I/O-Efficient Algorithms for Computing Contours on a Terrain

Bardia Sadri
Duke University

joint work with:

Pankaj K. Agarwal
Duke University

Lars Arge
MADALGO

Thomas Mølhave
MADALGO
A **terrain** is the graph of a **continuous bivariate function**.
A terrain is the graph of a **continuous bivariate function**.

In GIS the surface of earth is often represented as a terrain that interpolates collected data.
A terrain is the graph of a continuous bivariate function.

In GIS the surface of earth is often represented as a terrain that interpolates collected data.

LIDAR (Light Detection and Ranging)
- Massive (irregular) point sets (1-10m resolution)
- Becoming relatively cheap and easy to collect
- Appalachian mountains between 50GB to 5TB
Given a plane triangulation $\mathcal{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\mathcal{M}$. 

Representation: Triangulated Irregular Network (TIN)
Given a plane triangulation $\mathcal{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\mathcal{M}$. 
Given a plane triangulation $\mathbb{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\mathbb{M}$. 

\[(v, h(v))\]
Given a plane triangulation $\mathbb{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\mathbb{M}$. 

Representation: Triangulated Irregular Network (TIN)
Given a plane triangulation $\mathcal{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\mathcal{M}$. 
Given a plane triangulation $\bar{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\bar{M}$. 
Given a plane triangulation $\mathbb{M}$ with a height $h(v)$ for each vertex $v$, one can linearly interpolate $h$ in the interior of every face of $\mathbb{M}$. 

$$\mathbb{M} = (\mathbb{M}, h)$$
The level-set $M_\ell$ at height $\ell$ is $h^{-1}(\ell)$. 
The level-set $\mathcal{M}_\ell$ at height $\ell$ is $h^{-1}(\ell)$. 
The level-set $M_\ell$ at height $\ell$ is $h^{-1}(\ell)$.
Each connected component of a level set is called a contour.
The level-set $M_\ell$ at height $\ell$ is $h^{-1}(\ell)$.
Each connected component of a level set is called a contour.
Given levels $L = \{\ell_1, \ldots, \ell_k\}$, the contour map is $h^{-1}(L)$.
The level-set $M_\ell$ at height $\ell$ is $h^{-1}(\ell)$.
Each connected component of a level set is called a contour.
Given levels $L = \{\ell_1, \ldots, \ell_k\}$, the contour map is $h^{-1}(L)$. 
The level-set $M_\ell$ at height $\ell$ is $h^{-1}(\ell)$. Each connected component of a level set is called a contour. Given levels $L = \{\ell_1, \ldots, \ell_k\}$, the contour map is $h^{-1}(L)$.
Applications at least as early as the 18th century.
Applications at least as early as the 18th century.

Pretty Old Stuff!

Philosophical Transactions of Royal Society of London, 1779

XXXIII. An Account of the Calculations made from the Survey and Measures taken at Schehallien, in order to ascertain the mean Density of the Earth. By Charles Hutton, Esq. F. R. S.

This circumstance at first gave me much trouble and dissatisfaction, till I fell upon the following method by which the defect was in a great measure supplied, and by which I was enabled to proceed in the estimation of the altitudes both with much expedition and a considerable degree of accuracy. This method was the connecting together by a faint line all the points which were of the same relative altitude: by so doing, I obtained a great number of irregular polygons lying within, and at some distance from, one another, and bearing a considerable degree of resemblance to each other: these polygons were the figures of so many level or horizontal sections of the hills, the relative altitudes of all the parts of them being known; and as every base or little space had
Applications at least as early as the 18th century.

XXXIII. An Account of the Calculations made from the Survey and Measures taken at Schehallien, in order to ascertain the mean Density of the Earth. By Charles Hutton, Esq. F. R. S.
Given a set of levels $L = \{\ell_1, \ldots, \ell_k\}$, compute the contour map $h^{-1}(L)$ such that each contour is reported separately and in sorted (circular) order.
Given a set of levels $L = \{\ell_1, \ldots, \ell_k\}$, compute the contour map $h^{-1}(L)$ such that each contour is reported separately and in sorted (circular) order.
Given a set of levels \( L = \{ \ell_1, \ldots, \ell_k \} \), compute the contour map \( h^{-1}(L) \) such that each contour is reported separately and in sorted (circular) order.
Given a set of levels $L = \{\ell_1, \ldots, \ell_k\}$, compute the contour map $h^{-1}(L)$ such that each contour is reported separately and in sorted (circular) order.
Preprocess the terrain to answer **contour queries** efficiently: 
*Given a level $\ell \in \mathbb{R}$, return the level set $h^{-1}(\ell)$ such that each contour is reported separately and in sorted (circular) order.*

