INF555

Fundamentals of 3D

183	196	199	200	214	215	118	226	98	104
208	194	200	226	157	88	76	157	0	43
209	214	199	182	91	71	59	173	217	177
214	214	175	150	88	71	59	138	217	214
193	215	208	199	113	60	55	52	244	199
138	105	137	152	215	109	71	44	70	168
137	120	105	102	104	157	244	137	75	68
140	123	120	123	105	105	120	137	244	199
138	118	139	109	108	138	138	138	138	168
109	114	121	121	138	119	119	138	138	152

Lecture 1: Abstract Data Structures

Stacks, queues and hashing with applications in visual computing

Abstract Data Structures

- Concepts encapsulated into an **interface**
- Multiple implementations (various trade-offs)

Good for code reuse (APIs):
Actual implementation is hidden

Abstract Data Structures: Stack

Interface (method prototypes):

isEmpty
push(Element)
pop (or pull)

For example, two implementations:

- Array
- Linked list

```
public static void main(String [] args)
{
	Stack myStack=new Stack();
	int i;

	for(i=0;i<10;i++)
		myStack.Push(i);

	for(i=0;i<15;i++)

System.out.println(myStack.Pull());

}
```

Stack (array)

```
class StackArray
{
int nbmax;
int index;
int [ ] array;

// Constructors
StackArray(int n)
{
this.nbmax=n;
array=new int[nbmax]; index=-1;
System.out.println("Succesfully created a stack array object...");
}

// Methods: INTERFACE
void Push(int element)
{
if (index<nbmax-1)
      array[++index]=element;   }

int Pull()
{
if (index>=0 ) return array[index--];
else return -1;
}

}
```

Stack (linked list)

```
class Stack
{
List list;

Stack()
{
list=null;
}

void Push(int el)
{
if (list!=null)
        list=list.insertHead(el);
        else
        list=new List(el,null);
}

int Pull()
{int val;
if (list!=null)
    {val=list.element;
        list=list.next;}
        else val=-1;

return val;
}
}
```

```
class List
{
int element;
List next;

// Constructor
List(int el, List tail)
{
this.element=el;
this.next=tail;
}

List insertHead(int el)
{
return new List(el,this);
}

}
```

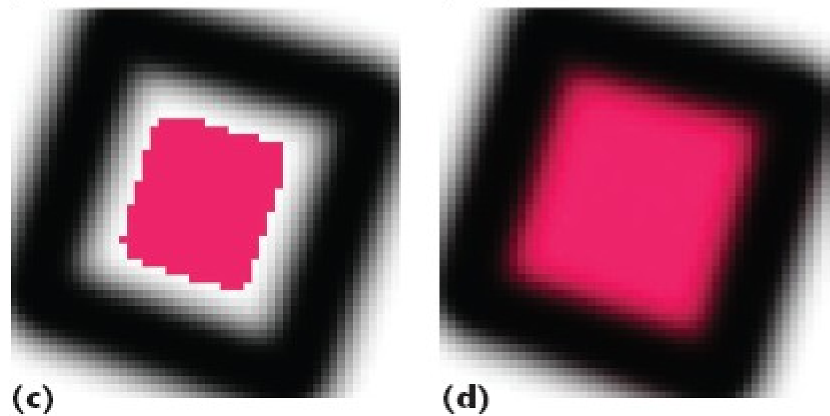
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Stack.html

Method Summary	
boolean	empty() Tests if this stack is empty.
Object	peek() Looks at the object at the top of this stack without removing it from the stack.
Object	pop() Removes the object at the top of this stack and returns that object as the value of this function.
Object	push(Object item) Pushes an item onto the top of this stack.
int	search(Object o) Returns the 1-based position where an object is on this stack.

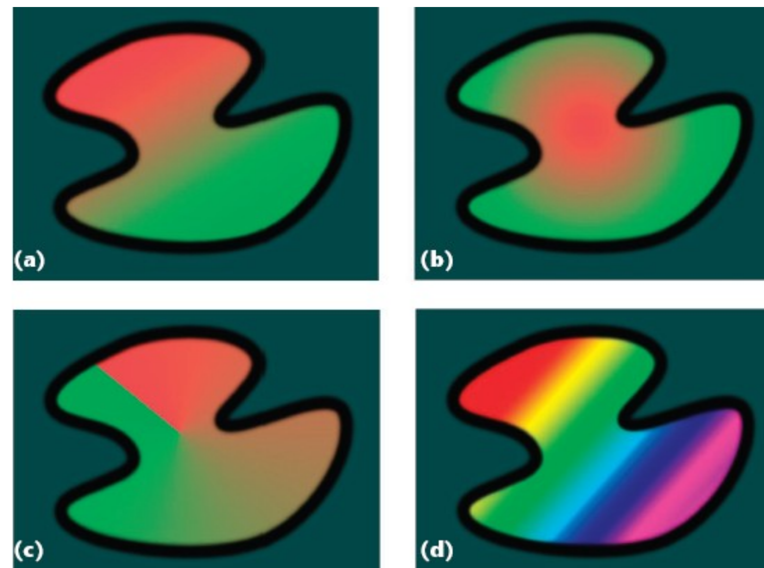
Area Flood Filling

Area Flood Filling

Tint fill+pattern fill



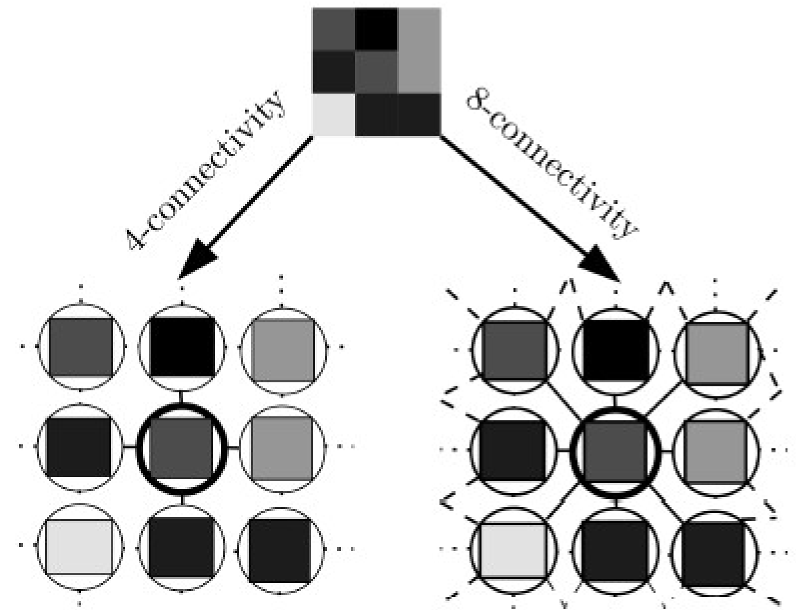
Tint filling



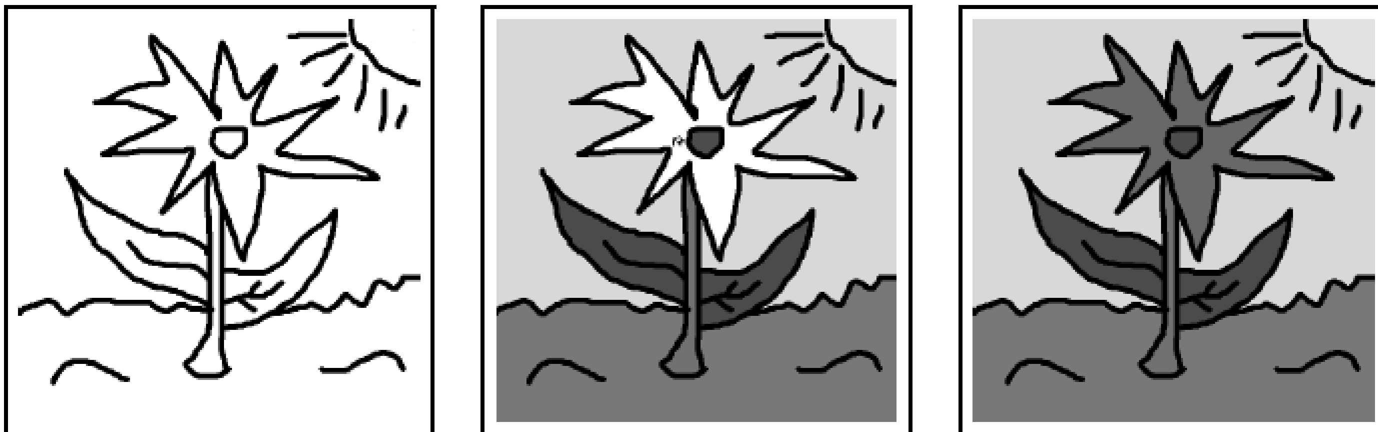
Gradient filling

Area Flood Filling

Image (pixel) connectivity: 4- or 8-connectivity
(but also hexagonal, etc.)



