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

Bardia Sadri
University of Toronto

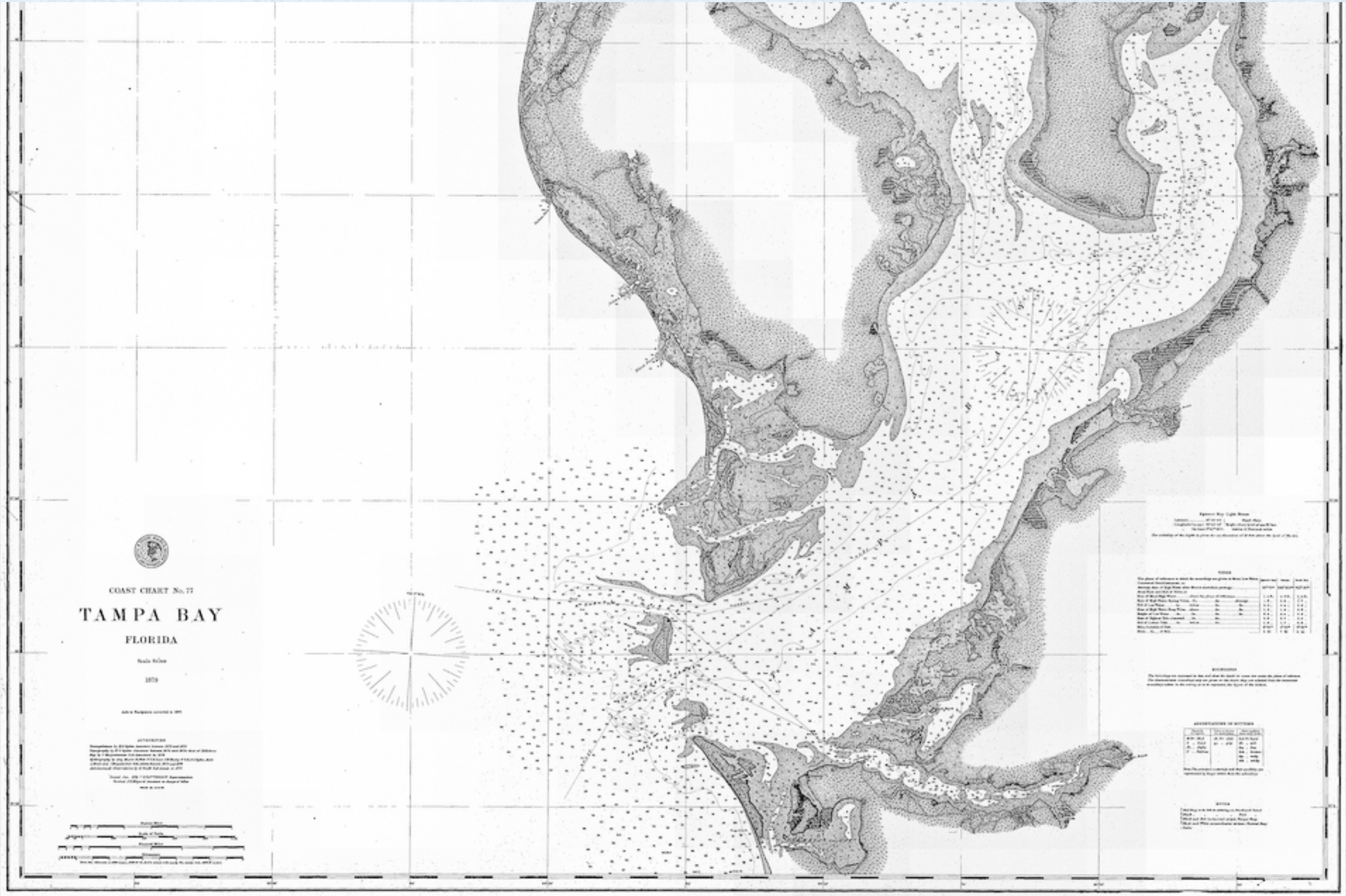
joint work with:

Pankaj K. Agarwal
Duke University

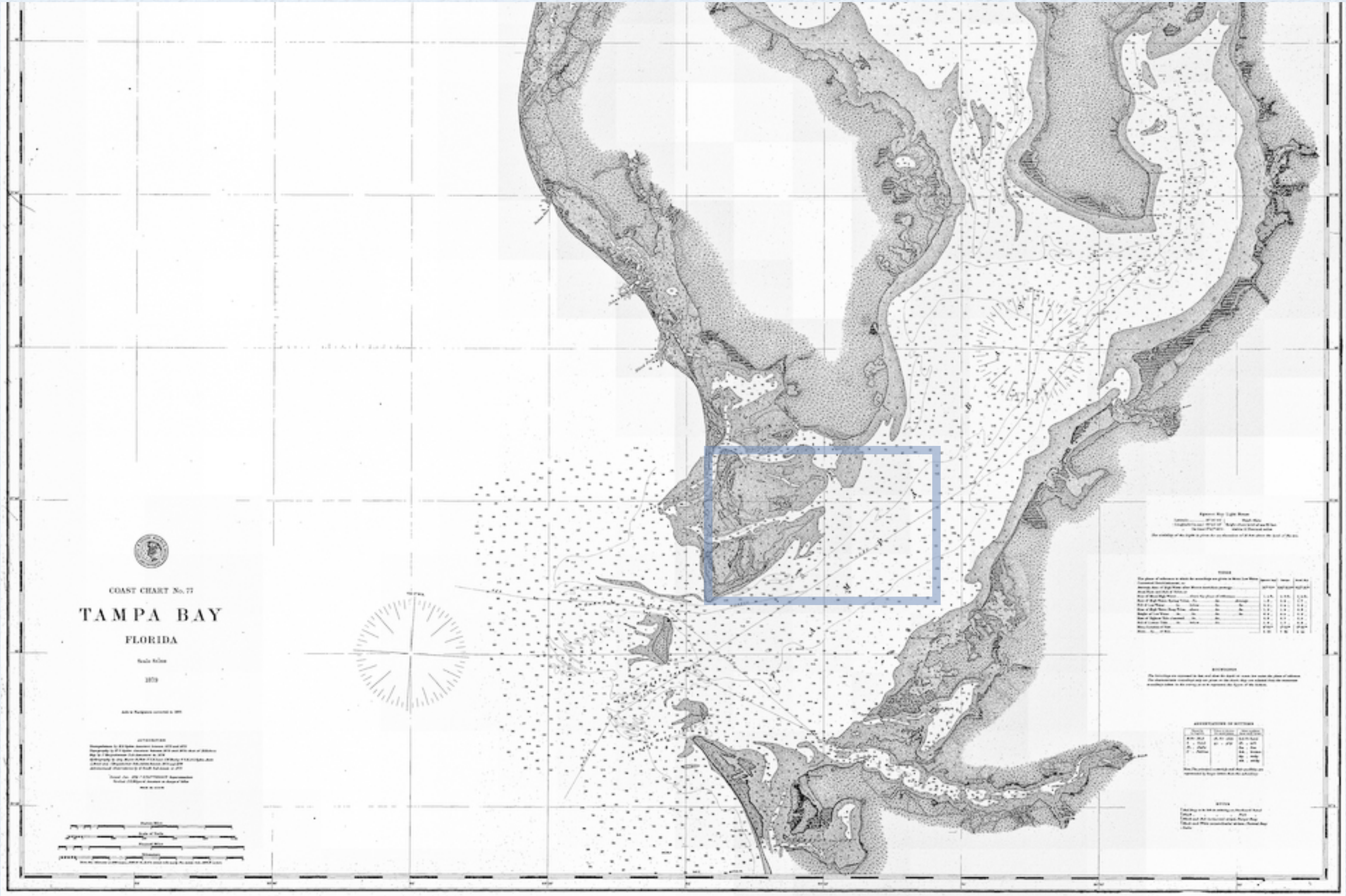
Lars Arge
MADALGO

Thomas Mølhave
MADALGO

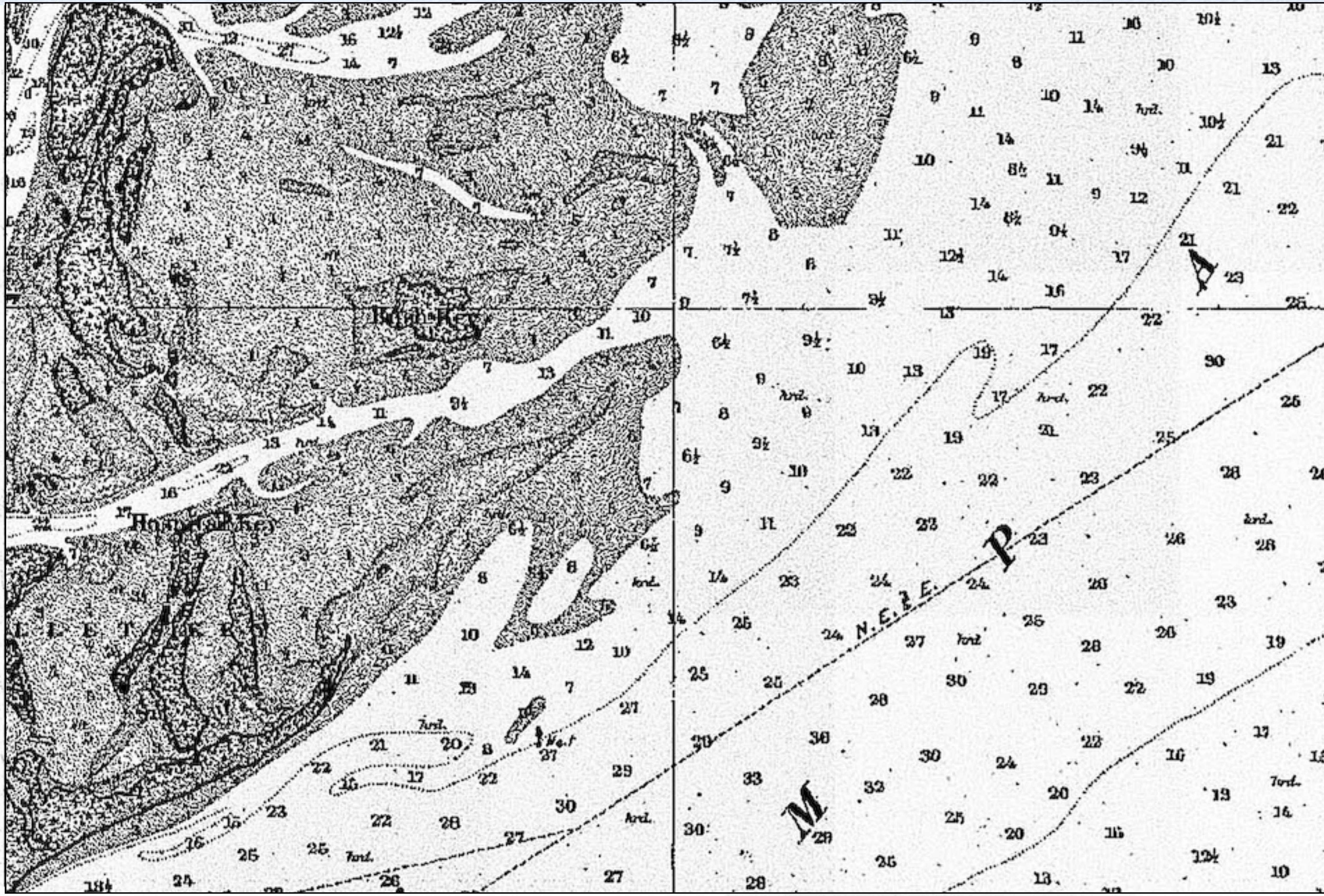
Pretty Old Stuff!

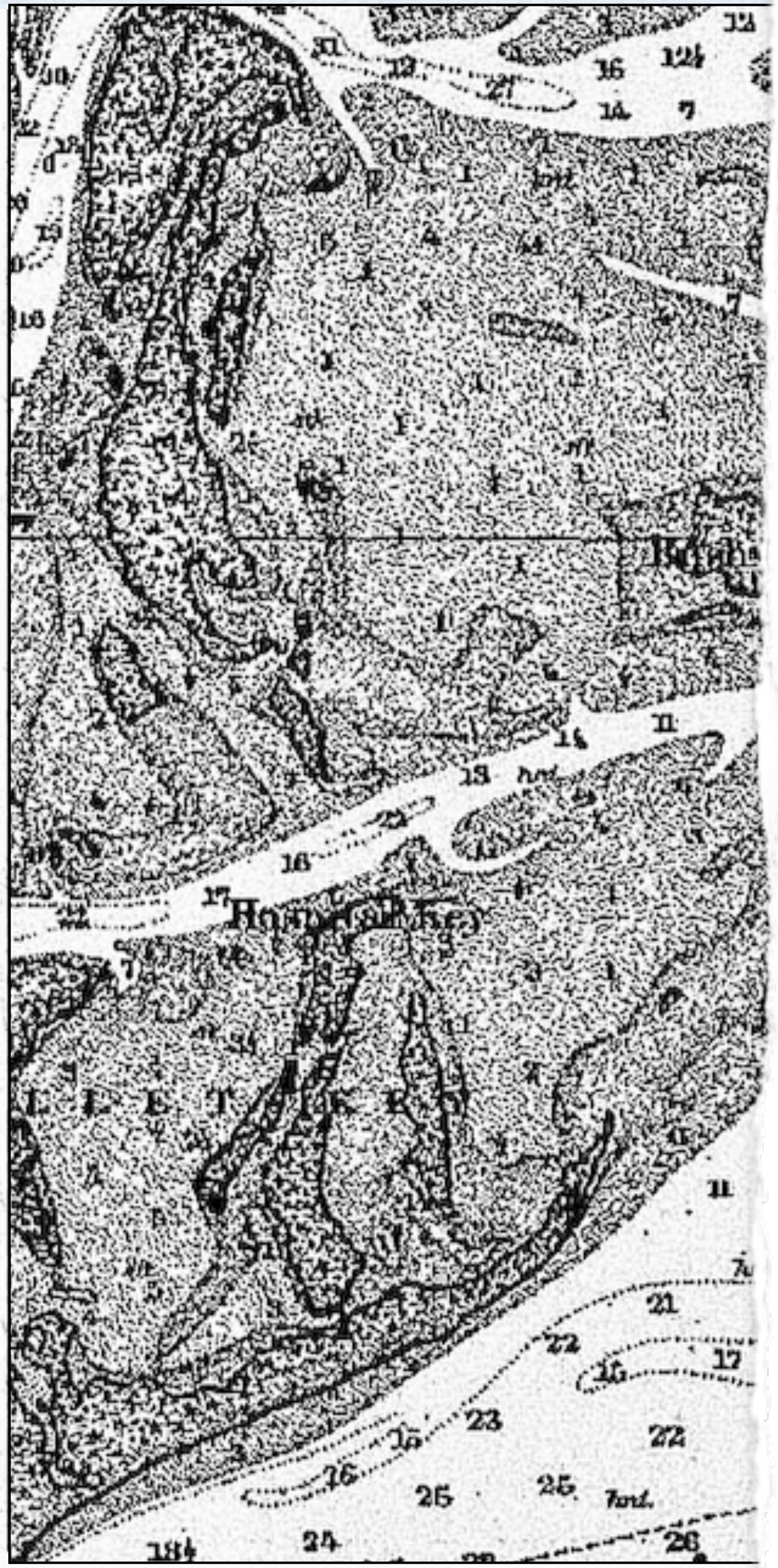


Pretty Old Stuff!



