Summer Internship 2016

Mentored by – Dr. Amit Chattopadhyay

Contour Tree

Presented by :-
Samarth Mishra

x---Date---x

Introduction

- Many imaging technologies and scientic simulations produce data in the form of sample points with intensity Values.
- Contour trees were proposed by van Kreveld et al for computing isolines on terrain maps in geographic information systems.
- With terrain maps, a surface model is computed from elevation values at sample points in the plane.
- Contours can be traced from a surface model relatively easily, given a starting point

Definition

The contour tree for a Morse function is defined as a graph in which:

- each leaf vertex represents the creation or deletion of a component at a local extremum of the parameter
- each interior vertex represents the joining and/or splitting of two or more components at a critical point.
- each edge represents a component in the level sets for all values of the parameter between the values of the data points at each end of the edge.

Morse Theory

-> Morse theory studies the changes in topology of level sets of f as the parameter x changes. Points at which the topology of the level sets change are called critical points.

-> Morse theory requires that the critical points be isolated – i.e. that they occur at distinct points and values.

-> A function that satisfies this condition is called a Morse function. All points other than critical points are called regular points

Abstract

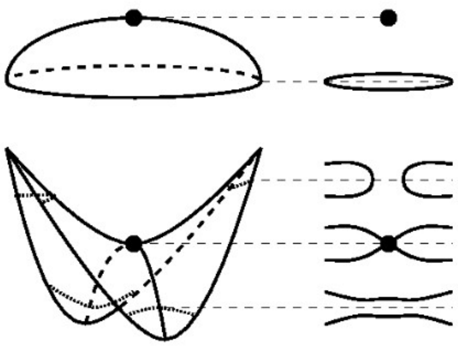
- The Contour Tree is a data structure that represents the relations between the connected components of the level sets in a scalar field.
- Two connected components that merge together (as one continuously changes the isovalue) are represented as two arcs that join at a node of the tree.
- The first efficient technique for Contour Tree computation in 2D was introduced by de Berg and van Kreveld. **O(n log n)**
- Recently Carr,Snoeyink and Axen presented an elegant extension to any di-mension based on a two-pass scheme that builds a Join Tree and a Split Tree that are merged into a unique Contour Tree. **O(m + n log n)**

Critical Points and Topology

- The points where the topology of the contour changes are called critical points.

The level set topology changes only at critical points

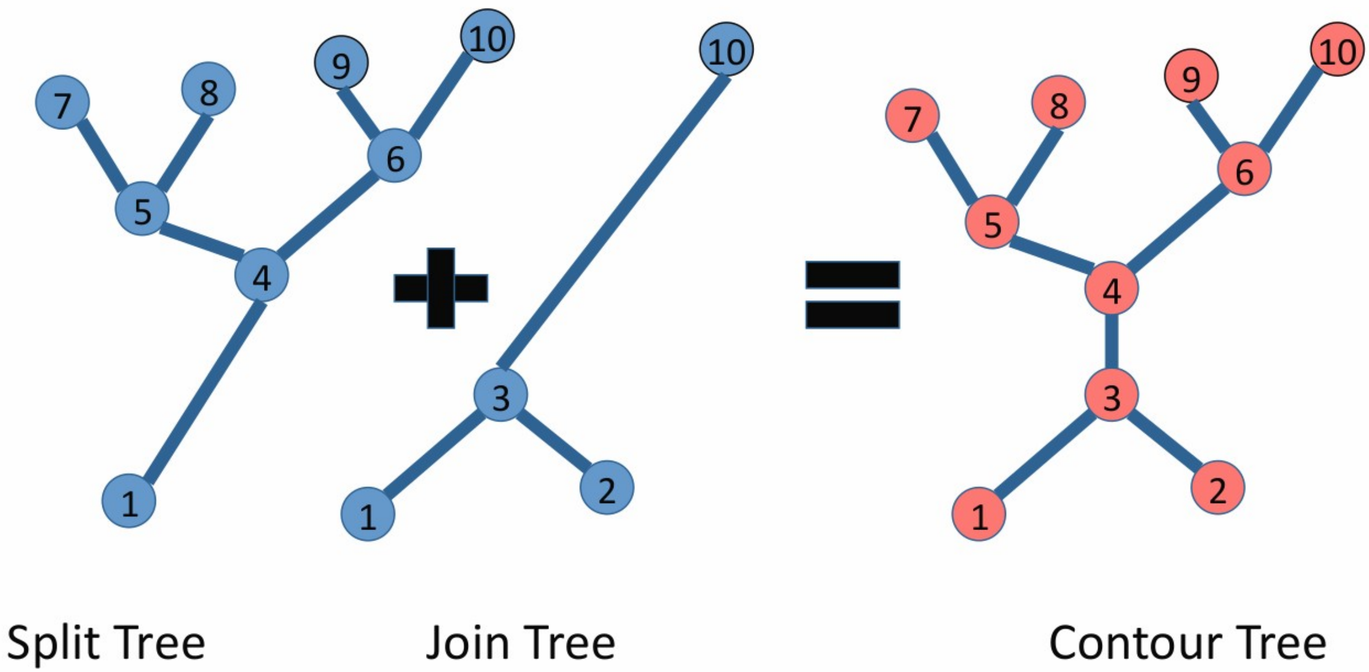
$$\frac{\partial f}{\partial x}(p) = \frac{\partial f}{\partial y}(p) = 0$$