Example of flood filling (cartoon animation, etc.)



http://www-graphics.stanford.edu/~levoy/sands_award.html

Area Flood Filling: Recursive Algorithm

FloodFillRecursive($\mathbf{I}, p, C_B, C_F$)
1. $\mathbf{I}[p] \leftarrow C_F$
2. **for** $q \in C_4(p)$
3. **do**
4. **if** $\mathbf{I}[q] = C_B$
5. **then** FloodFillRecursive($\mathbf{I}, q, C_B, C_F$)

Use an abstract data-structure: Stack

- new() $\rightarrow \emptyset$ returns an empty stack,
- pop(push(S, e)) $\rightarrow \mathsf{S}$,
- top(push(S, e)) $\rightarrow e$.

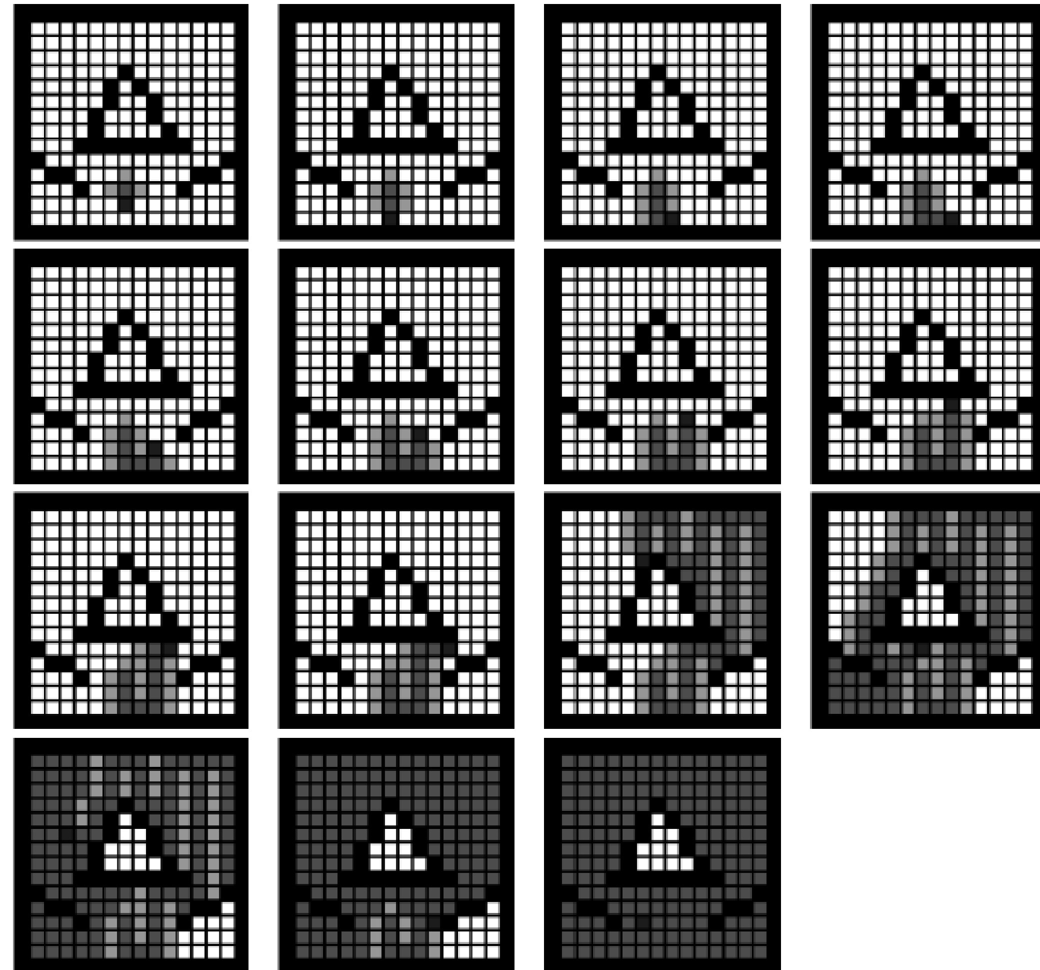
- isEmpty(new()) $\rightarrow$ true,
- isEmpty(push(S, e)) $\rightarrow$ false,
- isEmpty(pop(push(S, e)) $\rightarrow$ isEmpty(S).

Area flood filling using a stack

Handle recursion explicitly with the stack

$\text{FLOODFILLSTACK}(\mathbf{I}, p, C_B, C_F)$

1. ◁ S is a stack data structure ▷
2. $\mathsf{S} \leftarrow \emptyset$
3. $\mathbf{I}[p] \leftarrow C_F$
4. $\mathsf{S}.\text{push}(p)$
5. **while** $\mathsf{S} \neq \emptyset$
6. **do**
7. $p \leftarrow \mathsf{S}.\text{pop}()$
8. **for** $q \in C_4(p)$
9. **do**
10. ◁ 4-neighborhood pixels ▷
11. **if** $\mathbf{I}[q] = C_B$
12. **then** $\mathbf{I}[q] = C_F$
13. $\mathsf{S}.\text{push}(q)$



Depth first order

Area flood filling using a queue

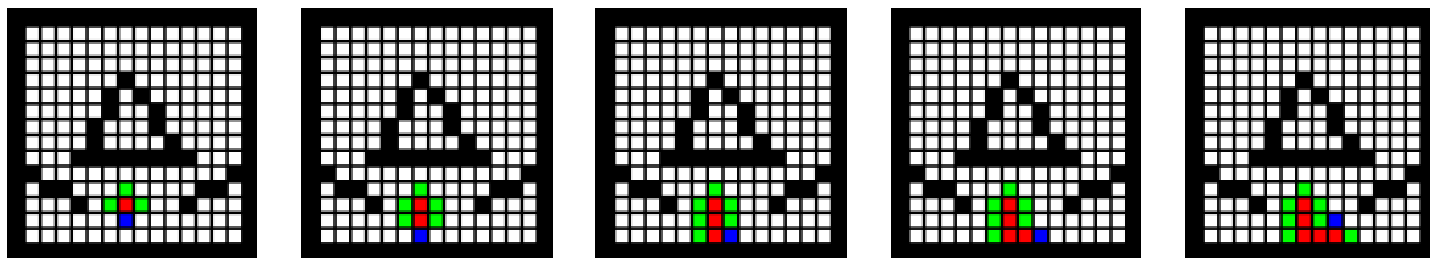
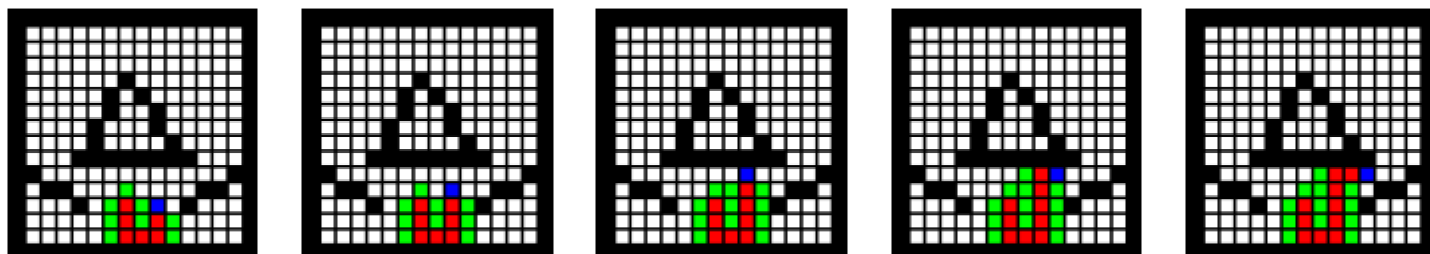
Properties of abstract queues:

- new() $\rightarrow \emptyset$ returns an empty queue,
- head(enqueue(new(), e)) $\rightarrow e$,
- dequeue(enqueue(new(), e)) $\rightarrow$ new(),
- head(enqueue(enqueue(Q, e), f)) $\rightarrow$ head(enqueue(Q, e)) (FIFO definition),
- dequeue(enqueue(enqueue(Q, f), e)) $\rightarrow$ enqueue(dequeue(enqueue(Q, f)), e).

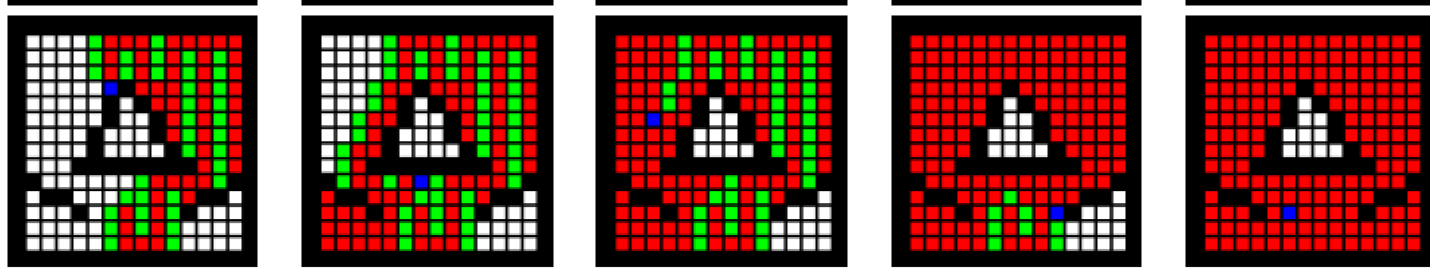
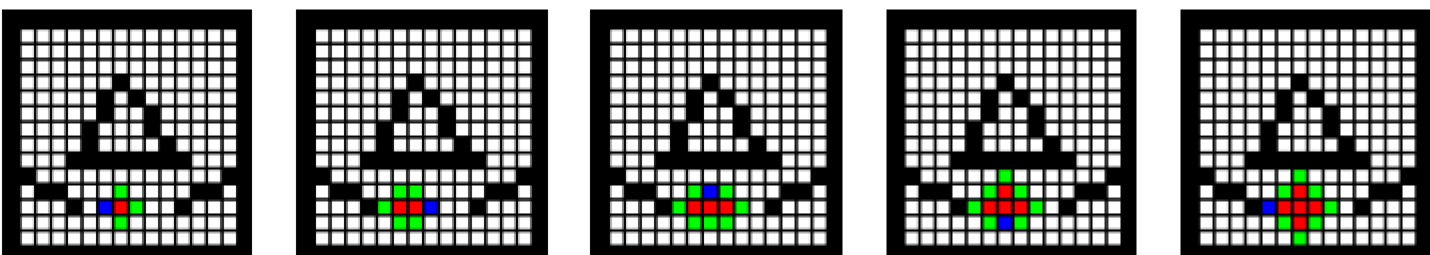
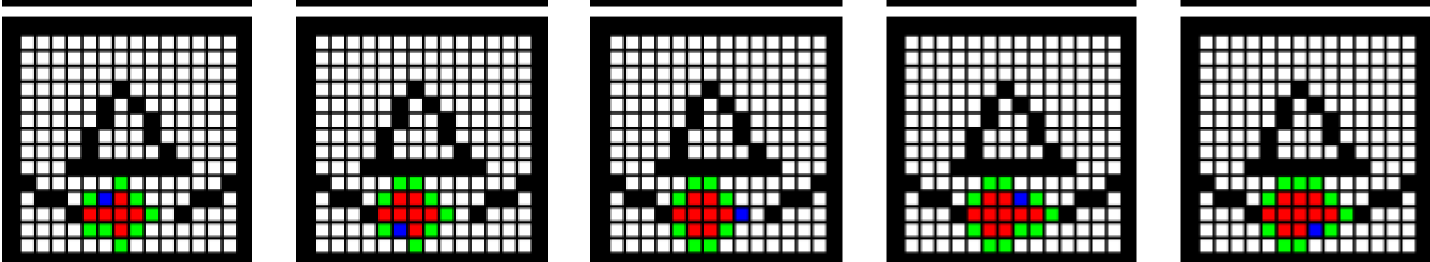
FloodFillQueue($\mathbf{I}, p, C_B, C_F$)

1. ◁ Create empty queue Q ▷
2. Q $\leftarrow \emptyset$
3. $\mathbf{I}[p] \leftarrow C_F$
4. Q.enqueue(p)
5. **while** Q $\neq \emptyset$
6.   **do** $p \leftarrow$ Q.head()
7.   **for** $q \in C_4(p)$
8.   **do if** $\mathbf{I}[q] = C_B$
9.   **then** $\mathbf{I}[q] \leftarrow C_F$
10.   Q.enqueue(q)
11.   Q.dequeue()

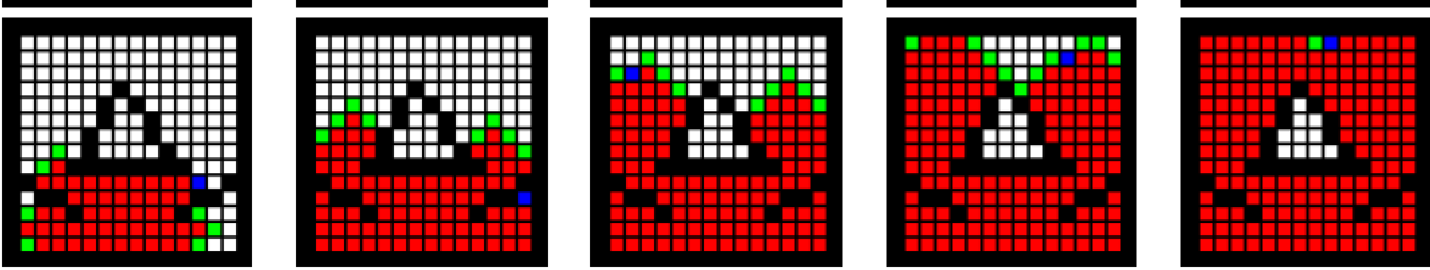
Breadth first order

Stack

Queue



Abstract dictionaries

Store items indexed by keys
Axiomatic semantic:

- $\mathrm{new}() \to \emptyset$ returns an empty dictionary,
- $\mathrm{find}(\mathrm{insert}(\mathsf{D}, k, v), k) \to v$,
- $\mathrm{find}(\mathrm{insert}(\mathsf{D}, j, v), k) \to \mathrm{find}(\mathsf{D}, k)$ if $k \neq j$,
- $\mathrm{delete}(\mathrm{new}(), k) \to \mathrm{new}()$,
- $\mathrm{delete}(\mathrm{insert}(\mathsf{D}, k, v), k) \to \mathrm{delete}(\mathsf{D}, k)$,
- $\mathrm{delete}(\mathrm{insert}(\mathsf{D}, j, v), k) \to \mathrm{insert}(\mathrm{delete}(\mathsf{D}, k), j, v)$ if $k \neq j$.