Pretty Old Stuff!



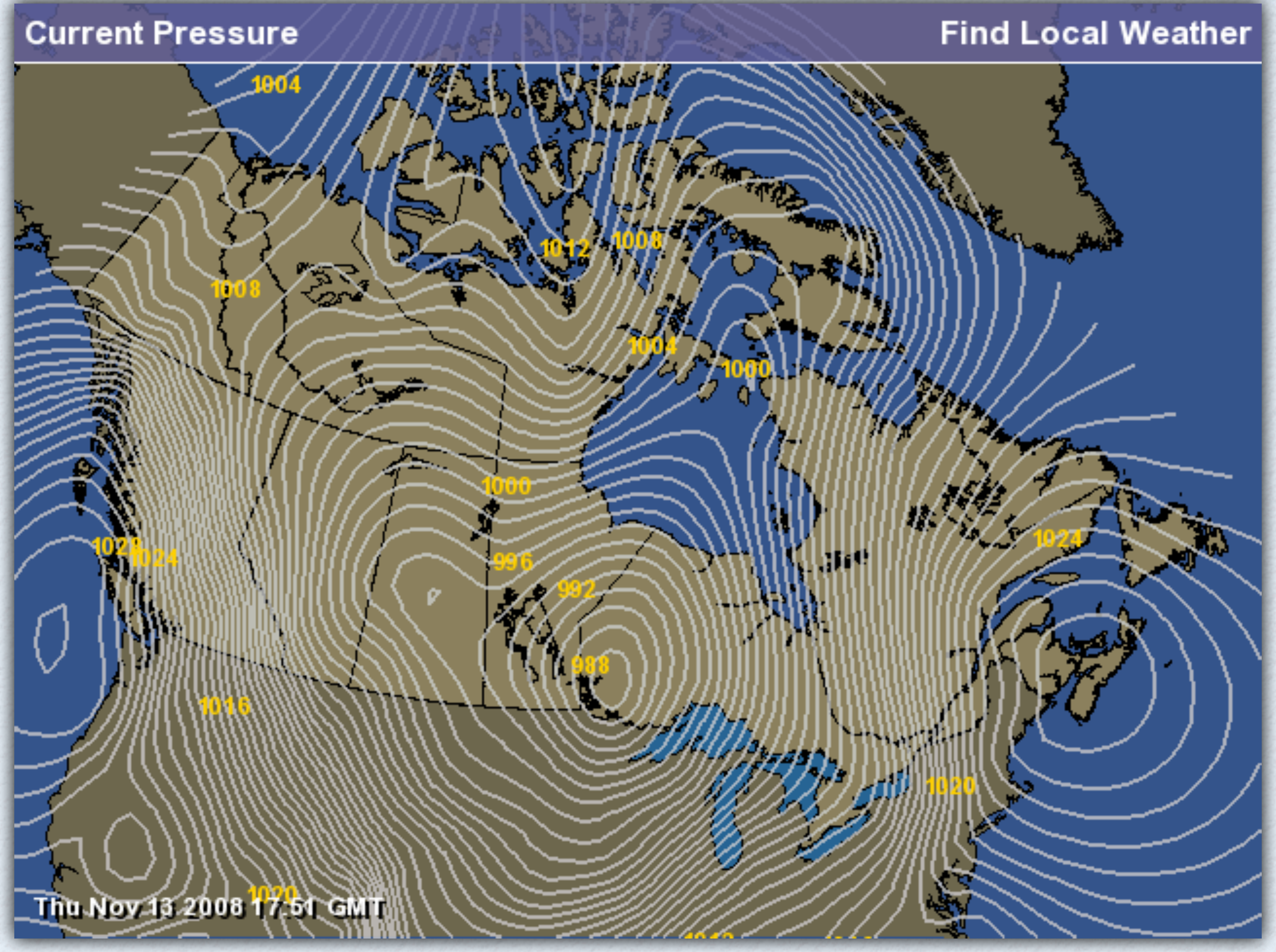


XXXIII. *An Account of the Calculations made from the Survey and Measures taken at* Schehallien, *in order to ascertain the mean Density of the Earth.* *By* Charles Hutton, *Esq. F. R. S.*

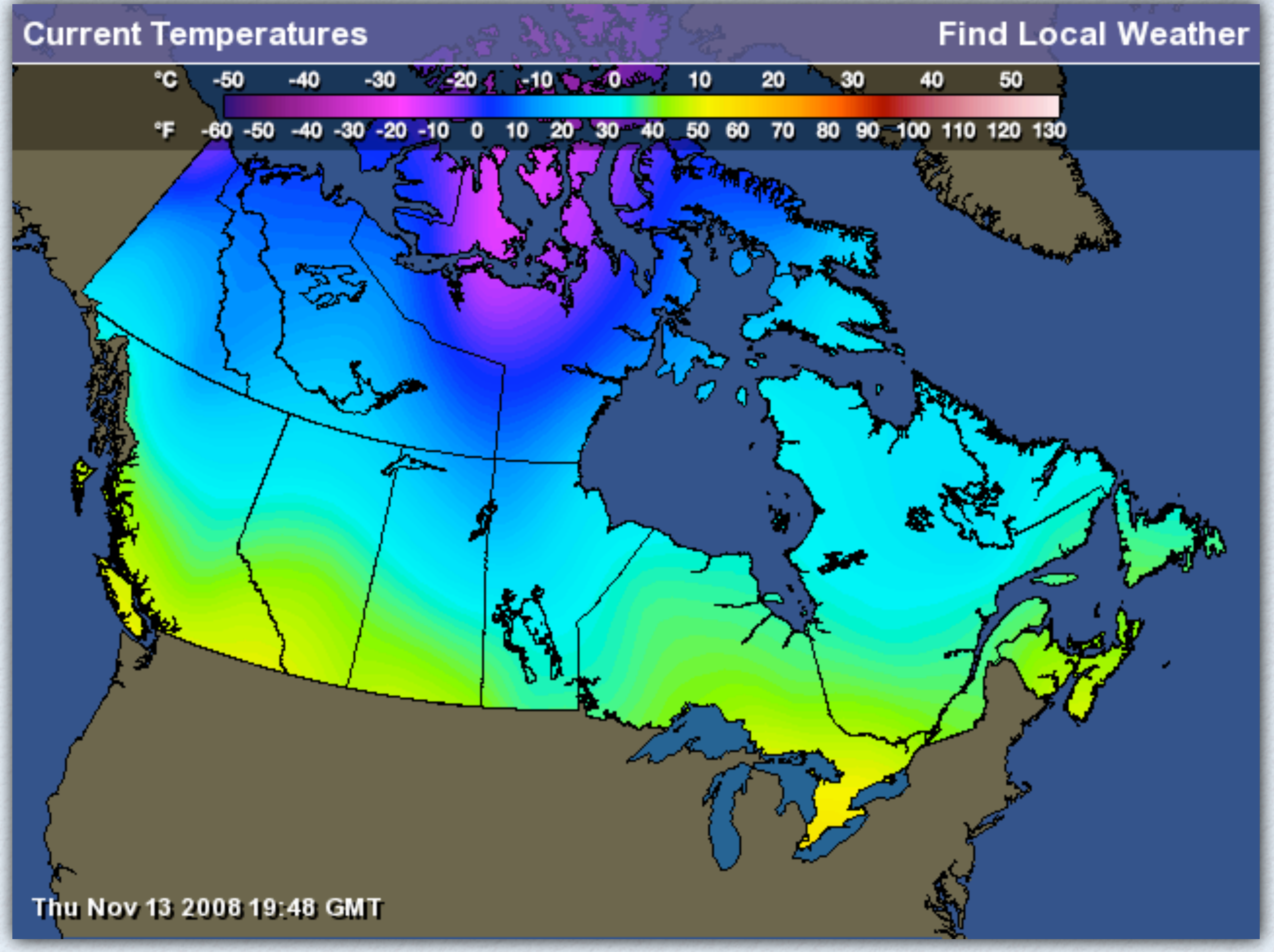
This circumstance at first gave me much trouble and dissatisfaction, till I fell upon the following method by which the defect was in a great measure supplied, and by which I was enabled to proceed in the estimation of the altitudes both with much expedition and a considerable degree of accuracy. This 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

Philosophical Transactions of Royal Society of London, 1779

Not Just for Altitude

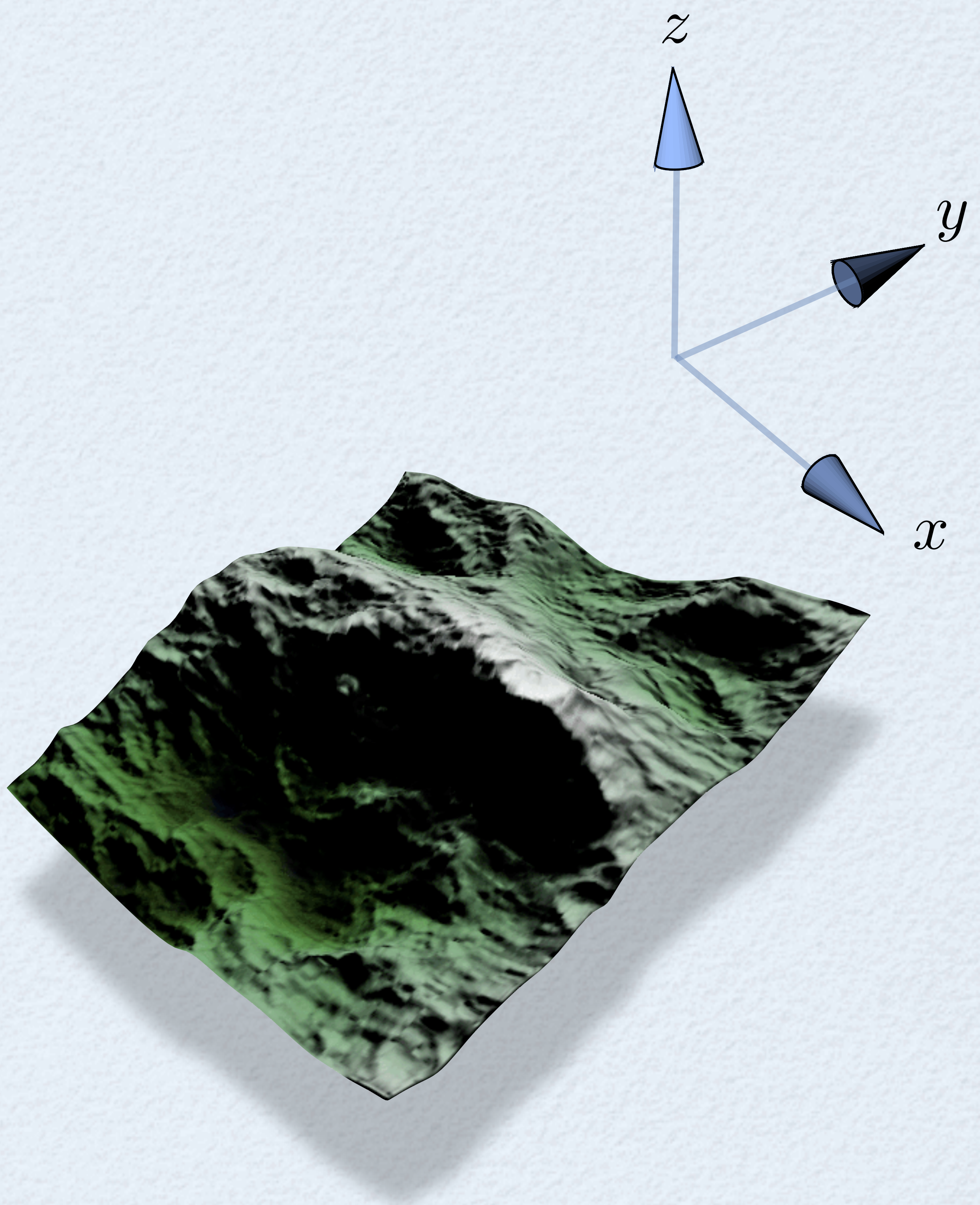


Not Just for Altitude



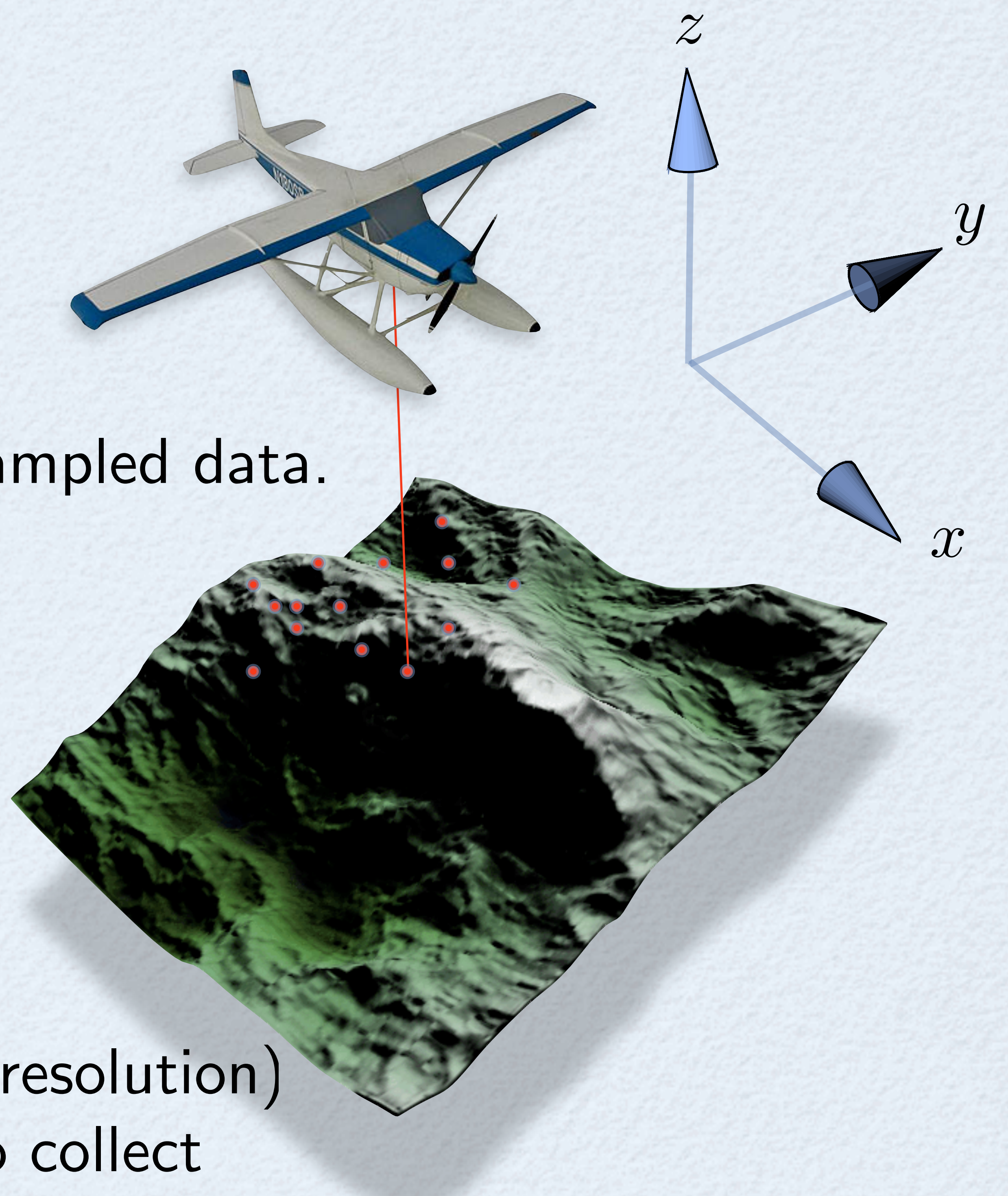
Terrains

A terrain is the graph of a continuous bivariate function.