Examples of 2D critical points

Contour Tree

Join + Split Trees = Contour Tree



Stages of the Algorithm

- Sorting of the vertices in the field
- Computing the Join Tree
- Computing the Split Tree
- Merging the Join Tree and Split Tree to build the Contour Tree

Join Tree

JoinTree$(vertices, edges)$

$JT \leftarrow$ NewTree()

$UF \leftarrow$ NewUF()

for $i = 0$ **to** $n - 1$ **do:**

  AddNode(JT, i)

  if IsMin$(\mathcal{F}, v_i)$ **then** NewSet(UF, i)

  for each edge $v_i v_j$ **with** $j < i$ **do:**

    $i' \leftarrow$ Find(UF, i)

    $j' \leftarrow$ Find(UF, j)

    if $j' \neq i'$ **then** AddArc(JT, i', j')

    Union(UF, i', j')

return JT

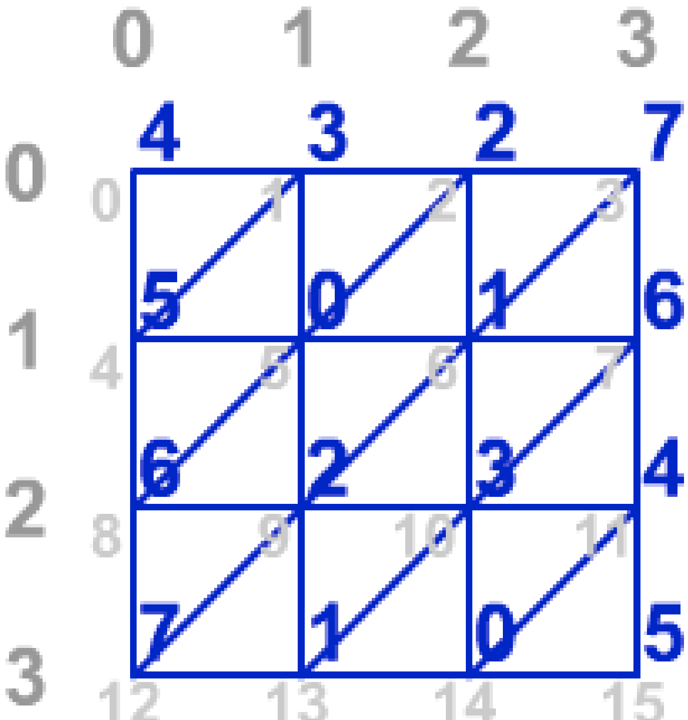
- **JT** is the Join Tree with `NewTree()` function initializing it.
- **UF** is the union-find Data Structure with `NewUF()` initializing it.
- `AddNode(JT,i)` adds a new node i to **JT**.
- `IsMin(F,v_i)` checks whether the vertex is a minimum. If true then it is added to a new set in **UF** by `NewSet(UF,i)` function.
- `Find(UF,i)` checks if the element is present in any of the set in **UF**. If true then it returns the representative element of the set.
- `AddArc(JT,i',j')` adds an edge between the element i' and j'.
- `Union(UF,i',j')` merges the set containing i' and j' and makes a new representative element.

Split Tree

- the main loop (line 3) would scan the vertices in reverse order
- the if statement in line 5 would test IsMax instead of IsMin
- the inner loop (line 6) would consider the edges (v_i, v_j) with $j > i$.

Example

Let us consider a scalar field :



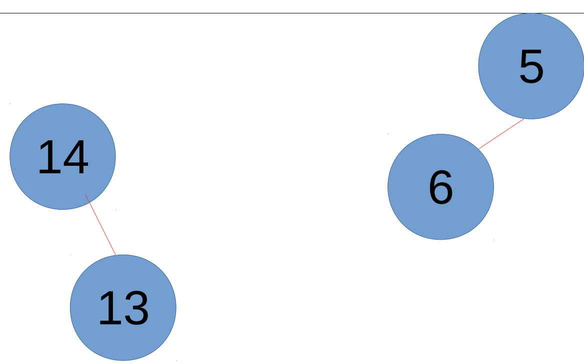
Scalar Field

Sorted Values

Vertex	5	14	6	13	2	9	1	10	0	11	4	15	7	8	3	12
Values	0	0	1	1	1	2	3	3	4	4	5	5	6	6	7	7