Unordered (collection) or ordered dictionaries
(=augmented dictionaries)

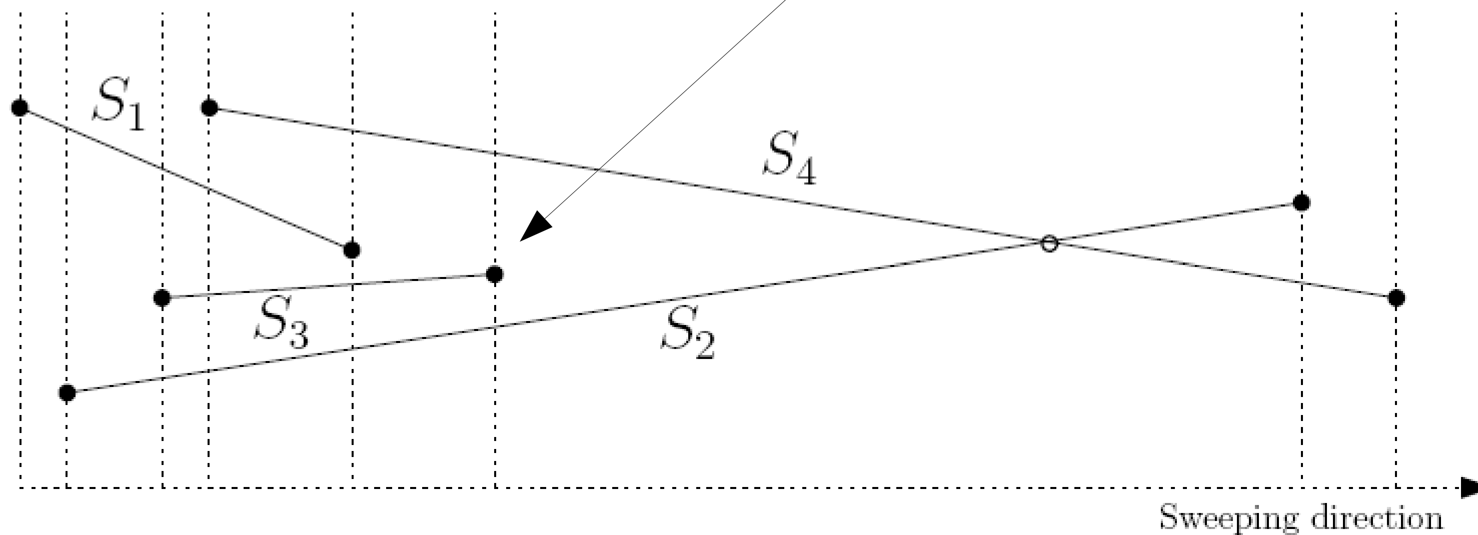
Dictionaries:
Detecting Any Segment Pair Intersection

Naive quadratic time algorithm:

NaiveSegmentIntersection($\mathcal{S} = \{S_1 = \{A_1, B_1\}, ..., S_n = \{A_n, B_n\}\}$)
1. ◁ Test pairwise segments for possible intersections ▷
2. $s \leftarrow 0$;
3. **for** $i \leftarrow 1$ **to** n
4.   **do for** $j \leftarrow i + 1$ **to** n
5.   **do if** IntersectSegment(S_i, S_j)
6.   **then** $s = s + 1$
7.   ReportIntersectionSegmentPair(S_i, S_j)
8. **return** s

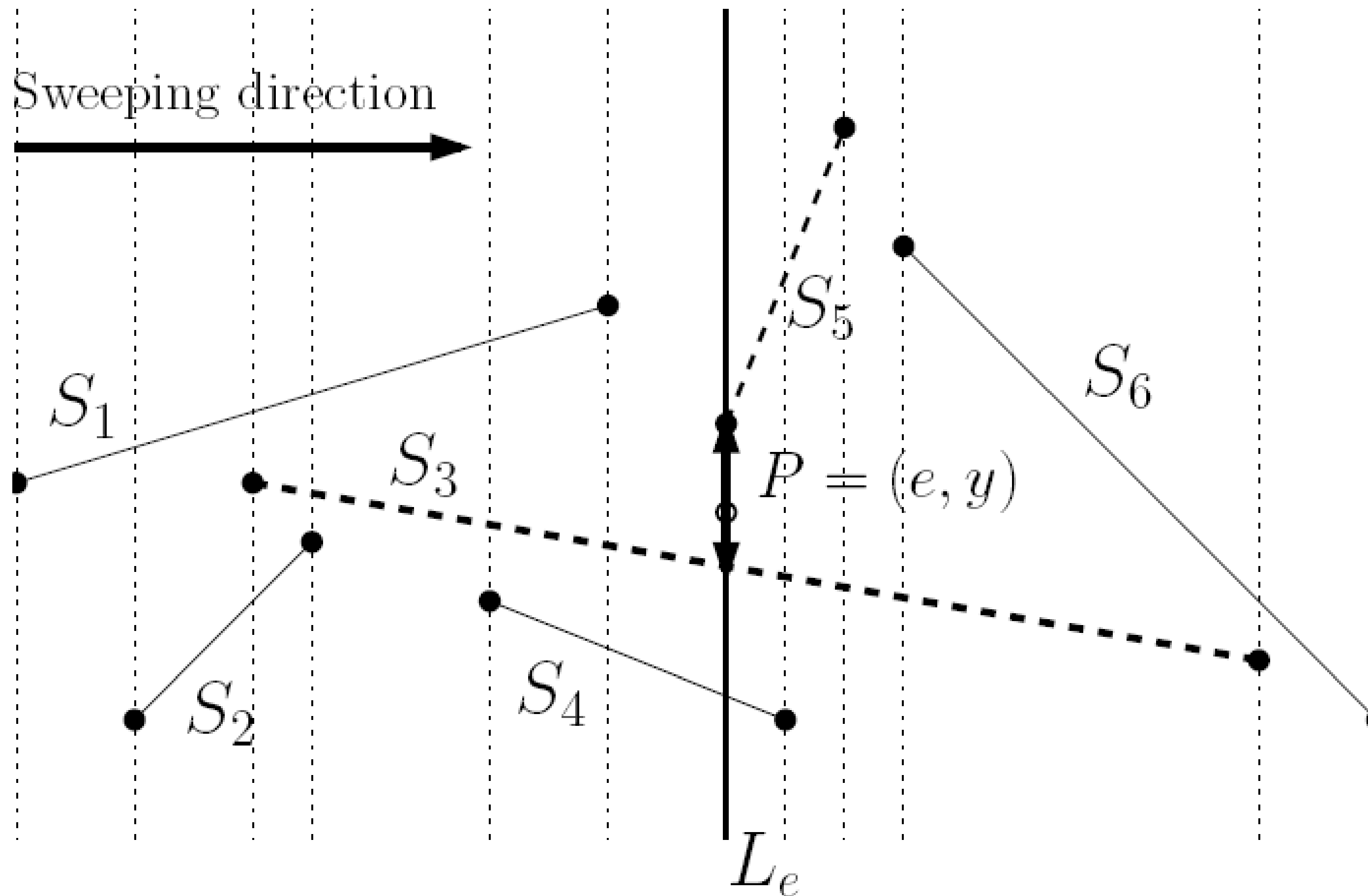
A structural property

S2 intersects S4: At some position, they become y-adjacent



Sweeping line: combinatorial algorithm: 2n extremities only

Ordered dictionary



For a given query (e,y) report the line segment immediately:

- above and
- below (e,y)

Dynamic insertion/deletion of line segment extremities

Implementation using binary trees (red-black, AVL, etc.)