Terrains

A terrain is the graph of a continuous bivariate function.



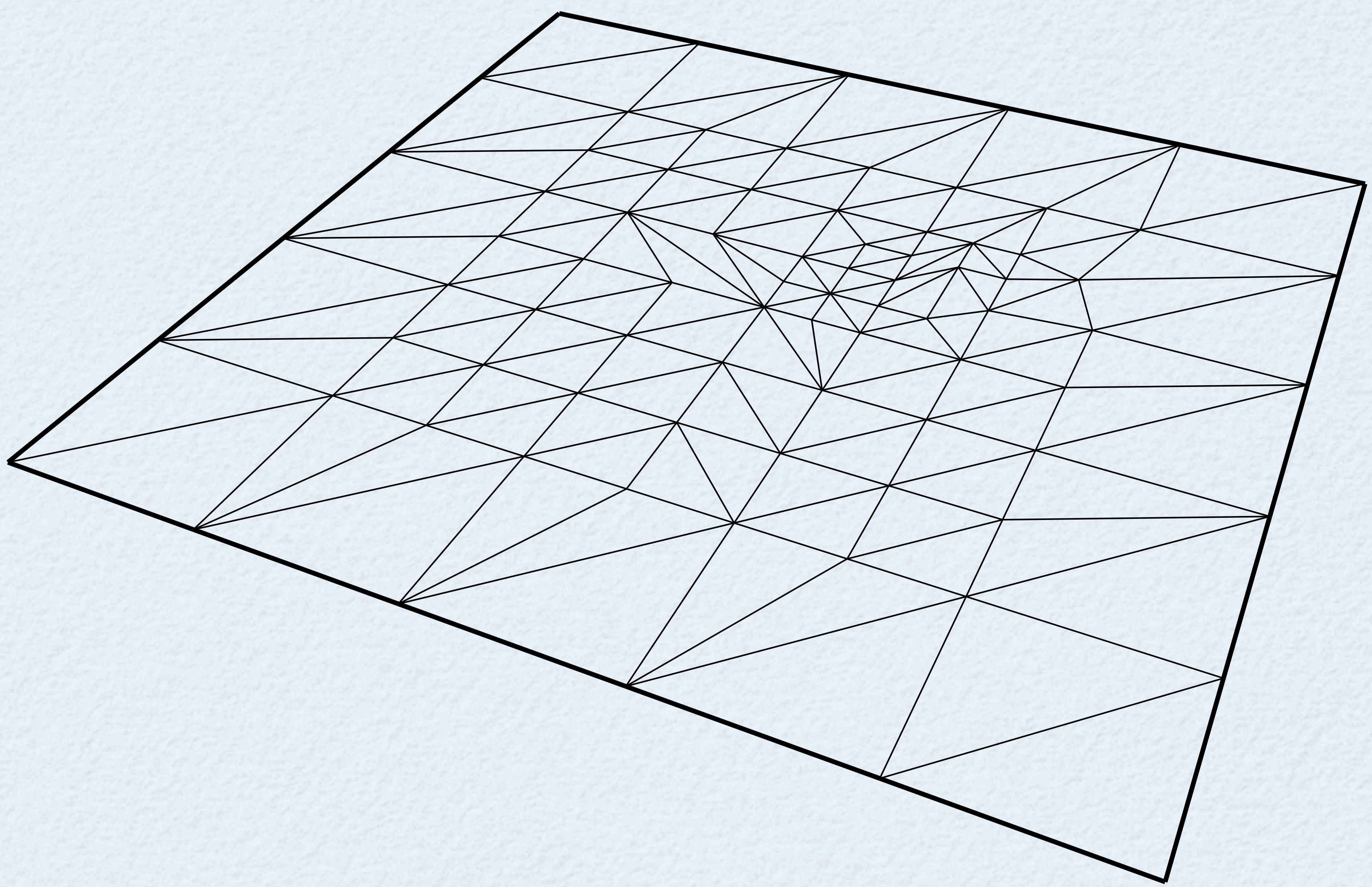
In practice terrains often interpolated sampled data.

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

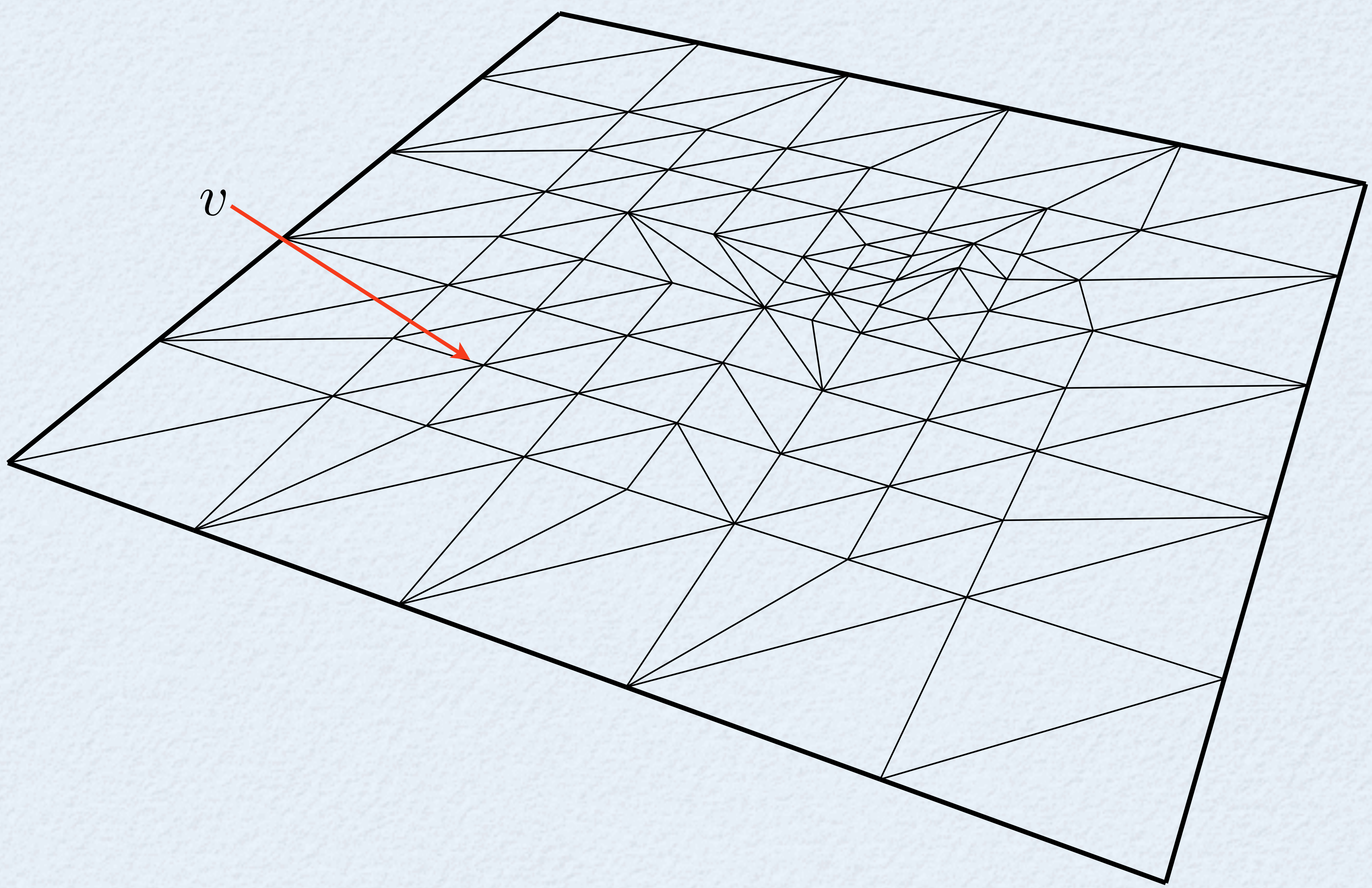
Representation: Triangulated Irregular Network (TIN)

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 $\mathbb{M}$.



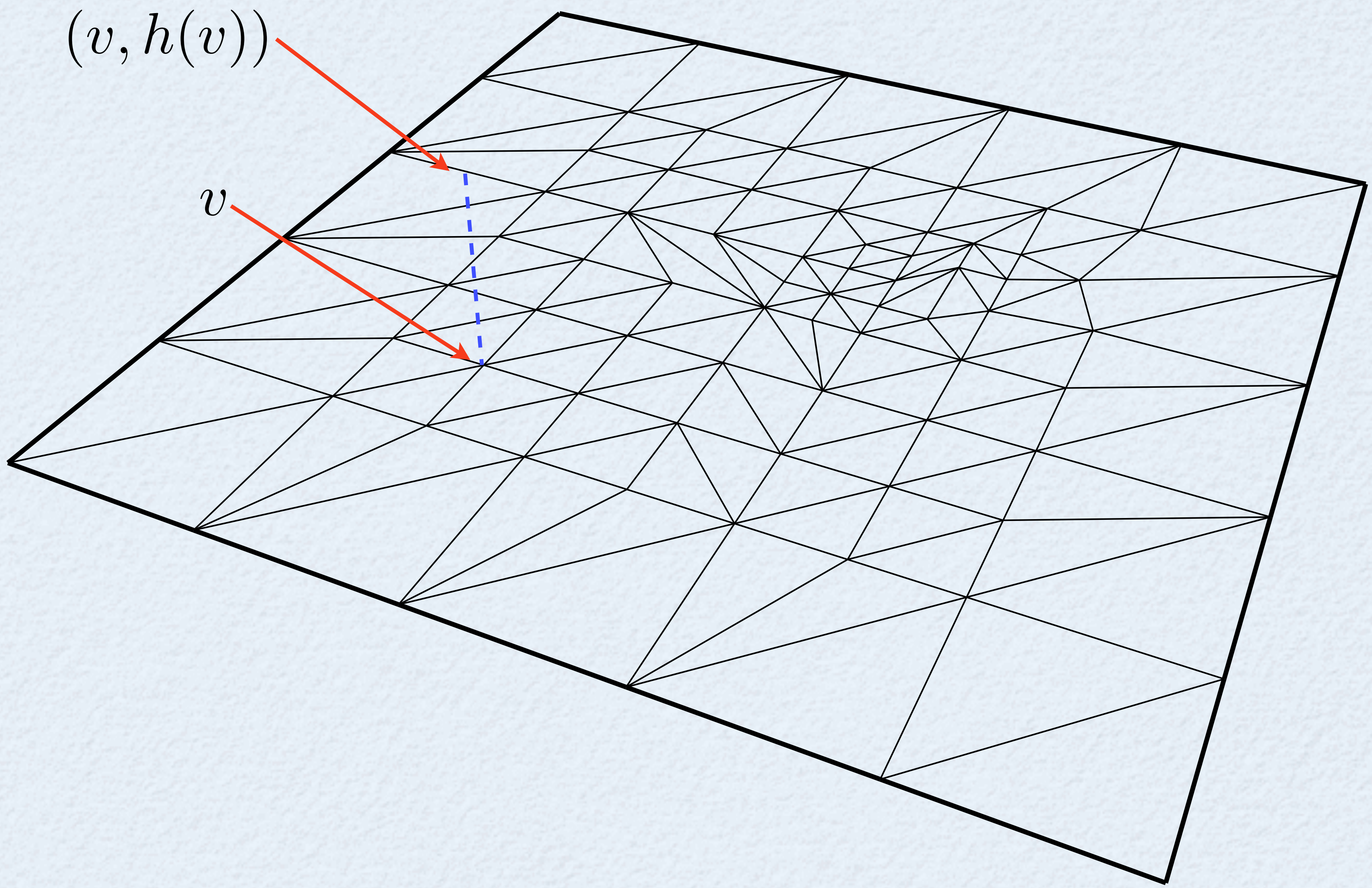
Representation: Triangulated Irregular Network (TIN)