**Output:** $a_1, a_2, \ldots, a_{13}, b_1, \ldots, b_{16}$
Classical Complexity: Number of basic operations as a function of $N$. 
Classical Complexity: Number of basic operations as a function of $N$.

Disk access is about $10^6$ times slower than main memory access.
To amortize access delay, disks transfer large contiguous blocks of data.

Classical Complexity: Number of basic operations as a function of $N$. Disk access is about $10^6$ times slower than main memory access. To amortize access delay, disks transfer large contiguous blocks of data.
The I/O Model

Classical Complexity: Number of basic operations as a function of $N$.

Disk access is about $10^6$ times slower than main memory access.
To amortize access delay, disks transfer large contiguous blocks of data.

$N = \text{number of items in the problem instance}$
$B = \text{number of items per disk block}$
$M = \text{number of items that fit main memory}$
$T = \text{number of items in output}$
To amortize access delay, disks transfer large contiguous blocks of data.

Disk access is about $10^6$ times slower than main memory access.

To amortize access delay, disks transfer large contiguous blocks of data.

**Classical Complexity:** Number of basic operations as a function of $N$.

$$N = \text{number of items in the problem instance}$$

$$B = \text{number of items per disk block}$$

$$M = \text{number of items that fit main memory}$$

$$T = \text{number of items in output}$$

**I/O-Complexity:** Number of I/Os as a function of $N$, $B$, $M$, and $T$. 

8
Classical Complexity: Number of basic operations as a function of $N$. Disk access is about $10^6$ times slower than main memory access. To amortize access delay, disks transfer large contiguous blocks of data.

$$
N = \text{number of items in the problem instance} \\
B = \text{number of items per disk block} \\
M = \text{number of items that fit main memory} \\
T = \text{number of items in output}
$$

I/O-Complexity: Number of I/Os as a function of $N$, $B$, $M$, and $T$. Important to store/access data to take advantage of locality.
To amortize access delay, disks transfer large contiguous blocks of data.

Disk access is about $10^6$ times slower than main memory access.

**Classical Complexity:** Number of basic operations as a function of $N$.

- $N$ = number of items in the problem instance
- $B$ = number of items per disk block
- $M$ = number of items that fit main memory
- $T$ = number of items in output

**I/O-Complexity:** Number of I/Os as a function of $N$, $B$, $M$, and $T$.

Important to store/access data to take advantage of locality.
To amortize access delay, disks transfer large contiguous blocks of data. Disk access is about $10^6$ times slower than main memory access.

### Classical Complexity

Number of basic operations as a function of $N$.

- $N = \text{number of items in the problem instance}$
- $B = \text{number of items per disk block}$
- $M = \text{number of items that fit main memory}$
- $T = \text{number of items in output}$

### I/O-Complexity

Number of I/Os as a function of $N$, $B$, $M$, and $T$.

Important to store/access data to take advantage of locality.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning</td>
<td>$N$</td>
<td>$N/B$</td>
</tr>
<tr>
<td>Sorting</td>
<td>$N \log B$</td>
<td>$\frac{N}{B} \log \frac{M}{B} \frac{N}{B}$</td>
</tr>
<tr>
<td>Permuting</td>
<td>$N$</td>
<td>$\min \left{ N, \frac{N}{B} \log \frac{M}{B} \frac{N}{B} \right}$</td>
</tr>
<tr>
<td>Searching</td>
<td>$\log_2 N$</td>
<td>$\log_B N$</td>
</tr>
</tbody>
</table>

### The I/O Model

<table>
<thead>
<tr>
<th>Operation</th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning</td>
<td>$N$</td>
<td>$N/B$</td>
</tr>
<tr>
<td>Sorting</td>
<td>$N \log B$</td>
<td>$\frac{N}{B} \log \frac{M}{B} \frac{N}{B}$</td>
</tr>
<tr>
<td>Permuting</td>
<td>$N$</td>
<td>$\min \left{ N, \frac{N}{B} \log \frac{M}{B} \frac{N}{B} \right}$</td>
</tr>
<tr>
<td>Searching</td>
<td>$\log_2 N$</td>
<td>$\log_B N$</td>
</tr>
</tbody>
</table>

**Sort**($N$)
To amortize access delay, disks transfer large contiguous blocks of data. Disk access is about $10^6$ times slower than main memory access.