Shamos-Hoey intersection detection algorithm

```
DetectSegmentIntersection(𝒮 = {S_1 = {A_1, B_1}, ..., S_n = {A_n, B_n}})
 1.  ◁ Test whether there exists an intersection between any pair of segments of 𝒮 ▷
 2.  ℰ ← {A_i.x, B_i.x | i ∈ [[1, n]]}
 3.  T ← SortIncreasingOrder(ℰ)
 4.  ◁ Initialize an empty dictionary D ▷
 5.  D ← ∅
 6.  for i ← 1 to 2n
 7.      do
 8.           ◁ Let S be the segment supporting endpoint x-coordinate T[i] ▷
 9.           switch
10.              case LeftEndpoint(T[i]) :
11.                    D.insert(S)
12.                    if Intersect(D.above(S), S) OR Intersect(D.below(S), S)
13.                       then return true
14.              case RightEndpoint(T[i]) :
15.                    D.delete(S)
16.                    if Intersect(D.above(S), D.below(S))
17.                       then return true
18.  return false
```

Priority queues:

- $\text{new}() \to \emptyset$ returns an empty priority queue Q,
- $\text{head}(\text{insert}(\text{new}(), v)) = v$,
- $\text{deleteHead}(\text{insert}(\text{new}(), v)) = \text{new}()$,
- $\text{head}(\text{insert}(\text{insert}(\mathsf{Q}, w), v)) =$

$$\begin{cases} v & \text{if priority}(v) < \text{priority}(\text{head}(\text{insert}(\mathsf{Q}, w))), \\ \text{head}(\text{insert}(\mathsf{Q}, w)) & \text{otherwise.} \end{cases},$$

- $\text{deleteHead}(\text{insert}(\text{insert}(\mathsf{Q}, w), v)) =$

$$\begin{cases} \text{insert}(\mathsf{Q}, w) & \text{if priority}(v) < \text{priority}(\text{head}(\text{insert}(\mathsf{Q}, w))), \\ \text{insert}(v, \text{deleteHead}(\text{insert}(\mathsf{Q}, w))) & \text{otherwise.} \end{cases}$$

(INF311: Implementation using implicit heaps stored in an array)

For the line sweep: 3 kinds of events:
- Begin extremity
- End extremity
- Intersection point

Bentley-Ottman algorithm

Complexity: O((n+I)log n)

Initialization:

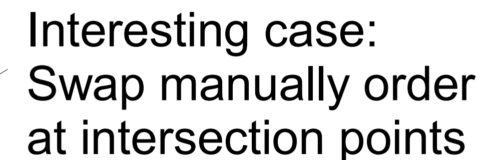
FindSegmentIntersections($\mathcal{S} = \{\{A_1, B_1\}, ..., \{A_n, B_n\}\}$)

1. ◁ Bentley-Ottman's algorithm for reporting all intersection points ▷
2. ◁ Q is a priority queue ▷
3. $k \leftarrow 0$
4. Q $\leftarrow \emptyset$
5. ◁ Insert the $2n$ line segment endpoints ▷
6. ◁ sorted according to their x-abscissae ▷
7. **for** $i \leftarrow 1$ **to** n
8.   **do**
9.   Q.insert(A_i, i)
10.   Q.insert(B_i, i)
11. ◁ While there are remaining events: ▷
12. ◁ Handle events and update the priority queue accordingly ▷
13. **while** Q $\neq \emptyset$
14.   **do**
15. $P \leftarrow$ Q.head()
16.   ◁ HandleEventPoint is described thereafter (see next page). ▷
17.   HandleEventPoint(P)

Bentley-Ottman algorithm: Handling events

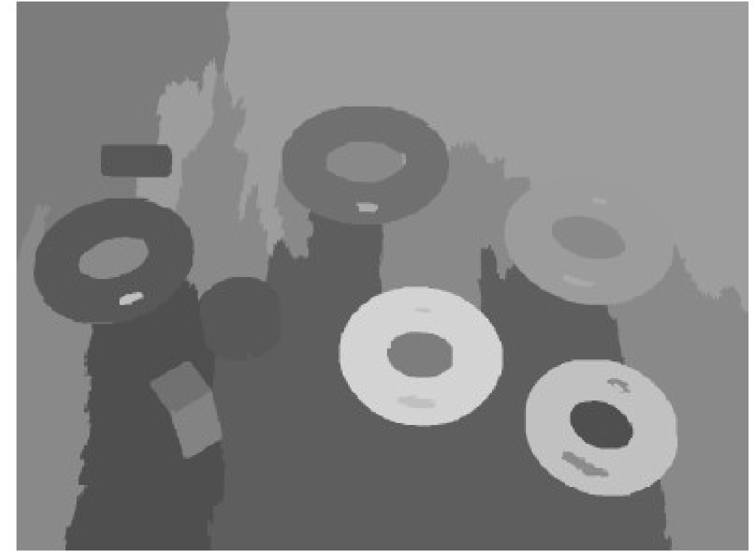
HandleEventPoint(P)

1. ◁ D is a dictionary data structure ▷
2. ◁ Q is a priority queue not storing duplicates (general position assumpti
3. ◁ $x_{S_1 S_2}$ represents the x-coordinate of intersection point $S_1 \cap S_2$ ▷
4. ◁ Let S denote the segment which contains endpoint P ▷
5. **switch**
6.   **case** LeftEndPoint(P) :
7.   D.insert(S)
8. $S_1 \leftarrow$ D.above(S)
9. $S_2 \leftarrow$ D.below(S)
10.   **if** $S_1 \cap S \neq \emptyset$
11.   **then** $k \leftarrow k + 1$
12.   Q.insert($x_{S_1 S}$)
13.   **if** $S_2 \cap S \neq \emptyset$
14.   **then** $k \leftarrow k + 1$
15.   Q.insert($x_{S_2 S}$)
16.   **case** RightEndPoint(P) :
17. $S_1 \leftarrow$ D.above(S)
18. $S_2 \leftarrow$ D.below(S)
19.   ◁ Avoid counting an already detected intersection point ▷
20.   **if** $S_1 \cap S_2 \neq \emptyset$ and $x_{S_1 S_2} \geq x_P$
21.   **then** $k \leftarrow k + 1$
22.   Q.insert($x_{S_1 S_2}$)
23.   D.delete(S)
24.   **case** IntersectionPoint(P) :
25.   ◁ S_1 and S_2 are the segments such that $P = S_1 \cap S_2$ ▷
26. $S_3 \leftarrow$ D.above(S_1)
27. $S_4 \leftarrow$ D.below(S_2)
28.   **if** $S_3 \cap S_2 \neq \emptyset$
29.   **then** $k \leftarrow k + 1$
30.   Q.insert($x_{S_3 S_2}$)
31.   **if** $S_1 \cap S_4 \neq \emptyset$
32.   **then** $k \leftarrow k + 1$
33.   Q.insert($x_{S_1 S_4}$)
34.   SwapDictionaryItems(S_1, S_2, D)



Union-Find abstract data-structures

Manage disjoint sets very efficiently under union

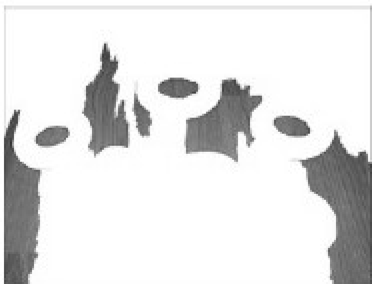


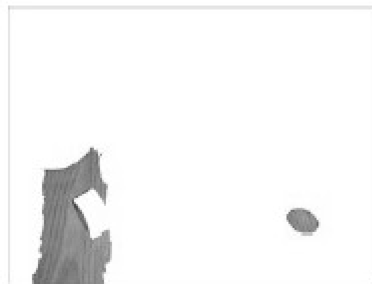
Example: Image segmentation (let a robot see!)