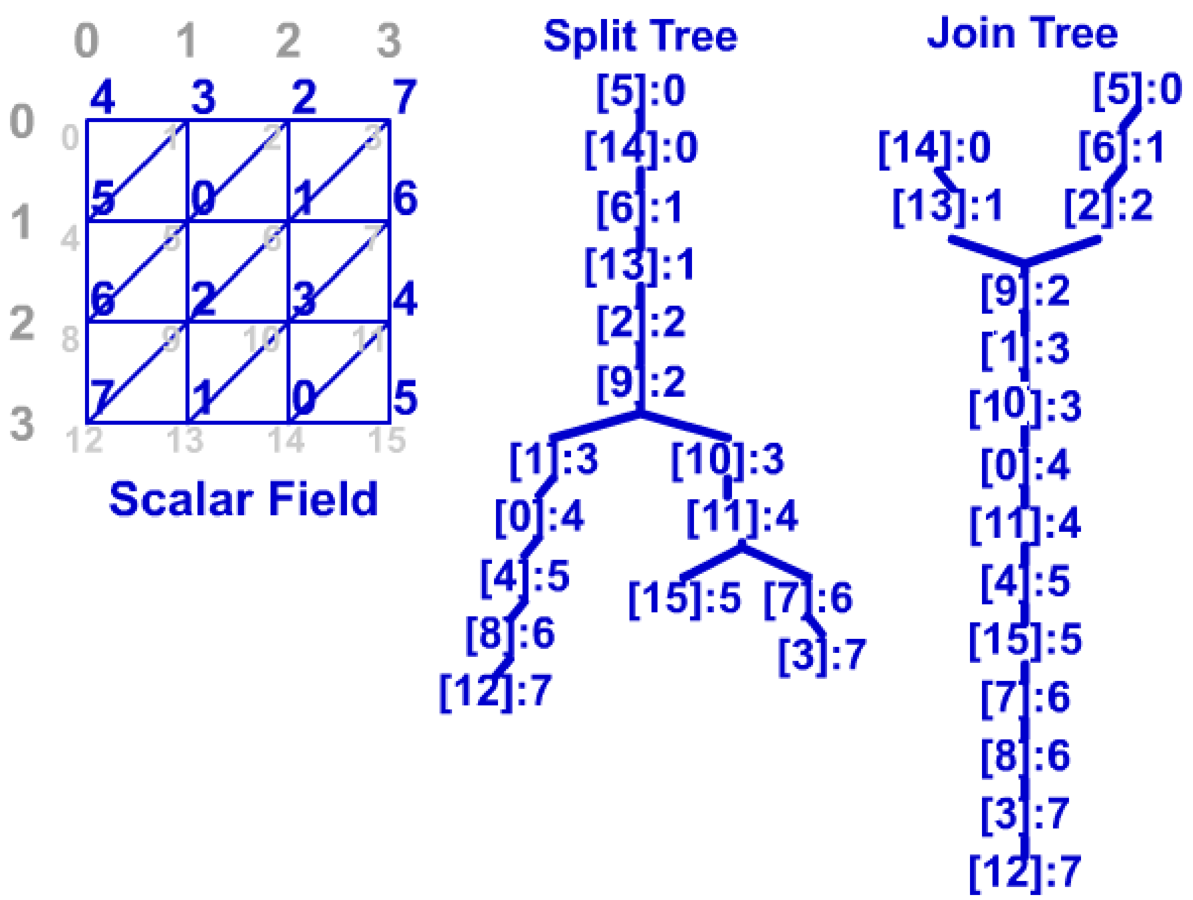
Note : If $f_i = f_j$ then in the case of ascending sorting i will lie before j if $i<j$

Join Tree

JT	UF
	[{5*}]
	[{5*} {14*}]
	[{5,6*} {14*}]
	[{5,6*} {14,13*}]

And So on ....for Join tree and the Split tree is created with the modifications as mentioned before.

Formation of Split Tree and Join Tree



Contour Tree

```
ContourTree(JT, ST)
Q ← NewQ()
CT ← NewTree()
for i = 0 to n − 1 do:
    AddNode(CT, i)
    if Leaf(JT, i) then Put(Q, [JT, i])
    if Leaf(ST, i) then Put(Q, [ST, i])
while [XT, i] ← Get(Q) do:
    j ← GetAdj(XT, i)
    DelNode(JT, i)
    DelNode(ST, i)
    AddArc(CT, ij)
    if Leaf(XT, j) then Put(Q, [XT, j])
return CT
```

- **JT** is the Join Tree **ST** is Split Tree.
- **Q** is Queue Data Structure with `NewQ()` inititalizing it.
- **CT** is Contour Tree with `NewTree()` inititalizing it.
- `AddNode(CT,i)` adds the node i to **CT**.
- `Leaf(XT,i)` checks if i is a node in **XT** {XT : JT/ST}.
- `Put(Q,[XT,i])` adds the node i to **Q** also adding the information about which tree **XT** {JT/ST}.
- `Get(Q)` pushes the elements out of **Q** getting node i and tree **XT**.
- `GetAdj(XT,i)` will give the information about the adjacent node to i in **XT** and store it in j.
- `DelNode(JT,i)` and `DelNode(ST,i)` deletes the node i in **JT** and **ST** respectively.
- `AddArc(CT,ij)` adds an edge between i and j in **CT**.