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



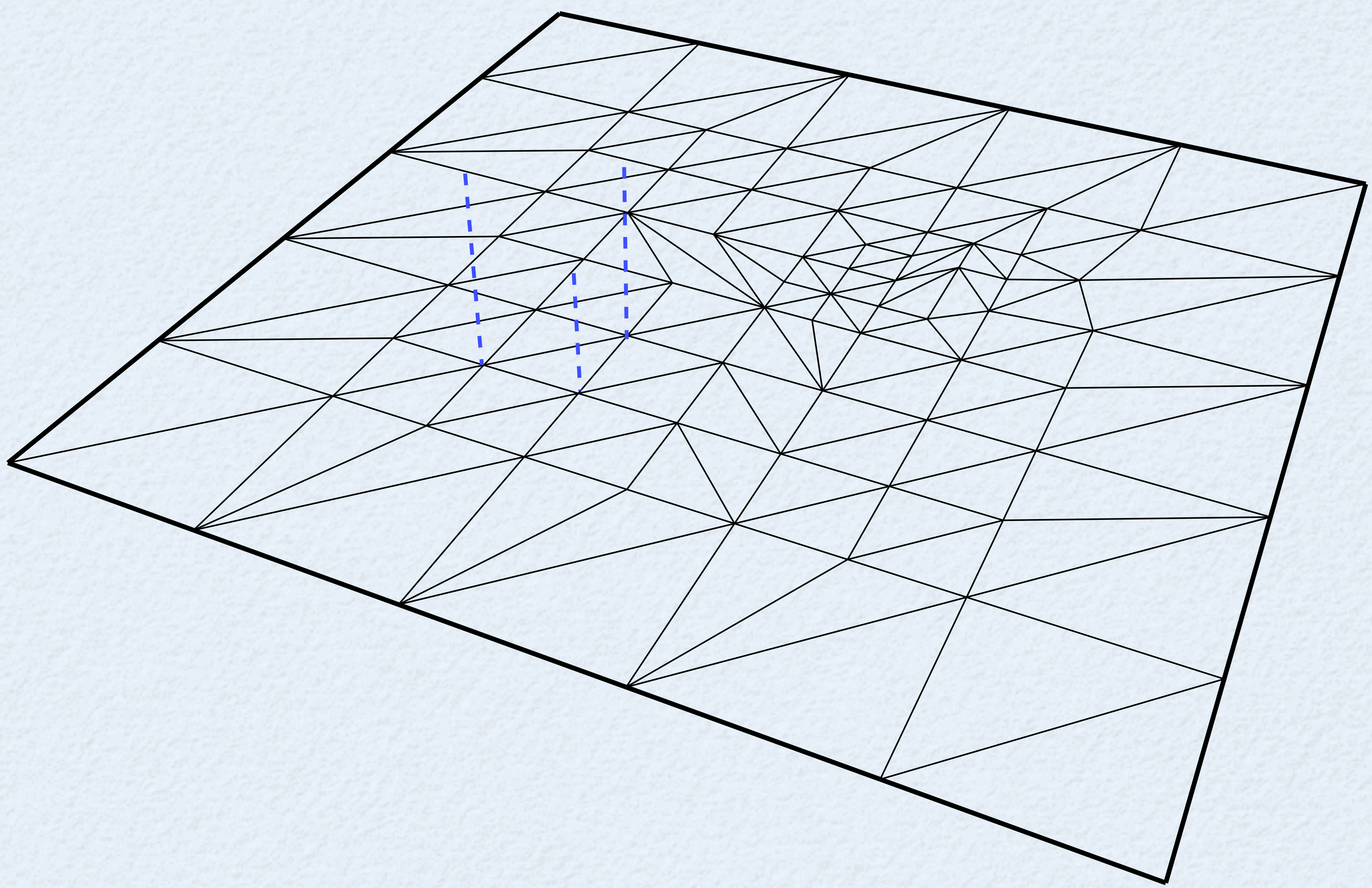
Representation: Triangulated Irregular Network (TIN)

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



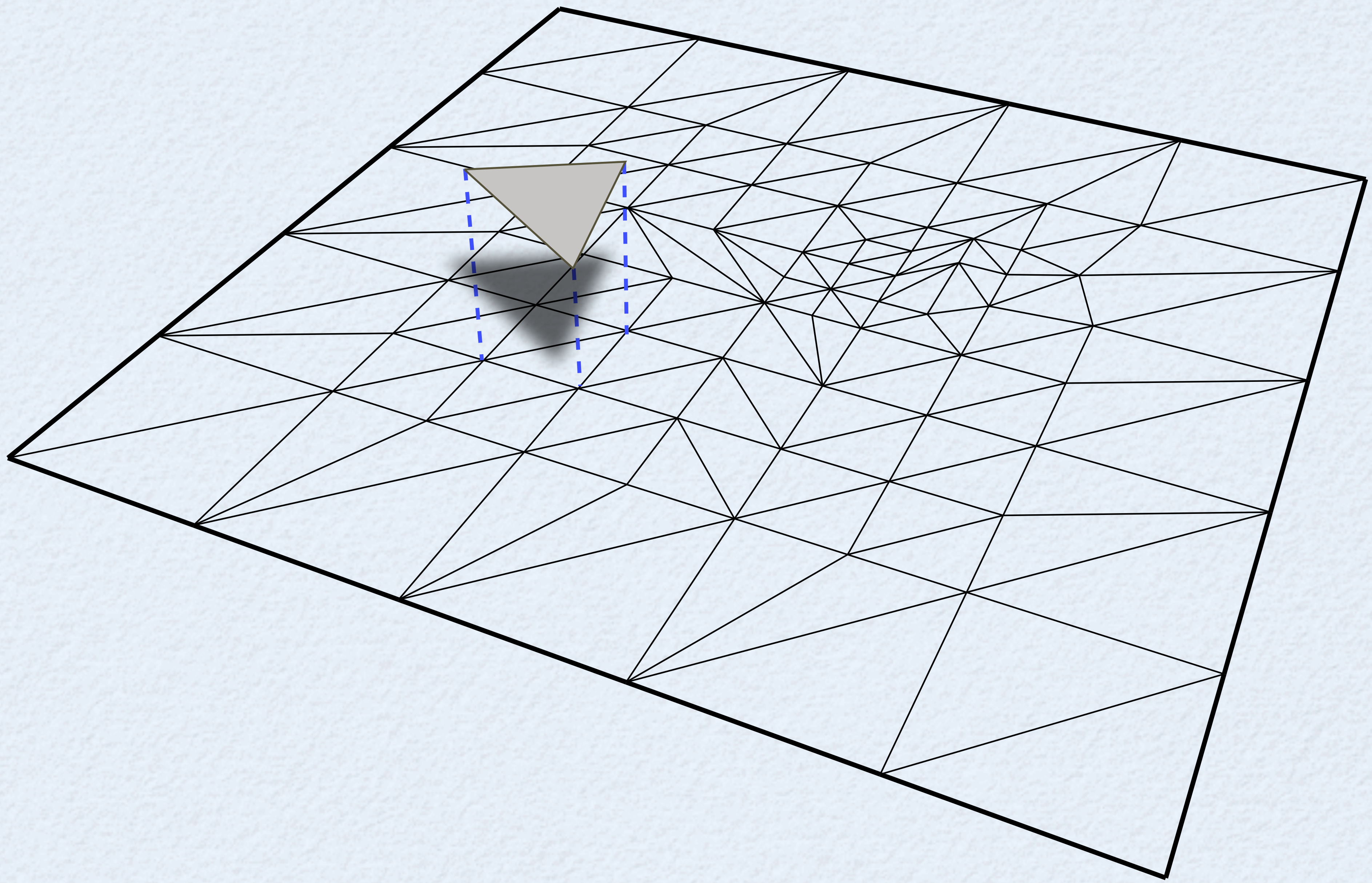
Representation: Triangulated Irregular Network (TIN)

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



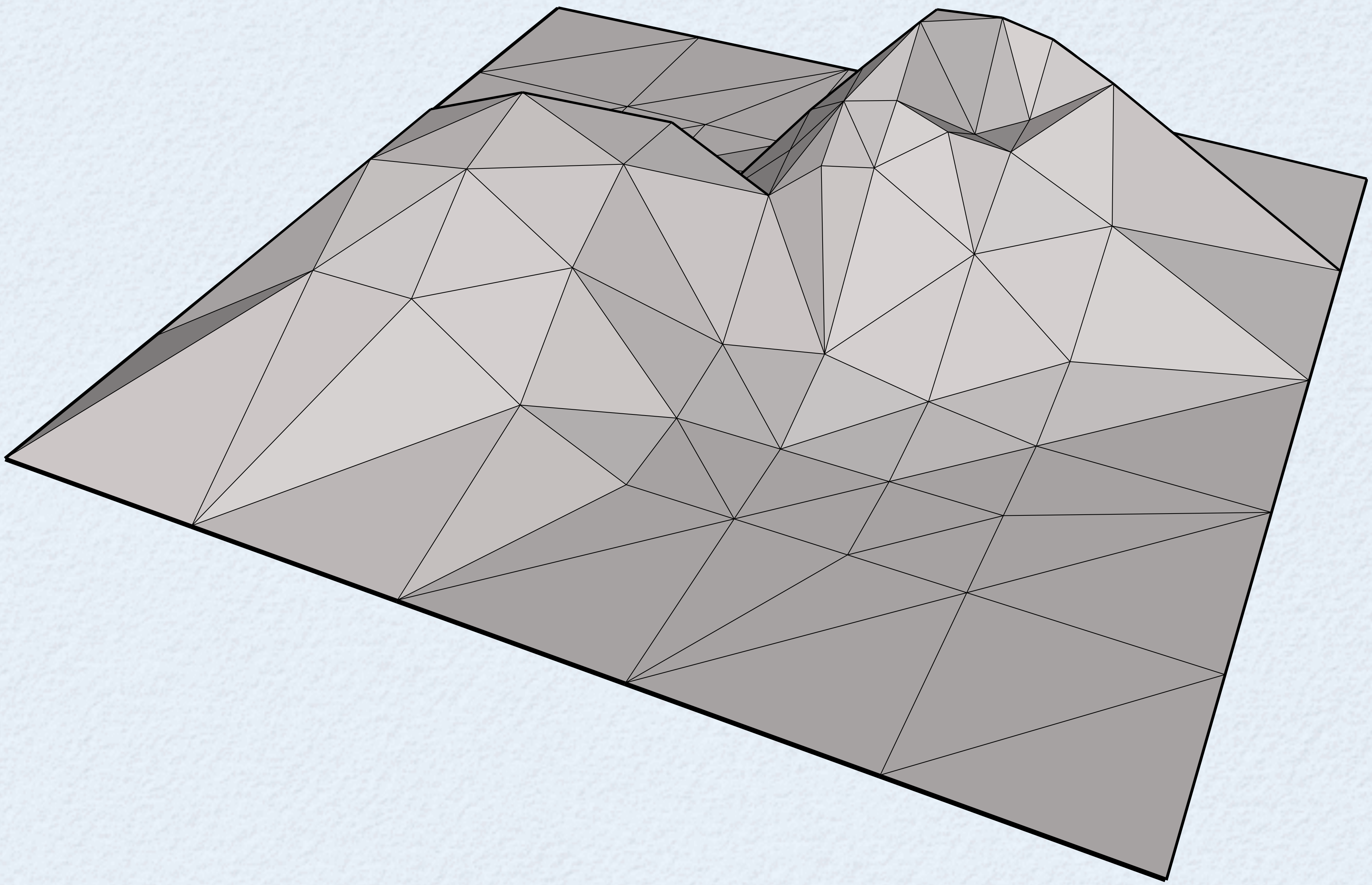
Representation: Triangulated Irregular Network (TIN)

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



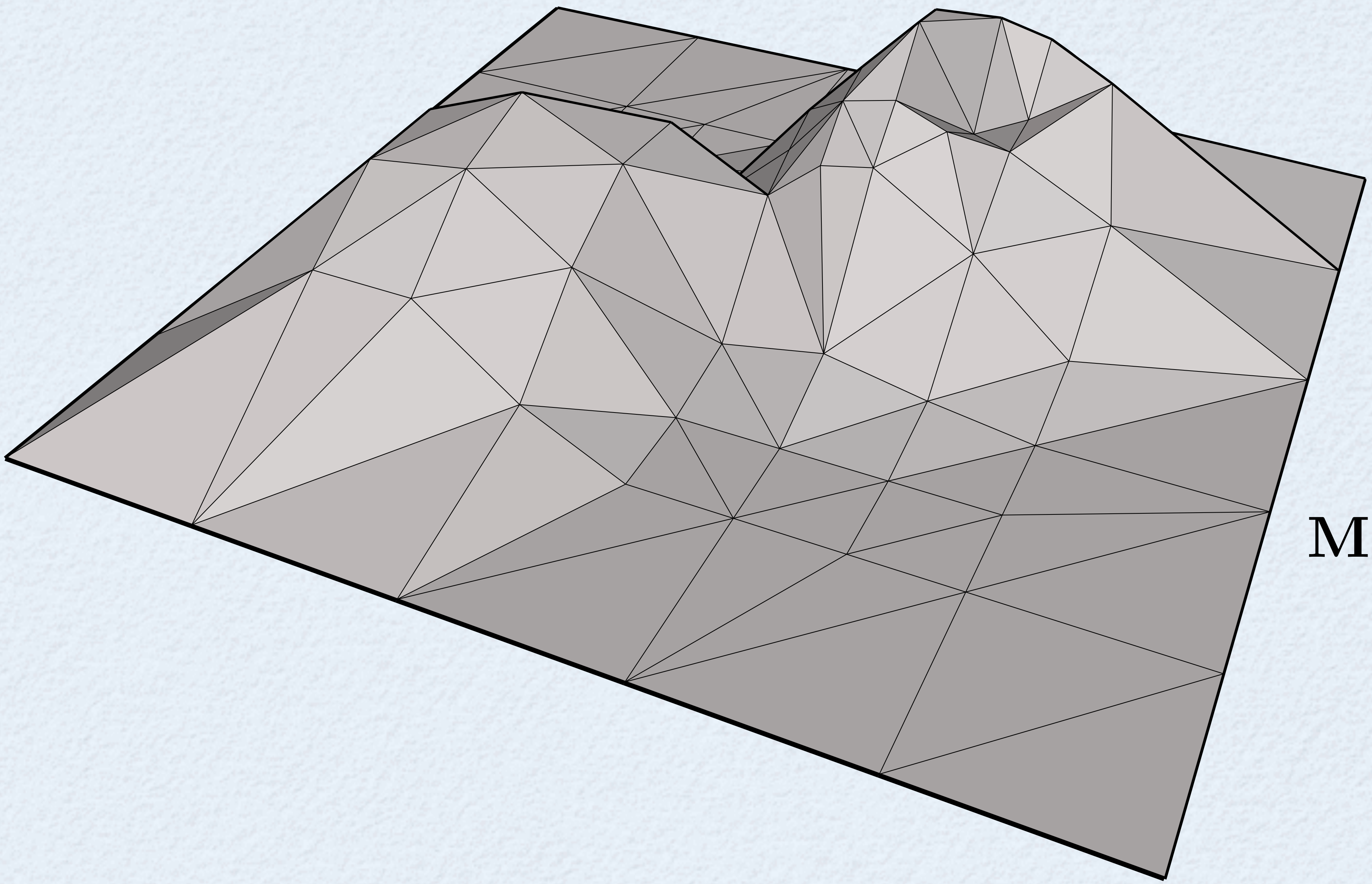
Representation: Triangulated Irregular Network (TIN)