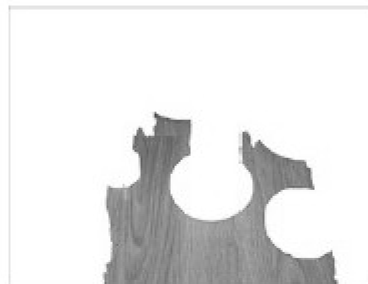
RegionMerging($\mathbf{I}$)

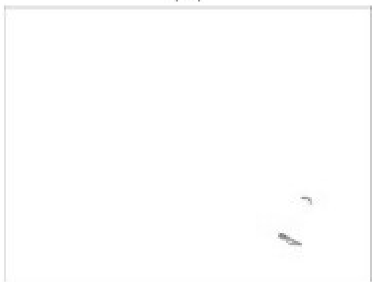
1. $\triangleleft$ Test merging predicates on sorted C_4 adjacent pixel pairs $\triangleright$
2. $\mathbf{P} \leftarrow \{P_l = (p_l, q_l, C_l = |\mathbf{I}(q_l) - \mathbf{I}(p_l)|) \mid \text{such that } q_l \in C_4(p_l)\}$
3. $\triangleleft$ Sort in increasing order according to C_l $\triangleright$
4. Sort($\mathbf{P}$)
5. **for** $i \leftarrow 1$ **to** $|\mathbf{P}|$
6. **do**
7. **if** region(p_l) $\neq$ region(q_l)
8. **then** $\triangleleft$ MergePredicate(r_1, r_2)=PerceptualUnit($r_1 \cup r_2$) $\triangleright$
9. **if** MergePredicate(region(p_l), region(q_l))
10. **then** Merge(region(p_l), region(q_l))

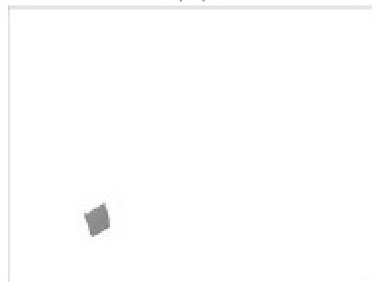
Union-Find abstract data-structures

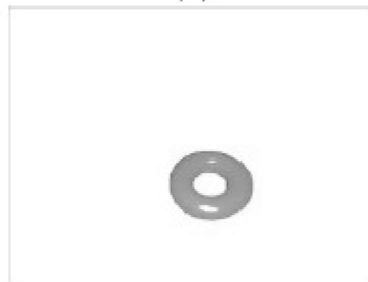

(1)


(2)

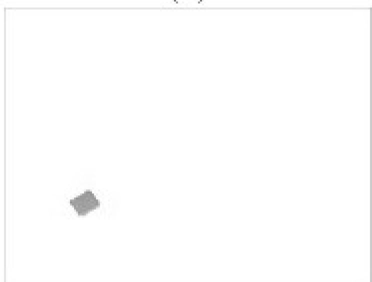

(3)

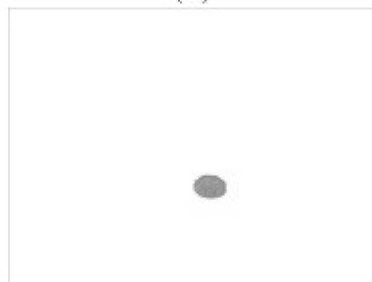

(4)

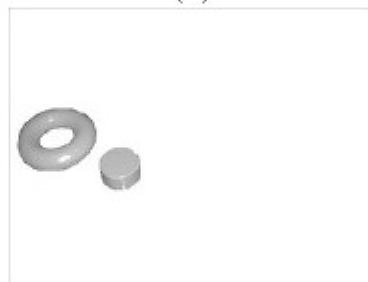

(5)


(6)

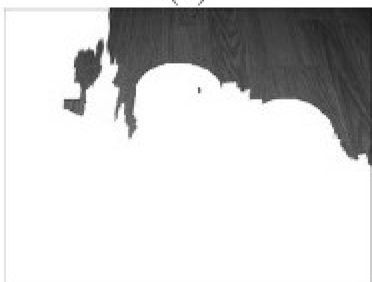
Each region is a set
Initially, each pixel defines a region

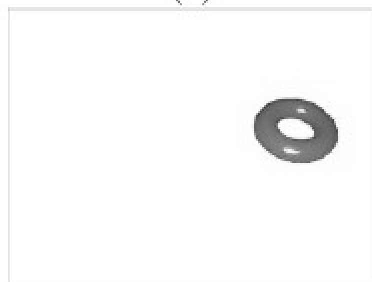

(7)

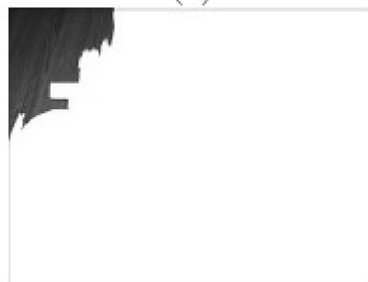

(8)


(9)

Merging regions=merging disjoint sets


(10)

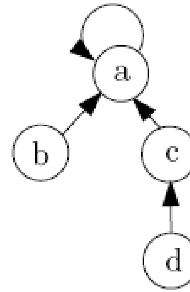

(11)


(12)

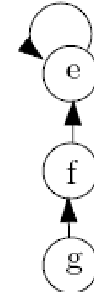
For a given element,
find the set that contains it

Union-Find abstract data-structures

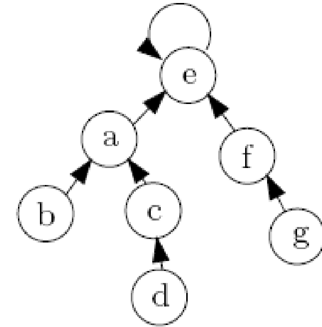
MakeSet(x)
1. parent(x) $\leftarrow x$
2. rank(x) $\leftarrow 0$