Classical Complexity: Number of basic operations as a function of $N$.

- $N$ = number of items in the problem instance
- $B$ = number of items per disk block
- $M$ = number of items that fit main memory
- $T$ = number of items in output

I/O-Complexity: Number of I/Os as a function of $N$, $B$, $M$, and $T$. Important to store/access data to take advantage of locality.
## Previous Work

Answering contour queries I/O-efficiently:

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing I/Os</th>
<th>Structure Size</th>
<th>Query I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chiang, Silva’97</td>
<td>$O(\text{Sort}(N))$</td>
<td>$O(N)$</td>
<td>$O(\log_B N + T/B)$</td>
</tr>
<tr>
<td>Agarwal, Arge, Murali, Varadarajan, Vitter’98</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(\log_B N + T/B)$</td>
</tr>
</tbody>
</table>
## Previous Work

Answering contour queries I/O-efficiently:

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing I/Os</th>
<th>Structure Size</th>
<th>Query I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chiang, Silva’97</strong></td>
<td>$O(\text{Sort}(N))$</td>
<td>$O(N)$</td>
<td>$O(\log_B N + T/B)$</td>
</tr>
<tr>
<td><strong>Agarwal, Arge, Murali, Varadarajan, Vitter’98</strong></td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(\log_B N + T/B)$</td>
</tr>
</tbody>
</table>
Answering contour queries I/O-efficiently:

<table>
<thead>
<tr>
<th></th>
<th>Preprocessing I/Os</th>
<th>Structure Size</th>
<th>Query I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chiang, Silva’97</td>
<td>$O(\text{Sort}(N))$</td>
<td>$O(N)$</td>
<td>$O(\log_B N + T/B)$</td>
</tr>
<tr>
<td>Agarwal, Arge, Murali, Varadharajan, Vitter’98</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(\log_B N + T/B)$</td>
</tr>
</tbody>
</table>

Unsorted Contours!
“Very naïve” Algorithm: Generate one contour at a time

Find one “seed” triangle intersecting each contour and trace out the contour sequentially.
“Very naïve” Algorithm: Generate one contour at a time

Find one “seed” triangle intersecting each contour and trace out the contour sequentially.
Find one “seed” triangle intersecting each contour and trace out the contour sequentially.
“Very naïve” Algorithm: Generate one contour at a time

Find one “seed” triangle intersecting each contour and trace out the contour sequentially.
“Very naïve” Algorithm: Generate one contour at a time

Find one “seed” triangle intersecting each contour and trace out the contour sequentially.

This is essentially the optimal algorithm for the RAM model.
“Very naïve” Algorithm: Generate one contour at a time

Find one “seed” triangle intersecting each contour and trace out the contour sequentially.

This is essentially the optimal algorithm for the RAM model.

I/O Complexity:

\[ O(N/B + T). \]

\# segments in the output
Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.
Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.
Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.
"Less naïve" Algorithm: Generate pieces, sort later

Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.

For segment $s$ at level $\ell_i$ store pair $(\ell_i, s)$ plus the segments before and after $s$ on contour containing $s$. 
"Less naïve" Algorithm: Generate pieces, sort later

**Scan** the triangles (in the order laid out on the disk) and generate all segments. Then **sort** the output.

For segment \( s \) at level \( \ell_i \) store pair \((\ell_i, s)\) plus the segments before and after \( s \) on contour containing \( s \).

Sort pairs on first component to separates level sets. Then use **successor/predecessor-sorting** to put contours in order.
“Less naïve” Algorithm: Generate pieces, sort later

Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.

For segment $s$ at level $l_i$ store pair $(l_i, s)$ plus the segments before and after $s$ on contour containing $s$.

Sort pairs on first component to separates level sets. Then use successor/predecessor-sorting to put contours in order.