Given a plane triangulation $\mathbb{M}$ 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)

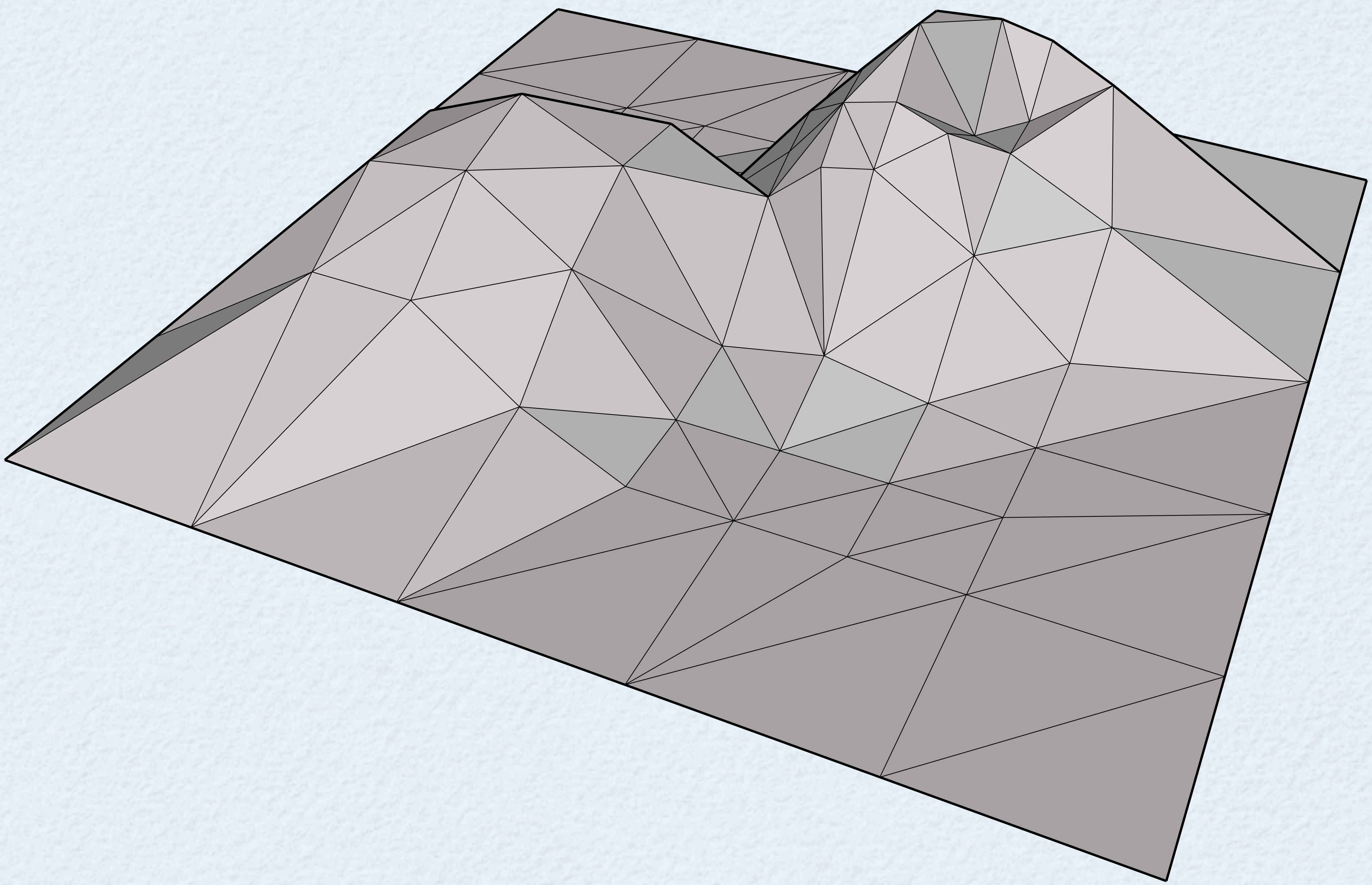
Given a plane triangulation $\mathbb{M}$ with a height $h(v)$ for each vertex v, one can linearly interpolate h in the interior of every face of $\mathbb{M}$.



$\mathbf{M} = (\mathbb{M}, h)$

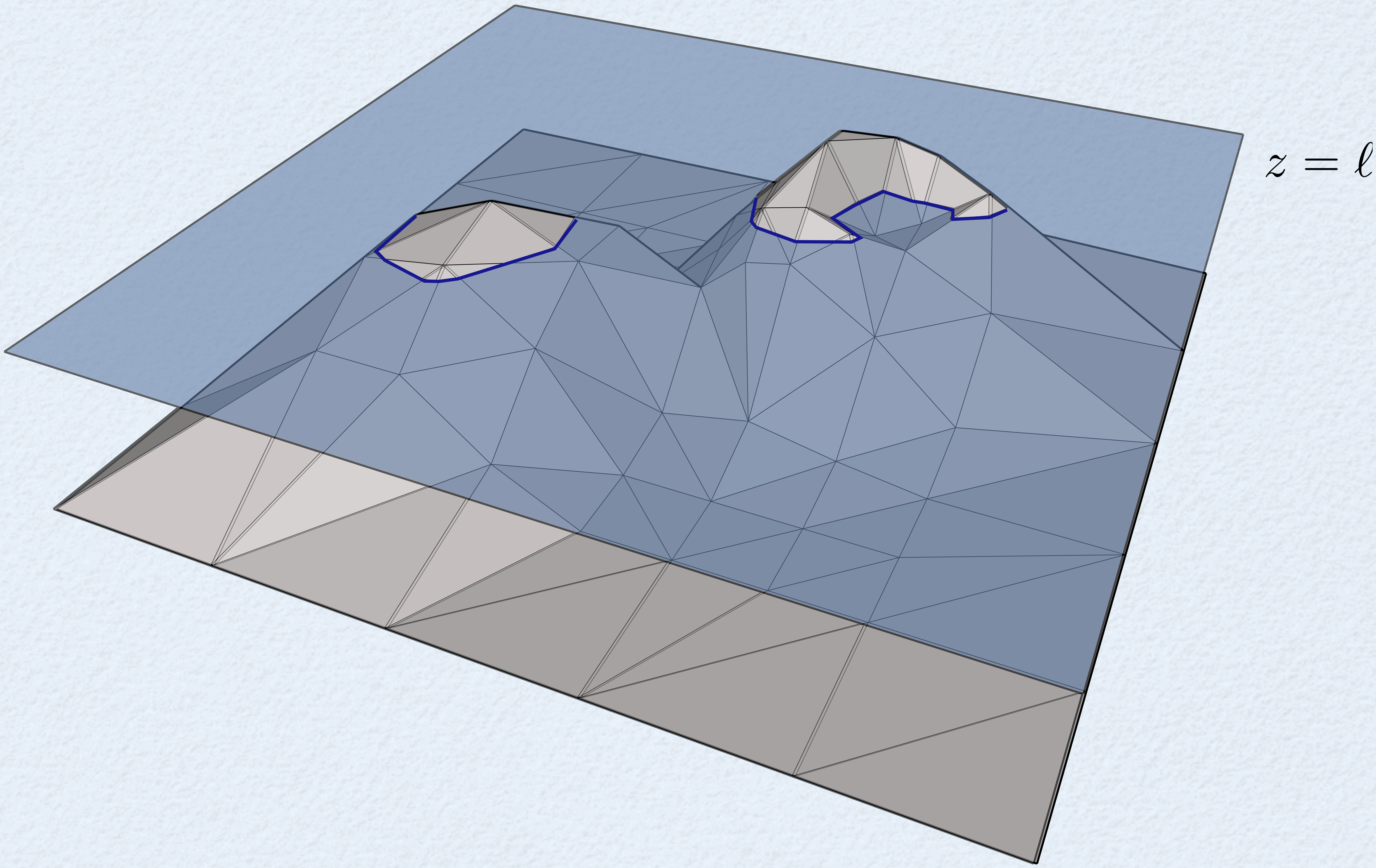
Level Sets, Contours, and Contour Maps

The level-set $\mathbb{M}_\ell$ at height ℓ is $h^{-1}(\ell)$.



Level Sets, Contours, and Contour Maps

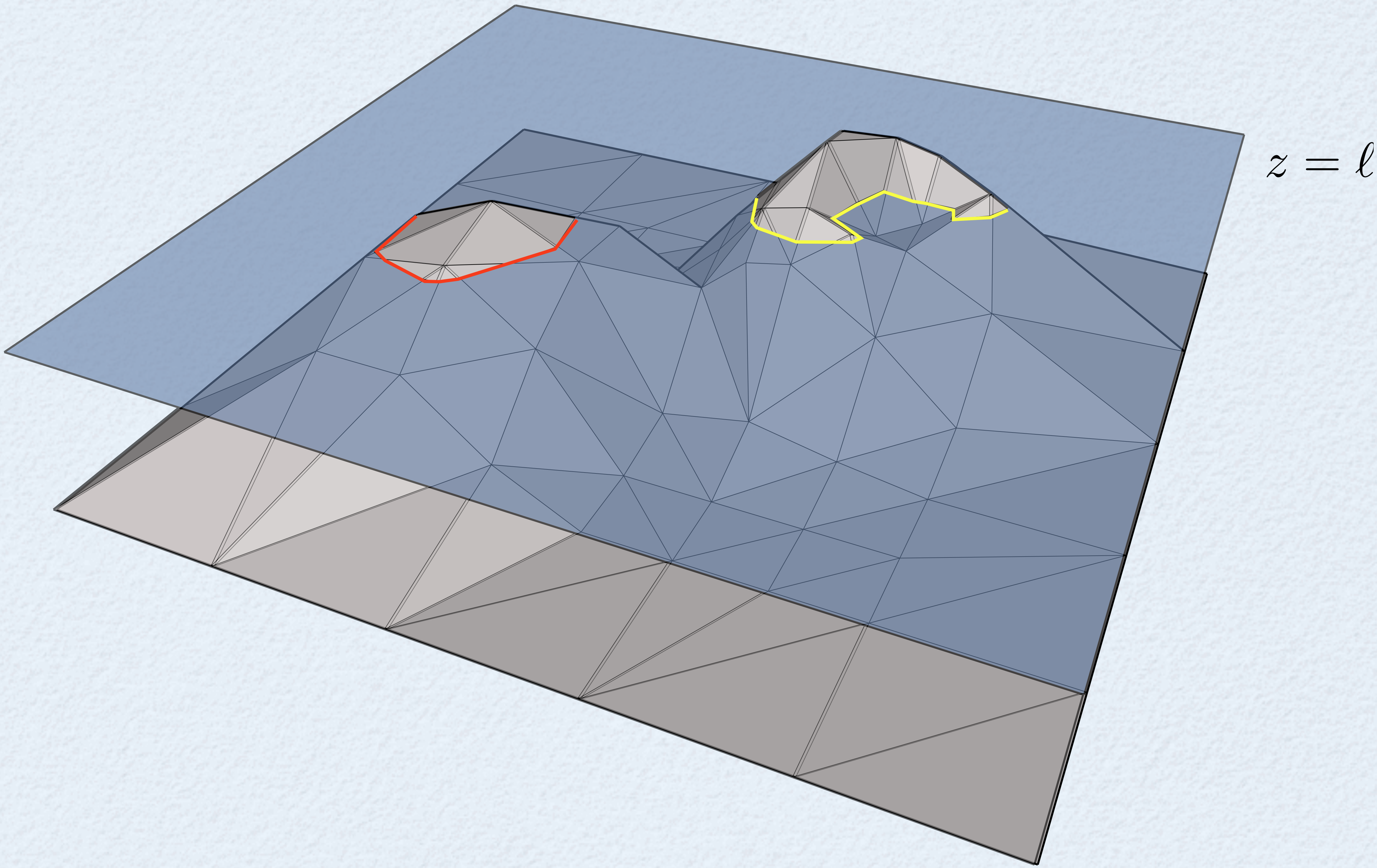
The level-set $\mathbb{M}_\ell$ at height ℓ is $h^{-1}(\ell)$.



Level Sets, Contours, and Contour Maps

The level-set $\mathbb{M}_\ell$ at height ℓ is $h^{-1}(\ell)$.

Each connected component of a level set is called a contour.

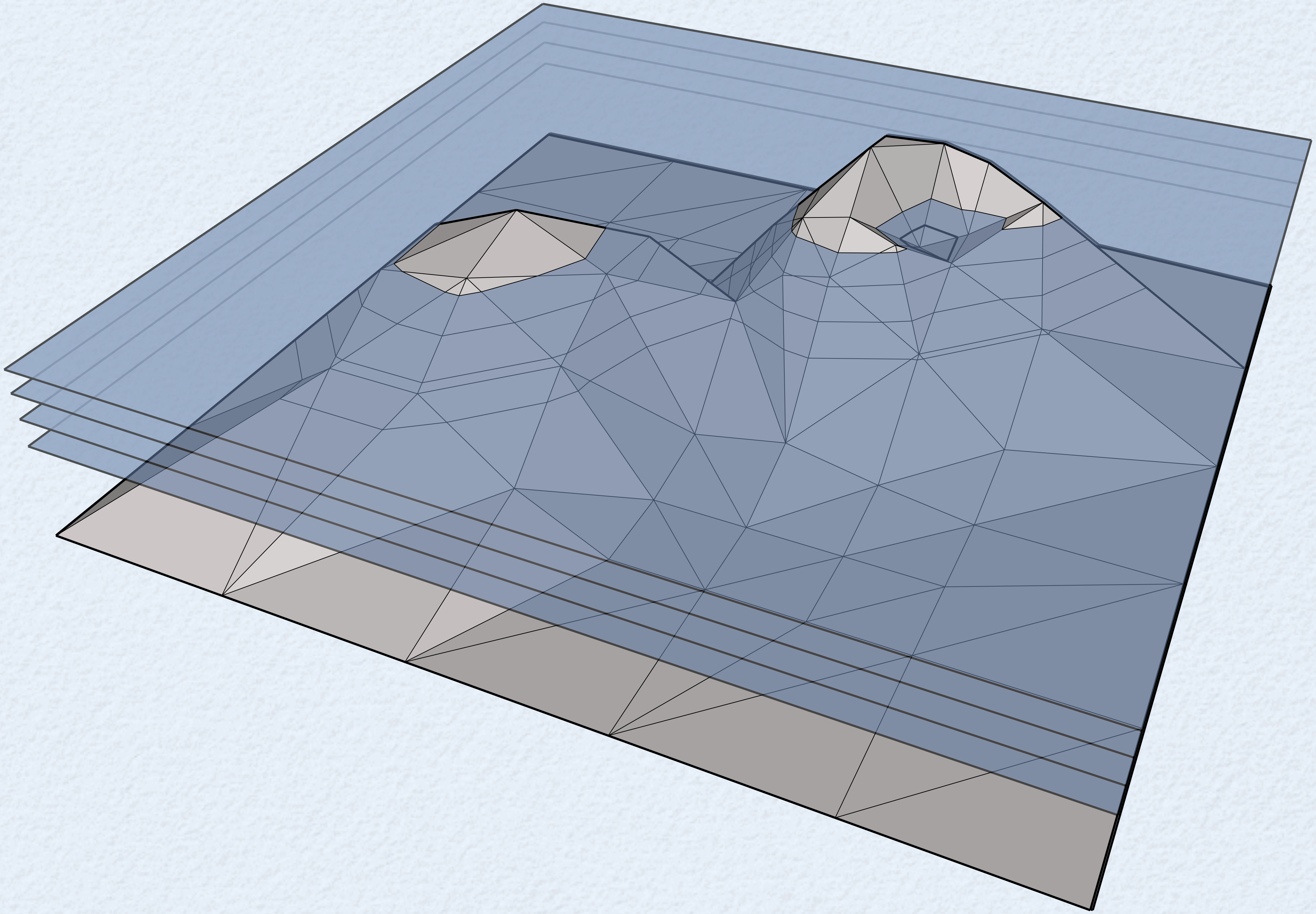


Level Sets, Contours, and Contour Maps

The level-set $\mathbb{M}_\ell$ at height ℓ is $h^{-1}(\ell)$.