$\mathcal{S}_1 = \{a, b, c, d\}$



$\mathcal{S}_2 = \{e, f, g\}$

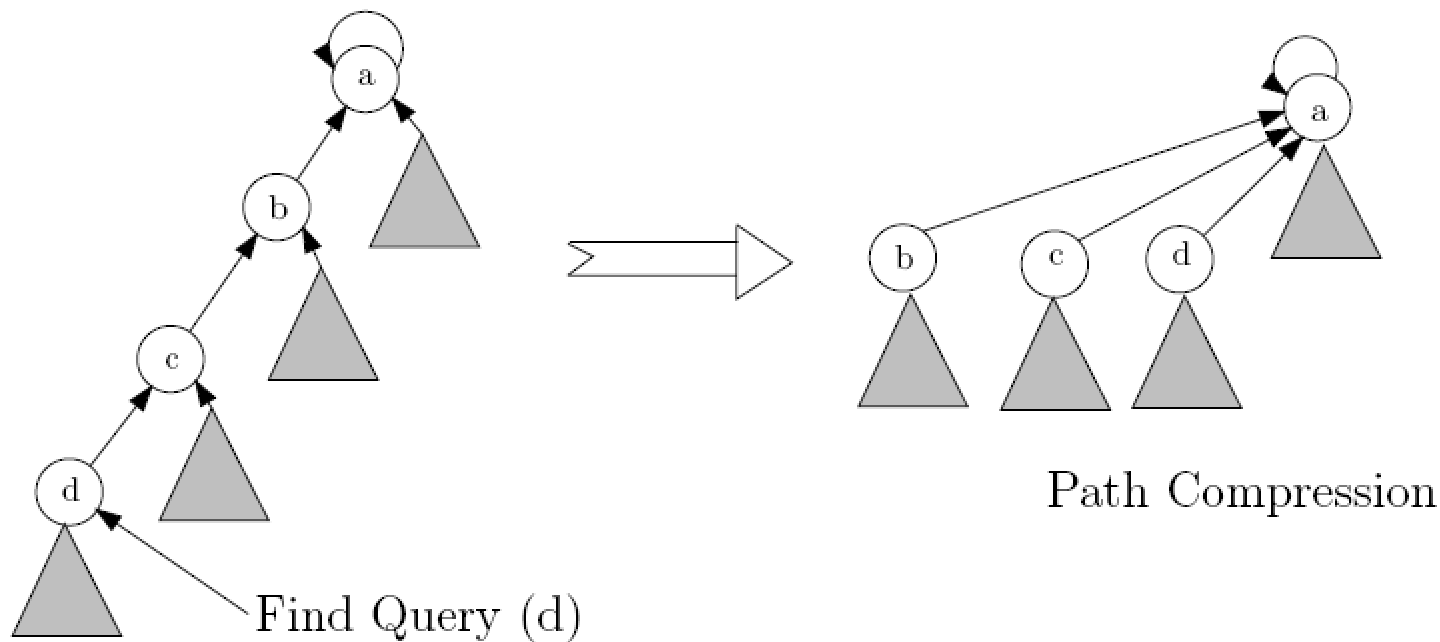


$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$

Find(x)
1. ◁ Walk to the leader element ▷
2. **while** $x \neq$ parent(x)
3. **do** $x \leftarrow$ parent(x)
4. **return** x

Union(x, y)
1. ◁ Union by rank and path compression ▷
2. $x_r \leftarrow$ Find(x)
3. $y_r \leftarrow$ Find(y)
4. **if** rank(x_r) $>$ rank(y_r)
5. **then** parent(x_r) $\leftarrow y_r$
6. **else** parent(y_r) $\leftarrow x_r$
7. **if** rank(x_r) $=$ rank(y_r)
8. **then** rank(x_r) $\leftarrow$ rank(x_r) $+ 1$

Path compression: Almost linear-size data-structure (inverse of Ackermann function)



Labelling connected components

Much easier problem than segmentation

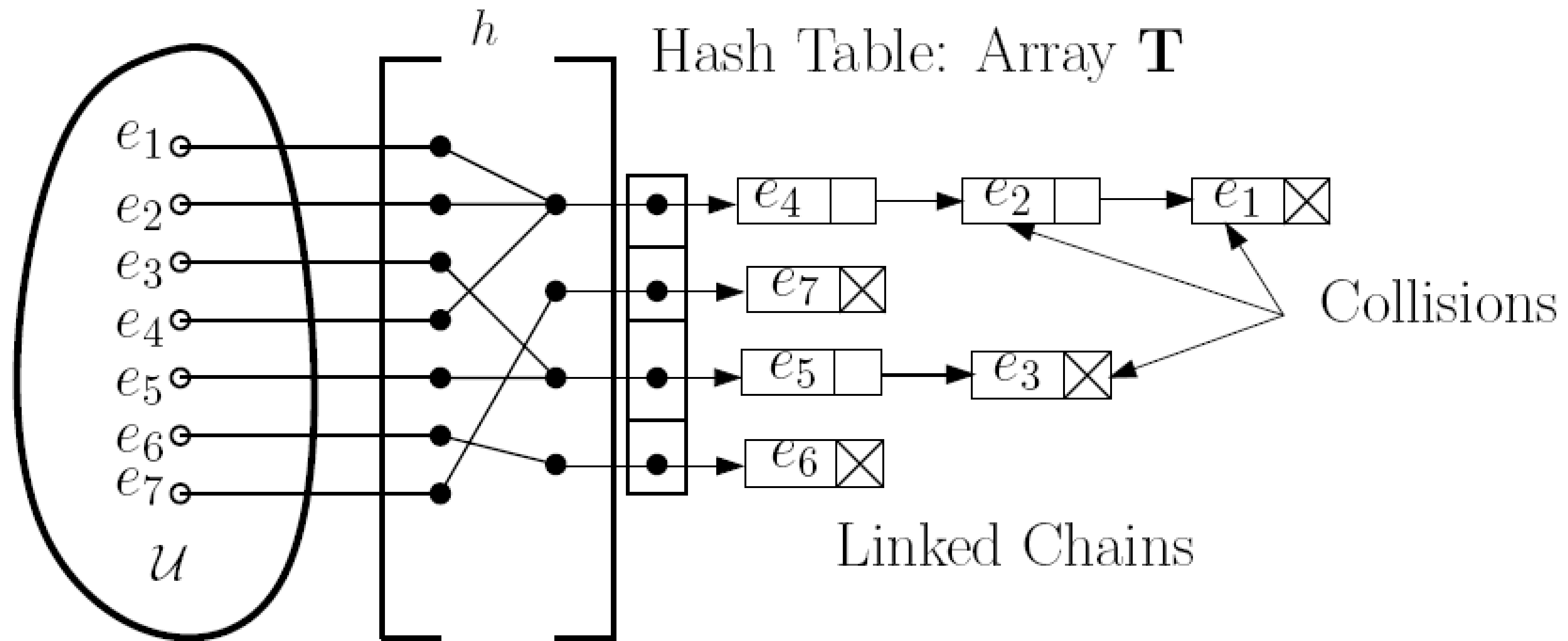
Disjoint set data-structure

Segmentation, clustering et tone mapping

(a) (b) (c)

G R B

Hashing



Hashing and collision (INF311):

- Open address
- Linked chains

Geometric hashing: Preprocessing

For each feature points pair:

- Define a local coordinate basis on this pair
- Compute and quantize all other feature points in this coordinate basis
- Record (model, basis) in a hash table

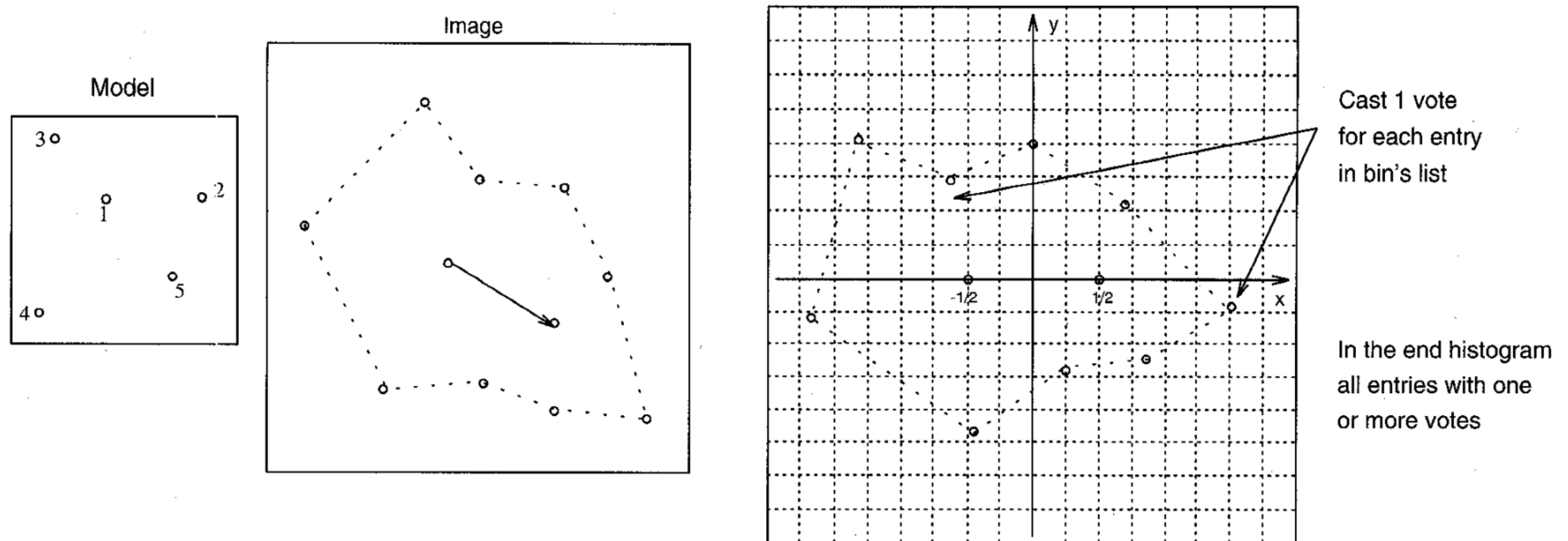
Geometric hashing: Recognition

Pick arbitrary ordered pair:
Compute the other points using this pair as a basis

For all the transformed points, vote all records (model, basis) appear in the corresponding entry in the hash table, and histogram them

Matching candidates: (model, basis) pairs with large number of votes.