I/O Complexity:

$$O(N/B + \text{Sort}(T)).$$
"Less naïve" Algorithm: Generate pieces, sort later

Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.

For segment $s$ at level $\ell_i$ store pair $(\ell_i, s)$ plus the segments before and after $s$ on contour containing $s$.

Sort pairs on first component to separates level sets. Then use successor/predecessor-sorting to put contours in order.

I/O Complexity:

$$O\left(\frac{N}{B} + \text{Sort}(T)\right).$$

This talk: $O(\text{Sort}(N) + \frac{T}{B})$. 
Idea: Grow Contours Contiguously

If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.
If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.
Idea: Grow Contours Contiguously

If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.
If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.
If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.

≺: such an ordering
Idea: Grow Contours Contiguously

If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.

≺: such an ordering
\( \Delta_\ell \): triangles that intersect level \( \ell \)
Idea: Grow Contours Contiguously

If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.

≺: such an ordering

$\Delta_\ell$: triangles that intersect level $\ell$
If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.

≺: such an ordering
\(\Delta_\ell\): triangles that intersect level \(\ell\)

The restriction of \(\prec\) to \(\Delta_\ell\) traverses each contour of \(M\) in circular order.
1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
The Algorithm

1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$. 

**The Algorithm**

![Diagram of a terrain with a sweep plane]
1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\preceq$.
3. When passing target level $\ell_i \in L$, **dump** contents of tree to disk.
1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
2. Keep triangles that intersect the sweep plane in a **search tree** ordered by $\prec$.
3. When passing target level $\ell_i \in L$, **dump** contents of tree to disk.
1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
2. Keep triangles that intersect the sweep plane in a **search tree** ordered by $\prec$.
3. When passing target level $\ell_i \in L$, **dump** contents of tree to disk.
1. **Sweep** the terrain by a horizontal plane in the \( \pm z \) direction.  

2. Keep triangles that intersect the sweep plane in a search tree ordered by \( \prec \).  

   - **Amortized Sort** \( (N) \)

3. When passing target level \( \ell_i \in L \), **dump** contents of tree to disk.
The Algorithm

1. **Sweep** the terrain by a horizontal plane in the $+z$ direction.
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$.
3. When passing target level $\ell_i \in L$, **dump** contents of tree to disk.
The Algorithm

1. **Sweep** the terrain by a horizontal plane in the $\pm z$ direction.  
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$.  
3. When passing target level $\ell_i \in L$, dump contents of tree to disk.

Using a persistent search tree, we can answer contour queries in $O(\log_B N + T/B)$ I/Os.
1. **Sweep** the terrain by a horizontal plane in the \( +z \) direction.

2. Keep triangles that intersect the sweep plane in a **search tree** ordered by \( \prec \).  

3. When passing target level \( \ell_i \in L \), **dump** contents of tree to disk.

Using a **persistent** search tree, we can answer contour queries in \( O(\log_B N + T/B) \) I/Os.

Preprocessing needs \( O(\text{Sort}(N)) \) I/Os and \( O(N) \) space.
**Theorem.** One can find in $O(\text{Sort}(N))$ I/Os a total ordering “≺” of triangles of $M$, s.t. for any $\ell$:

1. Triangles in $\Delta_\ell$ are ≺-sorted in cw or ccw order.
Theorem. One can find in $O(\text{Sort}(N))$ I/Os a total ordering “$\prec$” of triangles of $\mathbb{M}$, s.t. for any $\ell$:

1. Triangles in $\Delta_\ell$ are $\prec$-sorted in cw or ccw order.
2. Subsets of $\Delta_\ell$ intersecting distinct contours $C_1$ and $C_2$ don’t “interleave”:
   \[ t_1 < t_3 < t_2 \quad \implies \quad t_1 < t_4 < t_2 \]
**Theorem.** One can find in $O(\text{Sort}(N))$ I/Os a total ordering “≺” of triangles of $\mathbb{M}$, s.t. for any $\ell$:

1. Triangles in $\Delta_\ell$ are ≺-sorted in cw or ccw order.
2. Subsets of $\Delta_\ell$ intersecting distinct contours $C_1$ and $C_2$ don’t “interleave”:

   $t_1 ≺ t_3 ≺ t_2 \implies t_1 ≺ t_4 ≺ t_2$

We call ≺ a “level-ordering” of triangles in $\mathbb{M}$.
Theorem. One can find in $O(\text{Sort}(N))$ I/Os a total ordering $\prec$ of triangles of $\mathbb{M}$, s.t. for any $\ell$:

1. Triangles in $\Delta_\ell$ are $\prec$-sorted in cw or ccw order.
2. Subsets of $\Delta_\ell$ intersecting distinct contours $C_1$ and $C_2$ don’t “interleave”:
   
   $t_1 \prec t_3 \prec t_2 \implies t_1 \prec t_4 \prec t_2$

We call $\prec$ a “level-ordering” of triangles in $\mathbb{M}$. 

---

$\begin{align*}
\text{Diagram:}\quad a_1 a_2 b_1 c_1 c_2 c_3 b_2 b_3 d_1 d_2 b_4 a_3 a_4 a_5
\end{align*}$
**Theorem.** One can find in $O(Sort(N))$ I/Os a total ordering “≺” of triangles of $\mathbb{M}$, s.t. for any $\ell$:

1. Triangles in $\Delta_\ell$ are $\prec$-sorted in cw or ccw order.
2. Subsets of $\Delta_\ell$ intersecting distinct contours $C_1$ and $C_2$ don’t “interleave”:
   \[ t_1 \prec t_3 \prec t_2 \implies t_1 \prec t_4 \prec t_2 \]

We call $\prec$ a “level-ordering” of triangles in $\mathbb{M}$.

Can separate contours using a stack in $O(T/B)$ I/Os.
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

maxima
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An *elementary* terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).
An elementary terrain has no saddles; thus 1 max and 1 min (boundary). Take a monotone min (bd) to max path $P$ and delete its dual from $\mathbb{M}^*$. 

Level Ordering of Elementary Terrains
Level Ordering of Elementary Terrains

An elementary terrain has no saddles; thus 1 max and 1 min (boundary). Take a monotone min (bd) to max path $P$ and delete its dual from $\mathbb{M}^*$. 

[Diagram showing a 3D terrain with a red path and a 2D network.]
An elementary terrain has no saddles; thus 1 max and 1 min (boundary). Take a monotone min (bd) to max path $P$ and delete its dual from $\mathbb{M}^*$. 

"Level Ordering of Elementary Terrains"
Lemma. Every cycle of $\mathbb{M}^*$ loses precisely one edge.
An elementary terrain has no saddles; thus 1 max and 1 min (boundary). Take a monotone min (bd) to max path $P$ and delete its dual from $\mathcal{M}^*$. 

**Lemma.** Every cycle of $\mathcal{M}^*$ loses precisely one edge.

**Lemma.** The resulting dual graph is acyclic and the induced relation “≺” a partial order. Thus we can topologically sort it into a total order.
An elementary terrain has no saddles; thus 1 max and 1 min (boundary). Take a monotone min (bd) to max path $P$ and delete its dual from $\mathbb{M}^*$.

**Lemma.** Every cycle of $\mathbb{M}^*$ loses precisely one edge.

**Lemma.** The resulting dual graph is acyclic and the induced relation “≺” a partial order. Thus we can topologically sort it into a total order.

[Arge, Toma, Zeh’03]
What about non-elementary terrains?

A saddle is **negative** if it joins two disjoint connected components of its sublevel-set and **positive** otherwise.
A saddle is negative if it joins two disjoint connected components of its sublevel-set and positive otherwise.

What about non-elementary terrains?
What about non-elementary terrains?

A saddle is **negative** if it joins two disjoint connected components of its sublevel-set and **positive** otherwise.

If we replace \( h \) with \(-h\), the two types switch roles.
What about non-elementary terrains?

A saddle is **negative** if it joins two disjoint connected components of its sublevel-set and **positive** otherwise.

If we replace $h$ with $-h$, the two types switch roles.

[Agarwal, Arge, Yi '06] Positive and negative saddle points can be found in $O(\text{Sort}(N))$ I/Os.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.
Positive and Negative Cut-Trees

Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.

**Lemma.** The result is a tree (not just forest) that reaches every maximum.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.

Lemma. The result is a tree (not just forest) that reaches every maximum.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.

Lemma. The result is a tree (not just forest) that reaches every maximum.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.