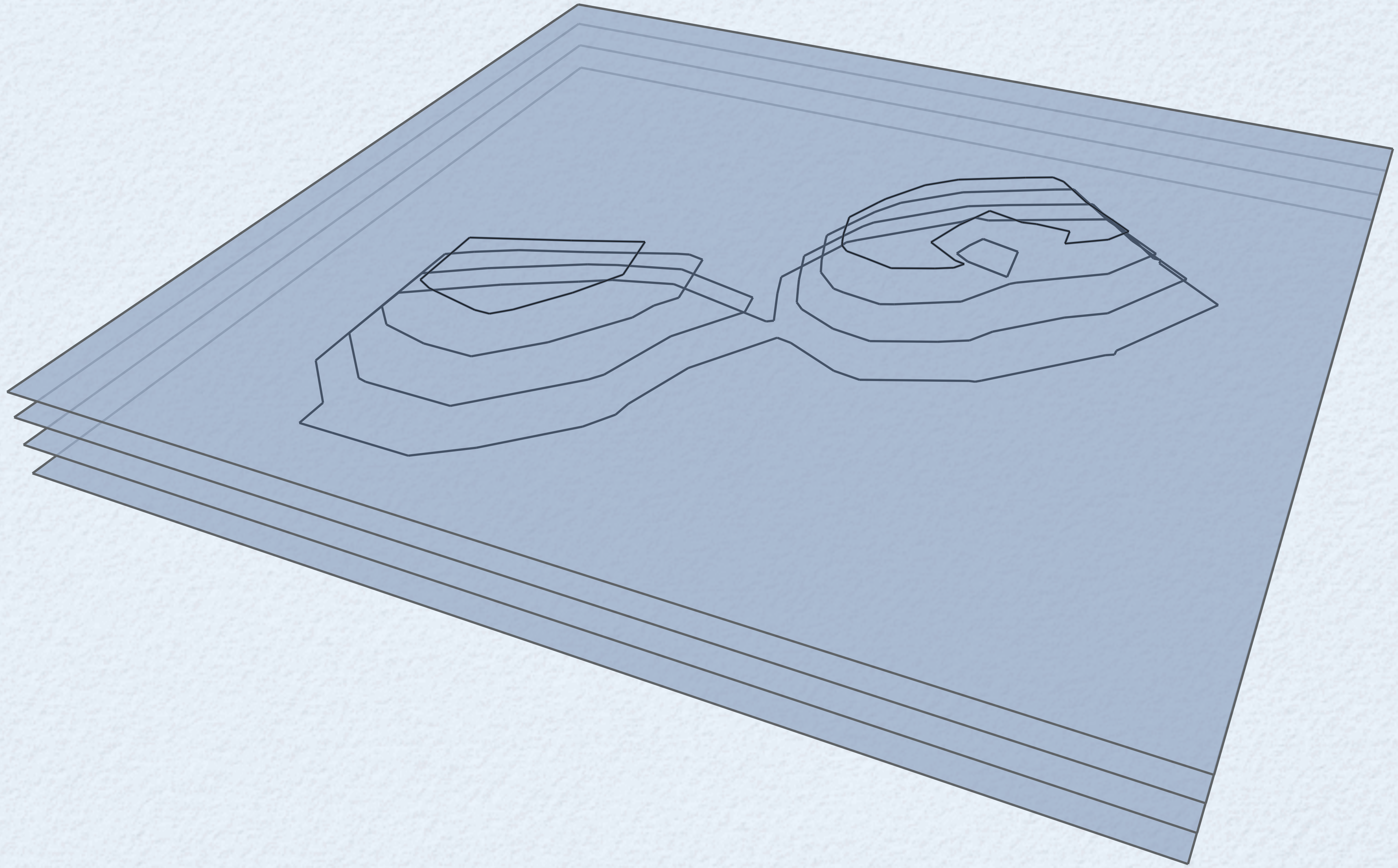
Each connected component of a level set is called a contour.

Given levels $L = \{\ell_1, \ldots, \ell_k\}$, the contour map is $h^{-1}(L)$.



Level Sets, Contours, and Contour Maps

The level-set $\mathbb{M}_\ell$ at height ℓ 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 $h^{-1}(L)$.

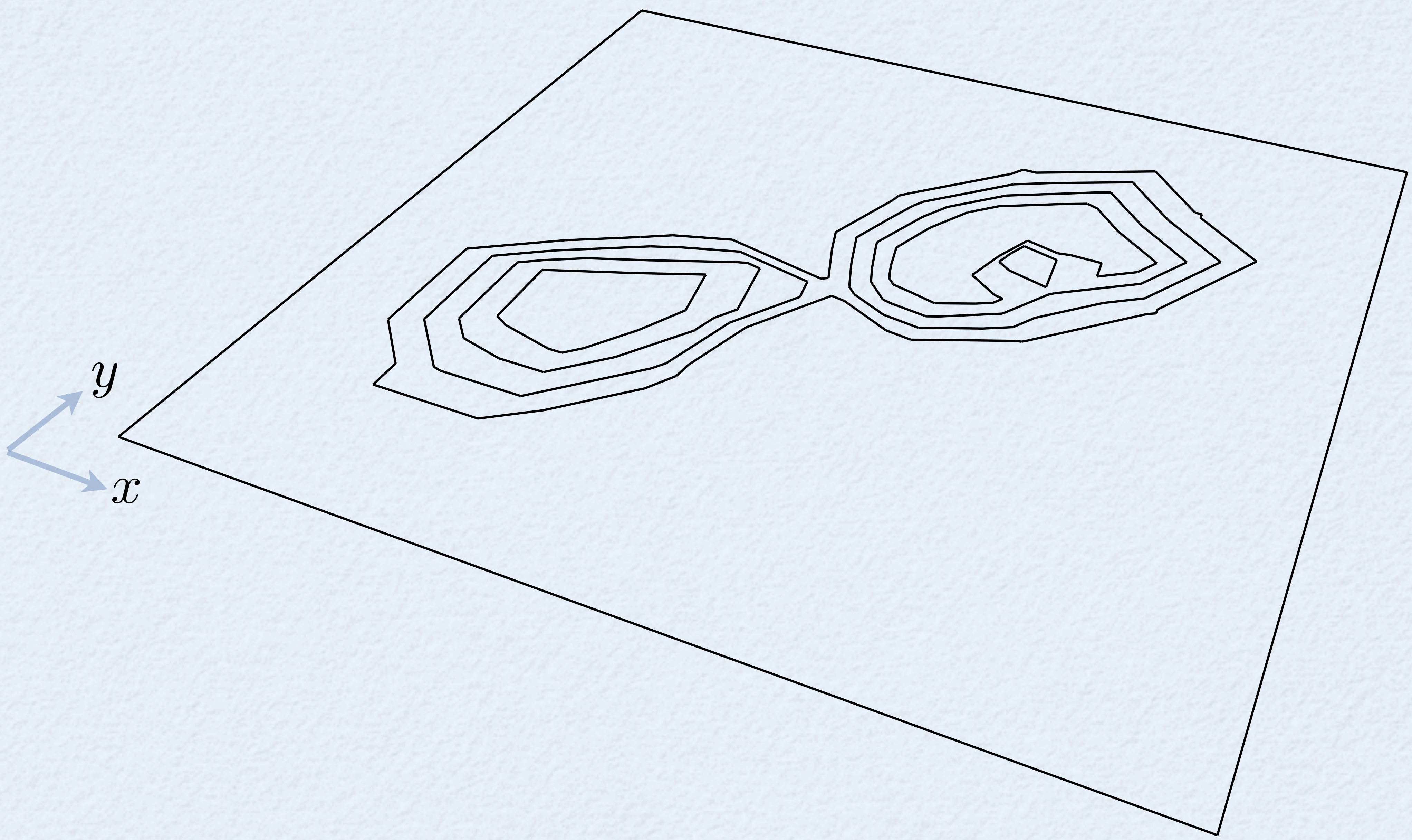


Level Sets, Contours, and Contour Maps

The level-set $\mathbb{M}_\ell$ at height ℓ is $h^{-1}(\ell)$.

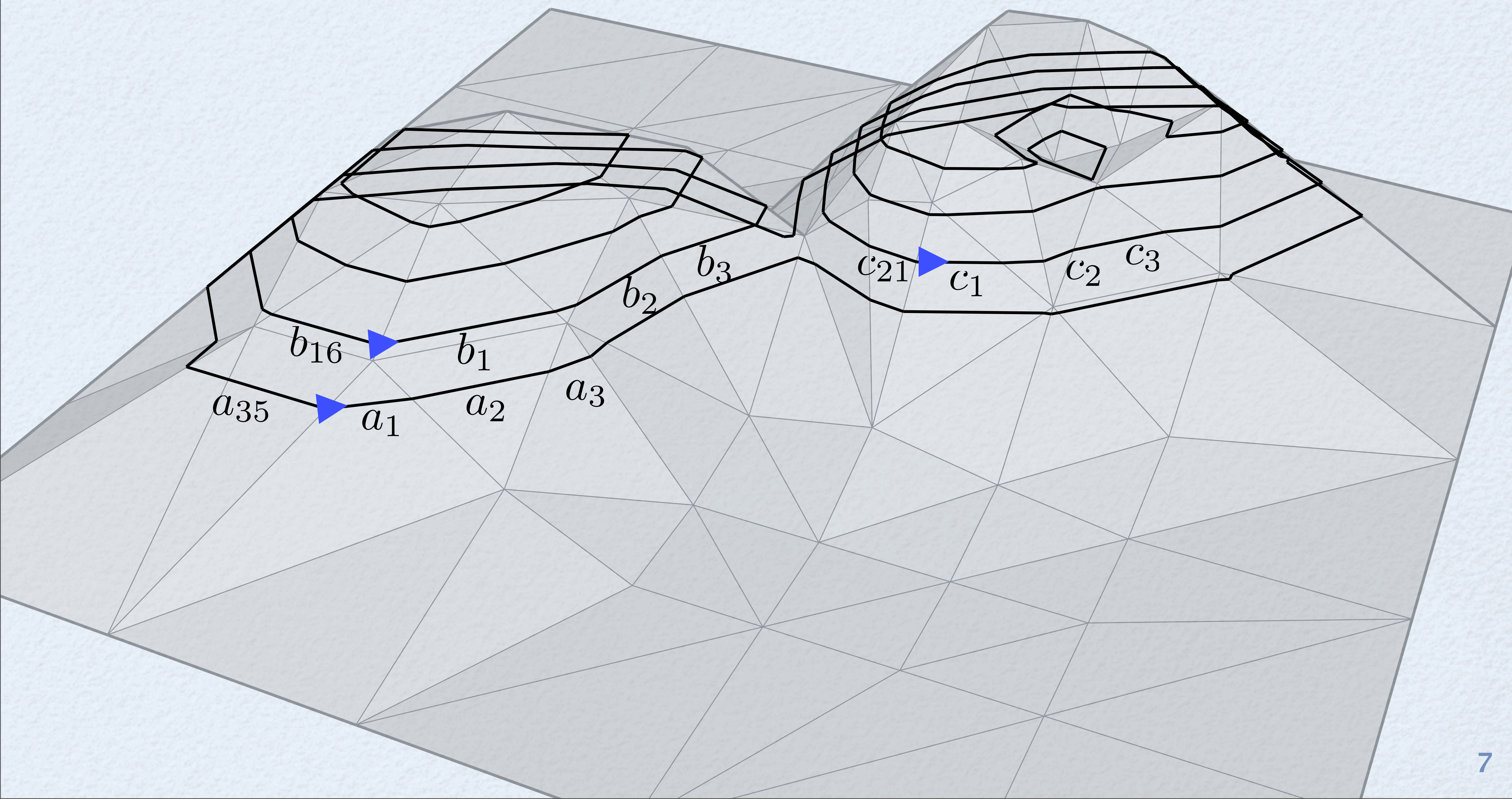
Each connected component of a level set is called a contour.

Given levels $L = \{\ell_1, \ldots, \ell_k\}$, the contour map is $h^{-1}(L)$.