Recover the transformation that results in the best least-squares match between all corresponding feature points
Transform the features, and verify against the input image features
(if fails, repeat this procedure with another pair...)



Geometric hashing

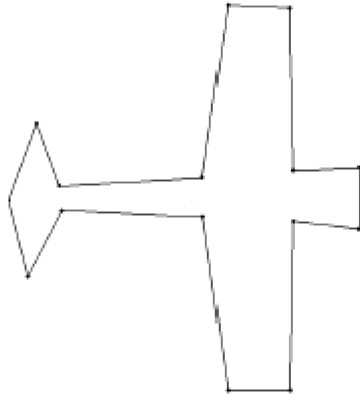
AddToHashTable($\mathbf{H}, M_i, C_i$)

1. $\triangleleft$ Hash table $\triangleright$
2. $\triangleleft$ $\mathcal{C}_i$ denotes the reference frame induced by $C_i = \{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}\}$ $\triangleright$
3. **for** $j \leftarrow 1$ **to** m_i
4. **do**
5. $\triangleleft$ Decompose $P_{i,j}$ onto the local frame $\triangleright$
6. $\triangleleft$ Solve for λ_1 and λ_2: $(\lambda_1, \lambda_2) = \mathcal{C}_{C_i}(P_{i,j})$ $\triangleright$
7. $\mathbf{p_{i,j}} = \frac{\mathbf{p}_{i,1}\mathbf{p}_{i,2}}{2} + \lambda_1 \mathbf{p}_{i,1} + \lambda_2 \mathbf{p}_{i,2}$
8. $\triangleleft$ Add model i is the hash bin of $P_{i,j}$ $\triangleright$
9. $\mathbf{H}[\lambda_1][\lambda_2] = i;$

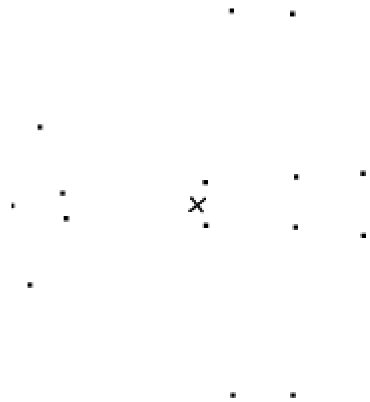
$$\begin{bmatrix} x_{i,j} \\ y_{i,j} \end{bmatrix} - \begin{bmatrix} \frac{x_{i,1}+x_{i,2}}{2} \\ \frac{y_{i,1}+y_{i,2}}{2} \end{bmatrix} = \underbrace{\begin{bmatrix} x_{i,1} & x_{i,2} \\ y_{i,1} & y_{i,2} \end{bmatrix}}_{\text{Reference frame}} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix},$$

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} x_{i,1} & x_{i,2} \\ y_{i,1} & y_{i,2} \end{bmatrix}^{-1} \begin{bmatrix} x_{i,j} - \frac{x_{i,1}+x_{i,2}}{2} \\ y_{i,j} - \frac{y_{i,1}+y_{i,2}}{2} \end{bmatrix}.$$

Geometric hashing



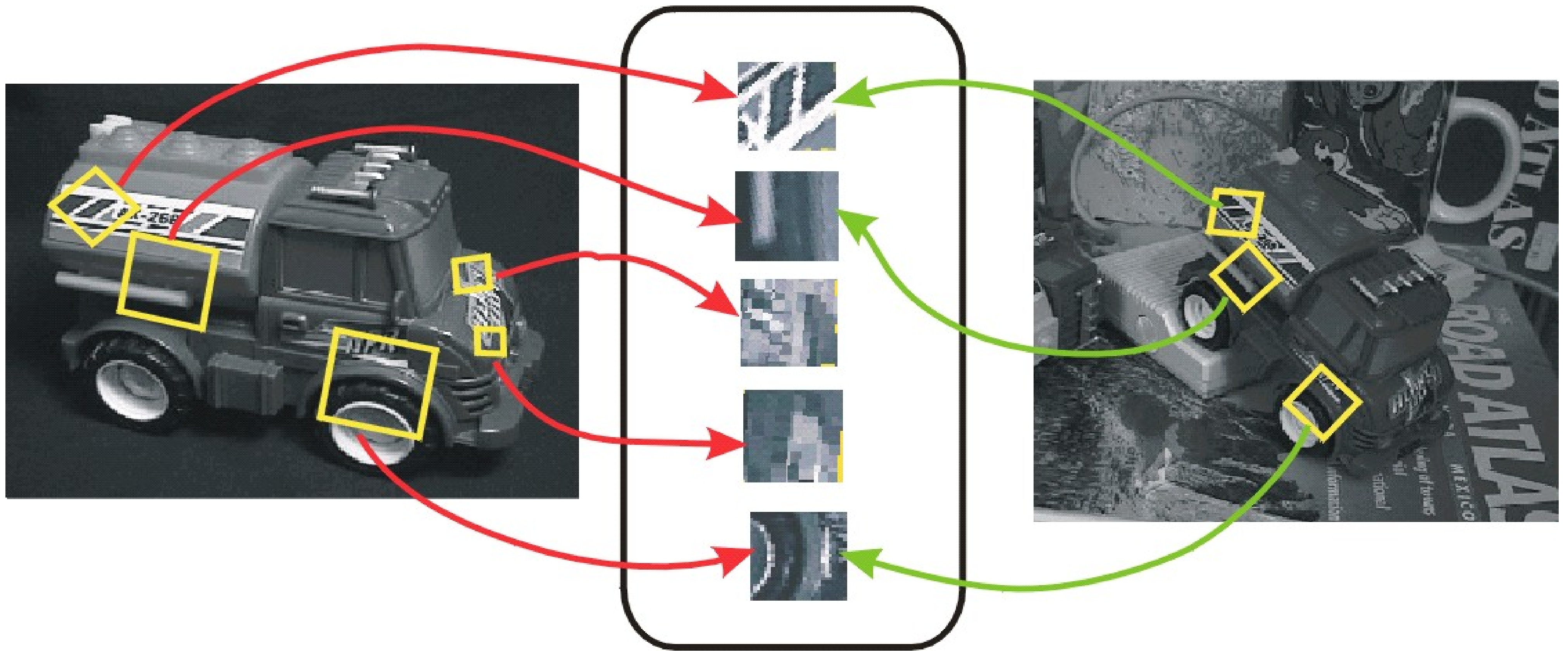
(a)



(b)

(c)

Geometric hashing: Recognition



Object Recognition from Local Scale-Invariant Features (SIFT)

Scalable recognition

Further readings

http://www.lix.polytechnique.fr/~nielsen/INF555/index.html

- Wolfson, H.J. & Rigoutsos, I (1997).
Geometric Hashing: An Overview.
IEEE Computational Science and Engineering, 4(4), 10-21.

- Andrew S. Glassner: Fill 'Er Up!
IEEE Computer Graphics and Applications 21(1): 78-85 (2001)