Lemma. The result is a tree (not just forest) that reaches every maximum.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.

Lemma. The result is a tree (not just forest) that reaches every maximum.
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.

Lemma. The result is a tree (not just forest) that reaches every maximum.
Turning a terrain into an elementary one by surgery
Turning a terrain into an elementary one by surgery
Turning a terrain into an elementary one by surgery
Turning a terrain into an elementary one by surgery
Turning a terrain into an elementary one by surgery
Turning a terrain into an elementary one by surgery

new max
Turning a terrain into an elementary one by surgery

new max
Turning a terrain into an elementary one by surgery

new max
Turning a terrain into an elementary one by surgery

new max
Lemma. Doing this removes all positive saddles and maxima and adds a new maximum.
What surgery does to contours?

The elementary terrain $M'$ has all the triangles of $M$ plus some of “new” triangles.
What surgery does to contours?

The elementary terrain $\mathbf{M}'$ has all the triangles of $\mathbf{M}$ plus some of “new” triangles.

All contours of a level set of $\mathbf{M}$ are combined in a single contour in $\mathbf{M}'$. 
What surgery does to contours?

The elementary terrain $M'$ has all the triangles of $M$ plus some of “new” triangles.

All contours of a level set of $M$ are combined in a single contour in $M'$

**Theorem.** In a contour of $M'$, corresponding contours of $M$ are broken (by segments from new triangles) in a nested (parenthesized) manner.

$$a_1 a_2 \ast \ast b_1 \ast \ast \ast c_1 c_2 c_3 \ast \ast \ast \ast b_2 b_3 \ast d_1 d_2 \ast \ast b_4 \ast \ast a_3 a_4 a_5$$
What surgery does to contours?

The elementary terrain $M'$ has all the triangles of $M$ plus some of “new” triangles.

All contours of a level set of $M$ are combined in a single contour in $M'$

**Theorem.** In a contour of $M'$, corresponding contours of $M$ are broken (by segments from new triangles) in a nested (parenthesized) manner.

$$[a_1 a_2 * *[b_1 * *[c_1 c_2 c_3] * * * b_2 b_3 *[d_1 d_2] * * b_4] * * a_3 a_4 a_5]$$
Where is nesting coming from?
Where is nesting coming from?
Where is nesting coming from?
Where is nesting coming from?
Where is nesting coming from?
Where is nesting coming from?
A contour is red (blue) if “locally” the sublevel set is in its outside (inside).
A contour is red (blue) if “locally” the sublevel set is in its outside (inside).
A contour is red (blue) if “locally” the sublevel set is in its outside (inside).
A contour is red (blue) if “locally” the sublevel set is in its outside (inside).

**Theorem.** Only the red tree connects a blue contour and its red children.
A contour is red (blue) if “locally” the sublevel set is in its outside (inside).

**Theorem.** Only the red tree connects a blue contour and its red children.

**Theorem.** If we contract each contour to a point, the result is a tree.
A contour is red (blue) if “locally” the sublevel set is in its outside (inside).

**Theorem.** Only the red tree connects a blue contour and its red children.

**Theorem.** If we contract each contour to a point, the result is a tree.
The embedding of the terrain is unimportant: the set of triangles that intersect a level set only depends on function value on vertices.
The embedding of the terrain is unimportant: the set of triangles that intersect a level set only depends on function value on vertices.

What about surfaces of genus $\geq 1$? (Orientable or not)
The embedding of the terrain is **unimportant**: the set of triangles that intersect a level set only depends on function value on vertices.

What about surfaces of genus $\geq 1$? (Orientable or not)

Can a topological sphere $M$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?
The embedding of the terrain is unimportant: the set of triangles that intersect a level set only depends on function value on vertices.

What about surfaces of genus $\geq 1$? (Orientable or not)

Can a topological sphere $M$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?
The embedding of the terrain is unimportant: the set of triangles that intersect a level set only depends on function value on vertices.

What about surfaces of genus $\geq 1$? (Orientable or not)

Can a topological sphere $M$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?
The embedding of the terrain is unimportant: the set of triangles that intersect a level set only depends on function value on vertices.

What about surfaces of genus $\geq 1$? (Orientable or not)

Can a topological sphere $M$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?

Thank You!