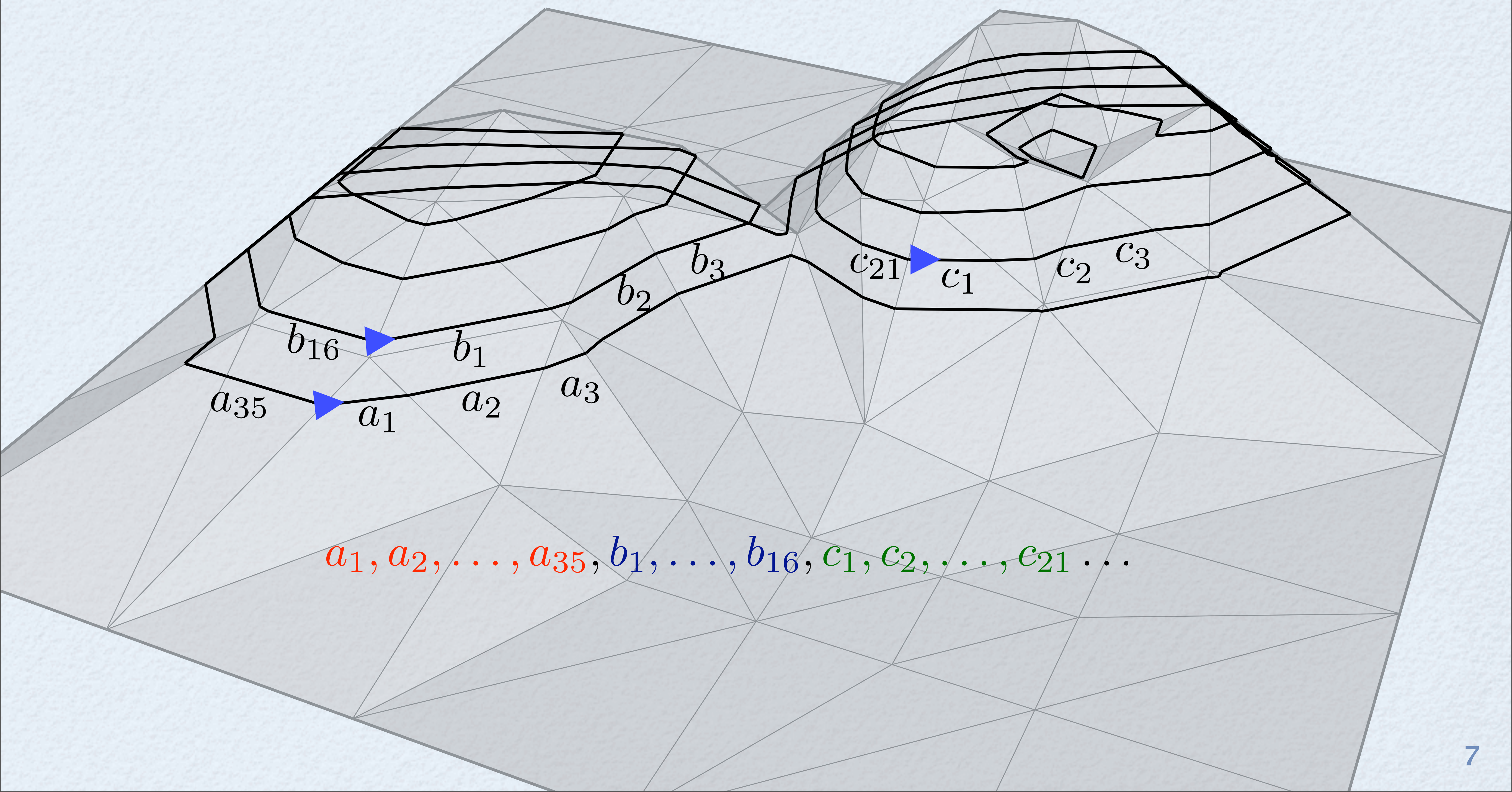
Computing Contour Maps

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.



Computing Contour Maps

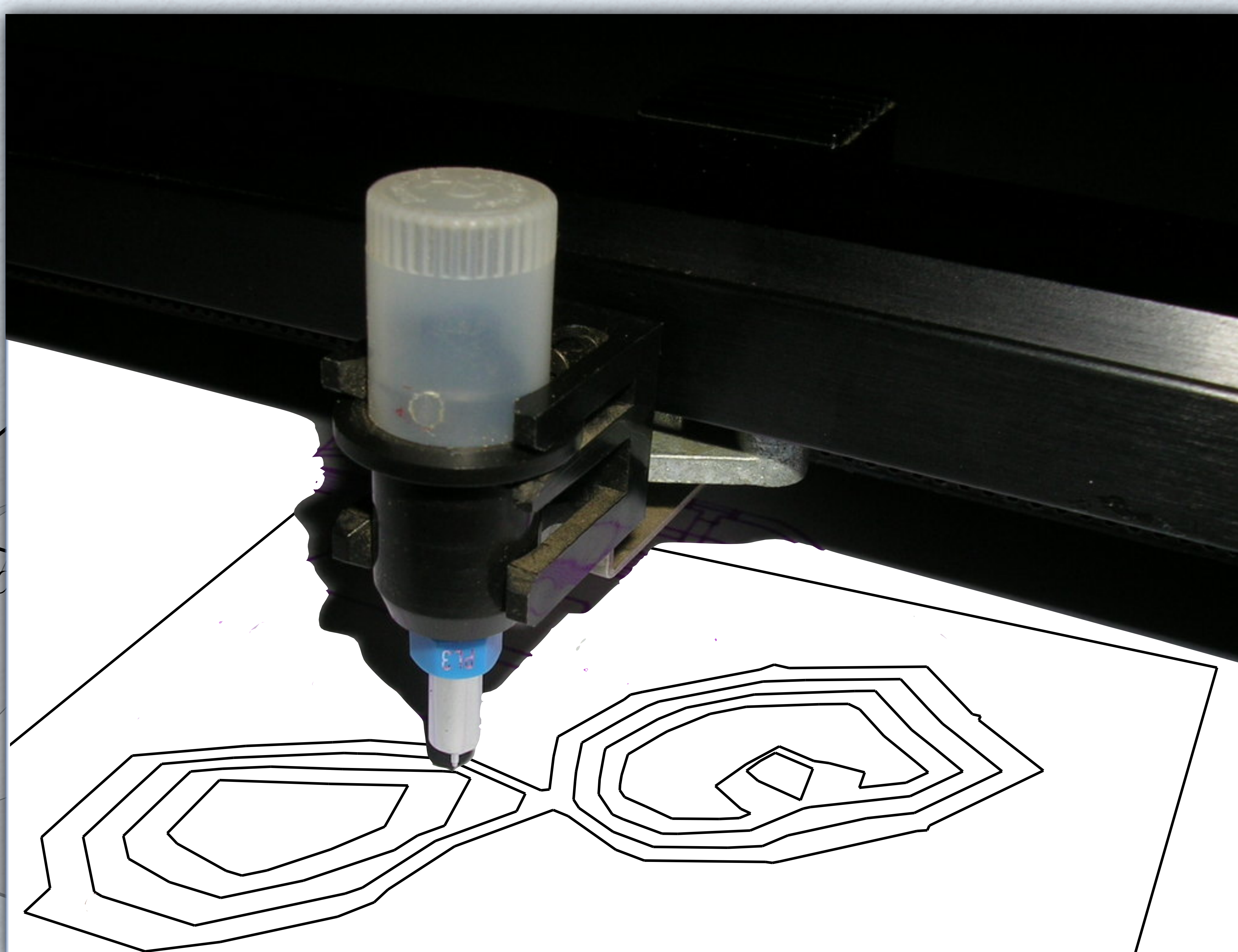
Given a set of levels $L = \{\ell_1, \ldots, \ell_k\}$, compute the contour map $h^{-1}(L)$ such that each contour is reported separately and in sorted (circular) order.



$a_1, a_2, \ldots, a_{35}, b_1, \ldots, b_{16}, c_1, c_2, \ldots, c_{21} \ldots$

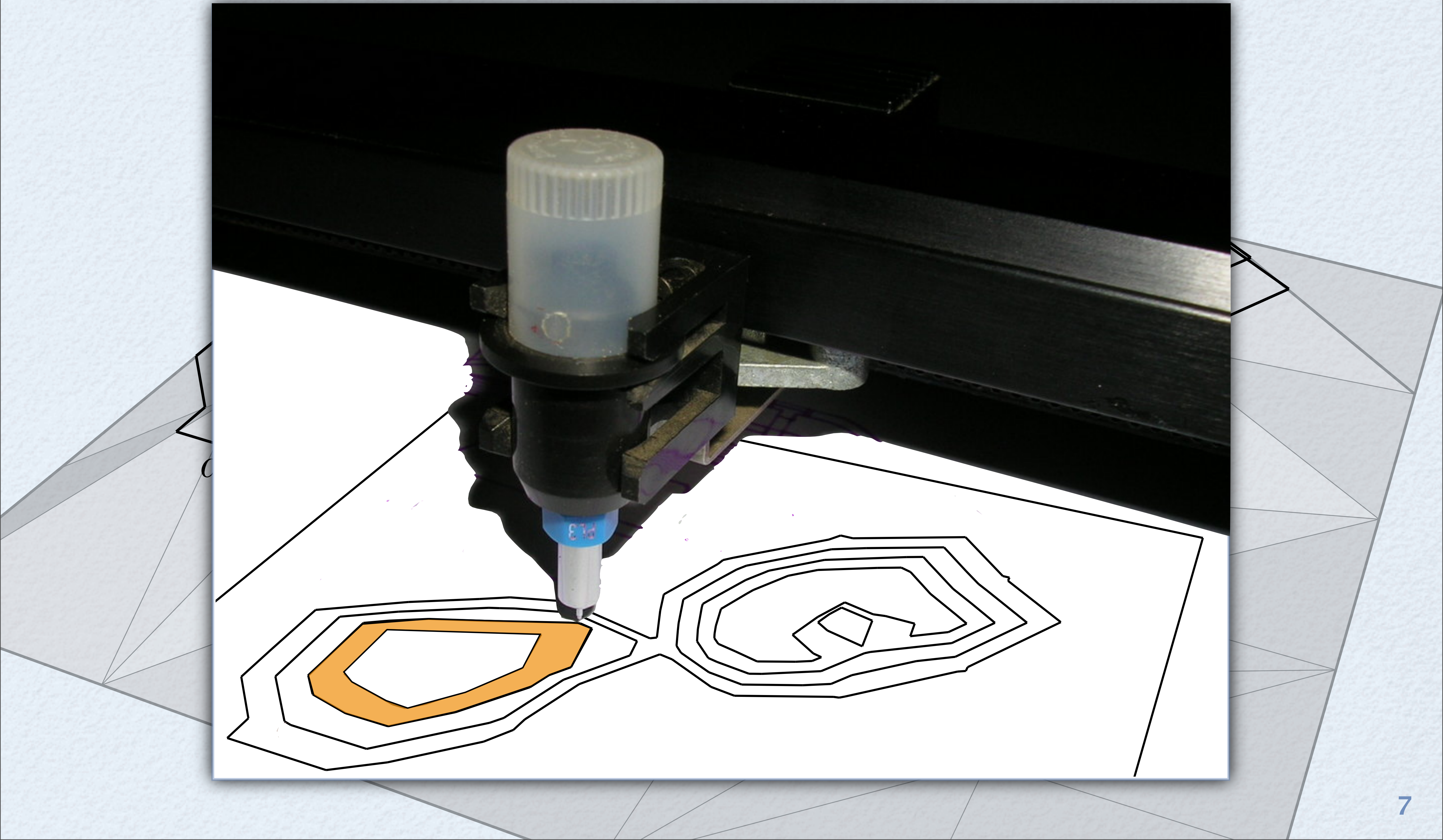
Computing Contour Maps

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.



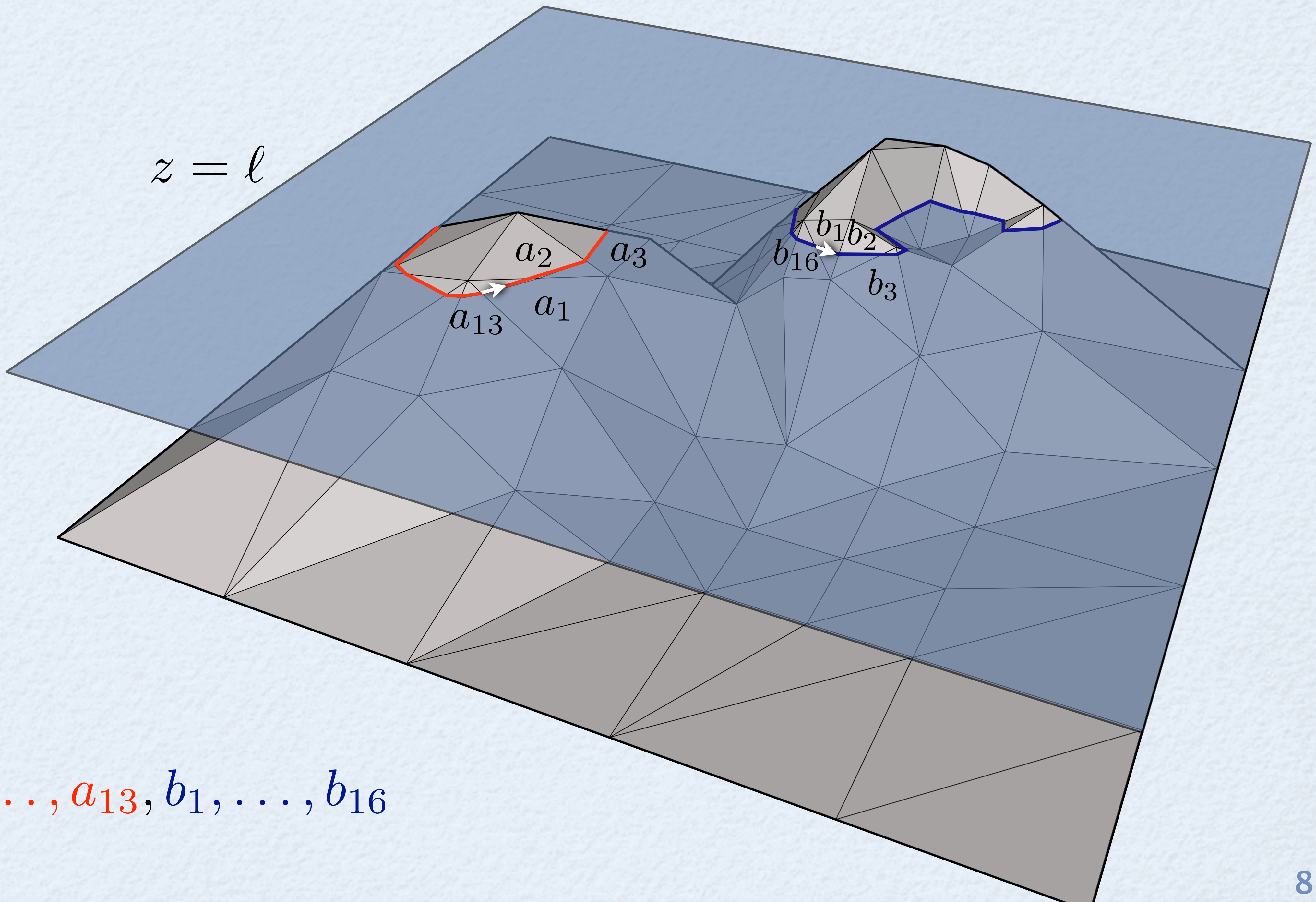
Computing Contour Maps

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.



