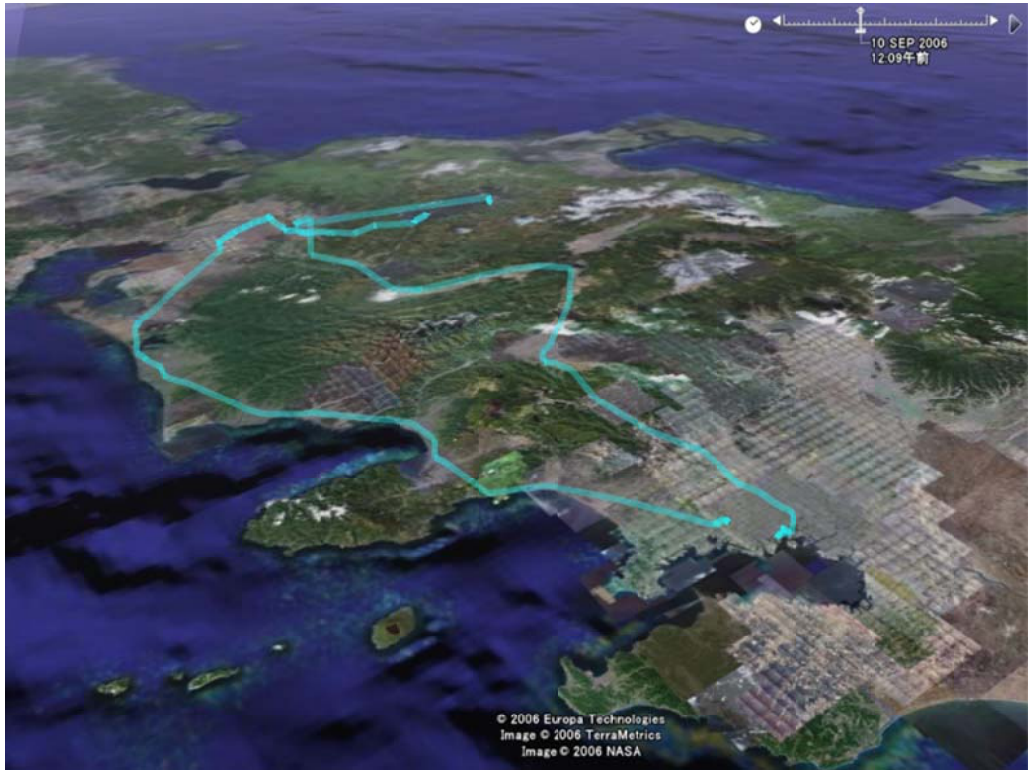




Experience Arithmetic



Autonomous Traceability



Summary

- PlaceEngine : location everywhere
- Sensonomy = Realworld + Folksonomy
- LifeTag: locaiton life-logging
  - People's complete location log
  - Every object will have location log.

- Privacy