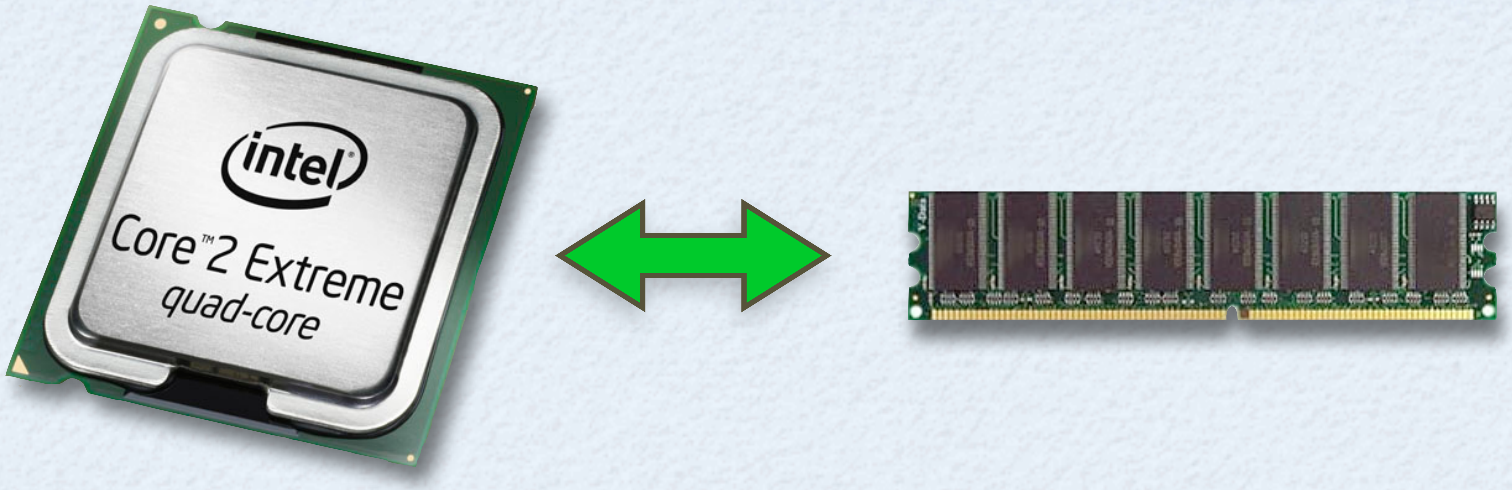
Answering Contour Queries

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.*



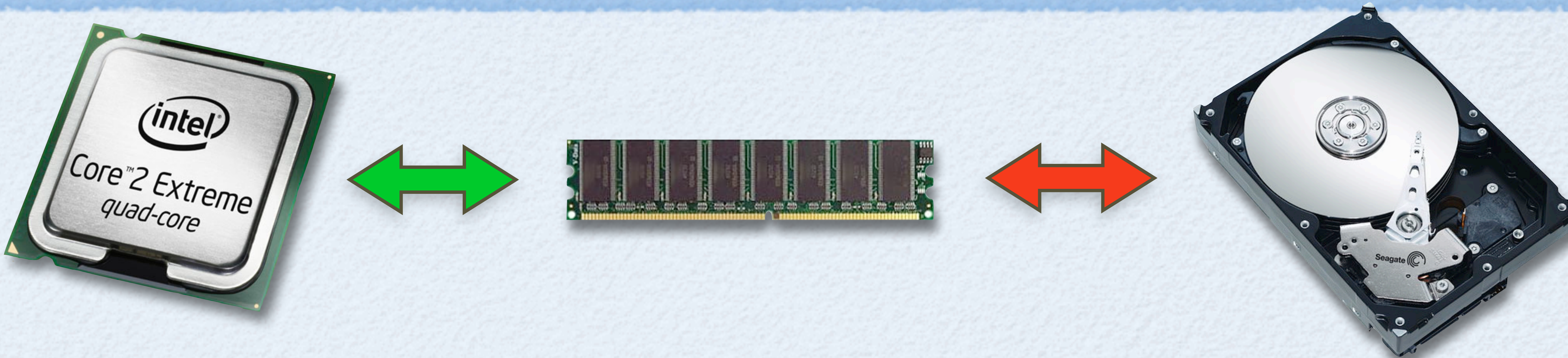
Output: $a_1, a_2, \ldots, a_{13}, b_1, \ldots, b_{16}$

The I/O Model



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

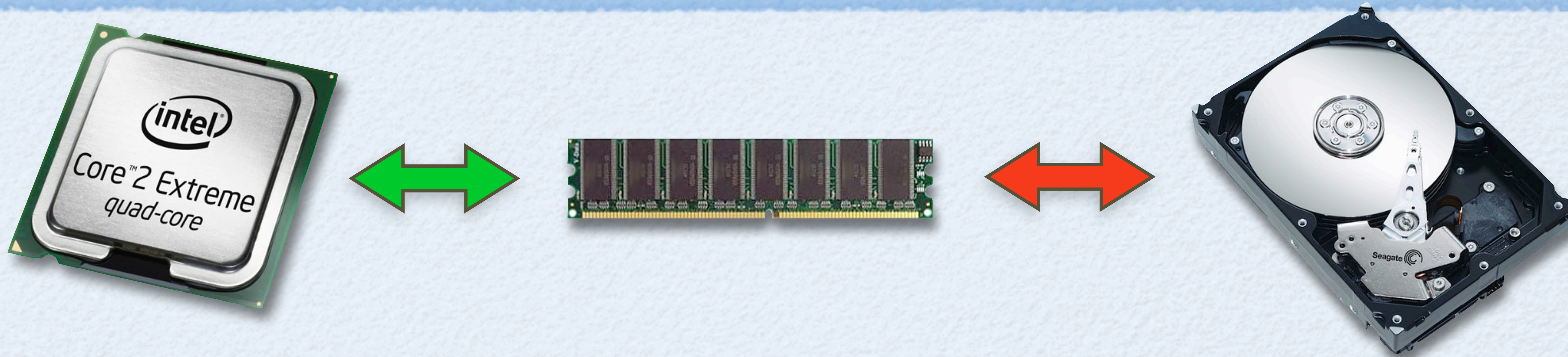
The I/O Model



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

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

The I/O Model

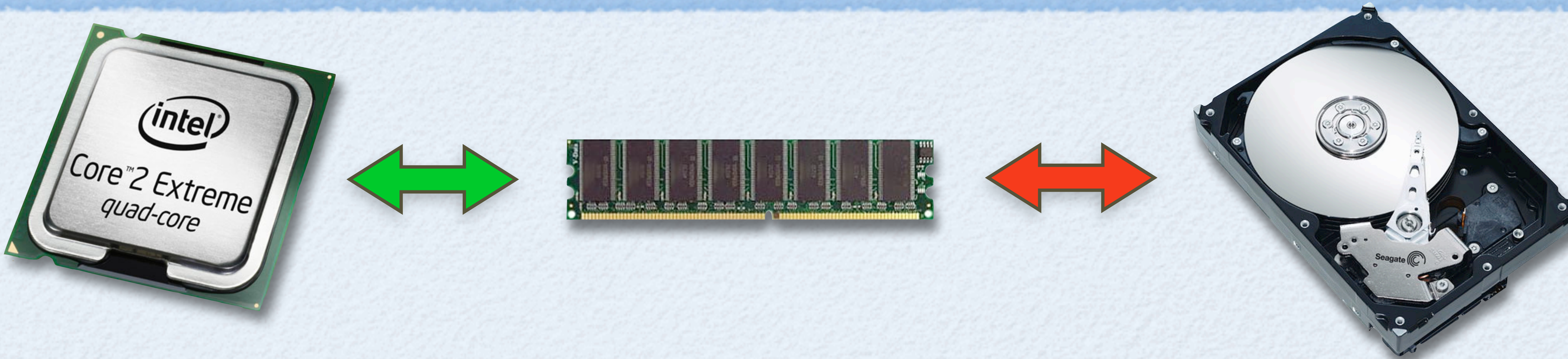


Classical Complexity: Number of basic operations as a function of N.

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

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

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.

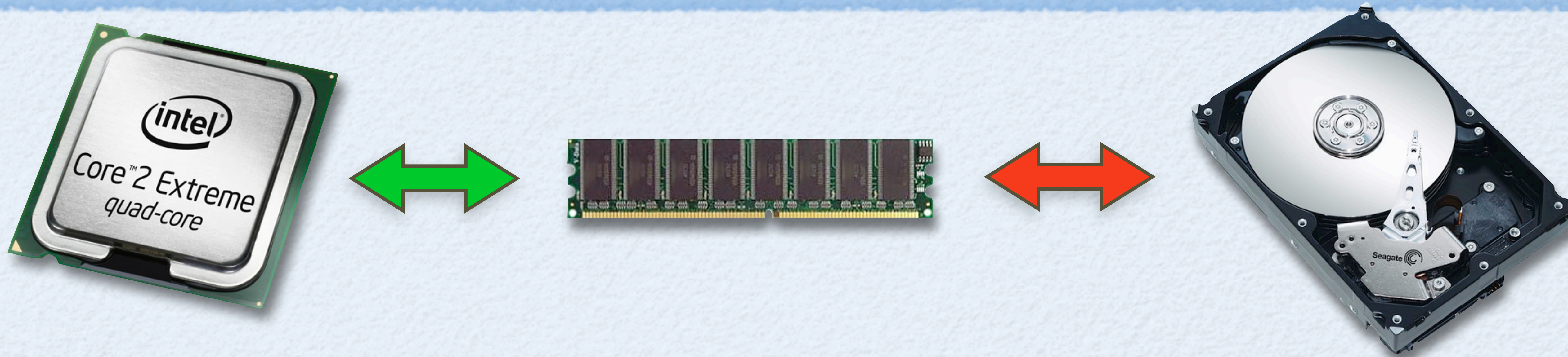
N = number of items in the problem instance

B = number of items per disk block

M = number of items that fit main memory

T = number of items in output

The I/O Model



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

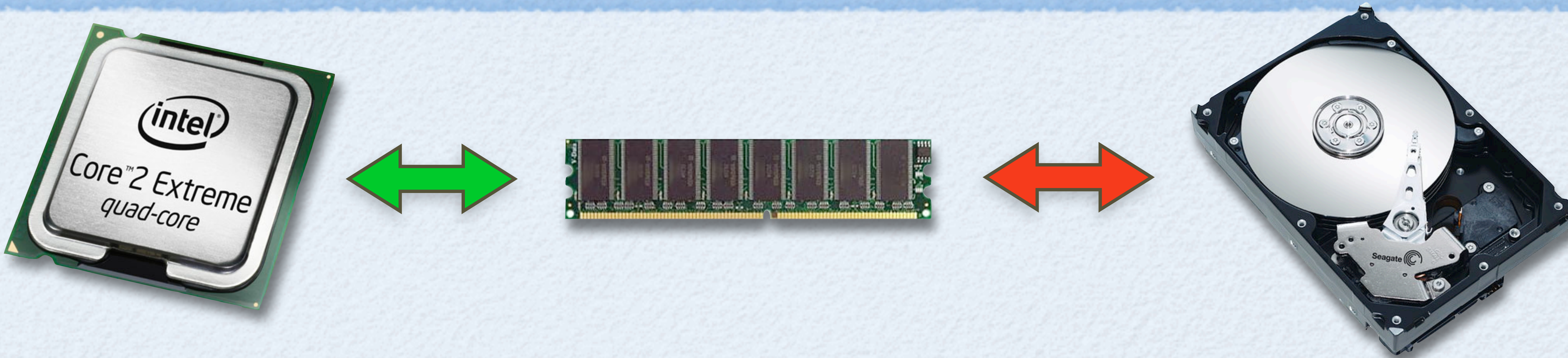
Disk access is about 10^6 times slower than main memory access.

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

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

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

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.

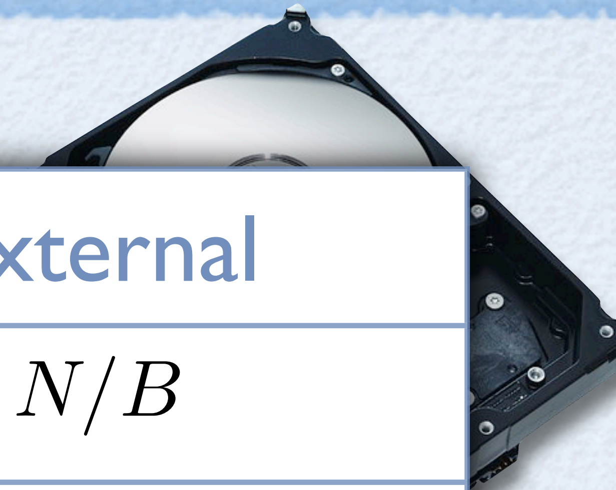
To amortize access delay, disks transfer large contiguous blocks of data.

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

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.

The I/O Model



	internal	external
Scanning	N	N/B
Sorting	$N \log N$	$\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}$
Permuting	N	$\min\left\{N, \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right\}$
Searching	$\log_2 N$	$\log_B N$

Class

Disk

To a

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

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.

The I/O Model

	internal	external
Scanning	N	N/B
Sorting	$N \log N$	$\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}$
Permuting	N	$\min\left\{N, \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right\}$
Searching	$\log_2 N$	$\log_B N$

Sort(N)

Class

Disk

To a

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

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

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

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

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.

The I/O Model

	internal	external
Scanning	N	N/B
Sorting	$N \log N$	$\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B} \ll N$
Permuting	N	$\min\left\{N, \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right\}$
Searching	$\log_2 N$	$\log_B N$

Sort(N)

Class

Disk

To a

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:

	Preprocessing I/Os	Structure Size	Query I/Os
Chiang, Silva'97	$O(\mathsf{Sort}(N))$	$O(N)$	$O(\log_B N + T/B)$
Agarwal, Arge, Murali, Varadrarajan, Vitter'98	$O(N)$	$O(N)$	$O(\log_B N + T/B)$

Previous Work

Answering contour queries I/O-efficiently:

	Preprocessing I/Os	Structure Size	Query I/Os
Chiang, Silva'97	$O(\text{Sort}(N))$	$O(N)$	$O(\log_B N + T/B)$
Agarwal, Arge, Murali, Varadrarajan, Vitter'98	$O(N)$	$O(N)$	$O(\log_B N + T/B)$

Previous Work

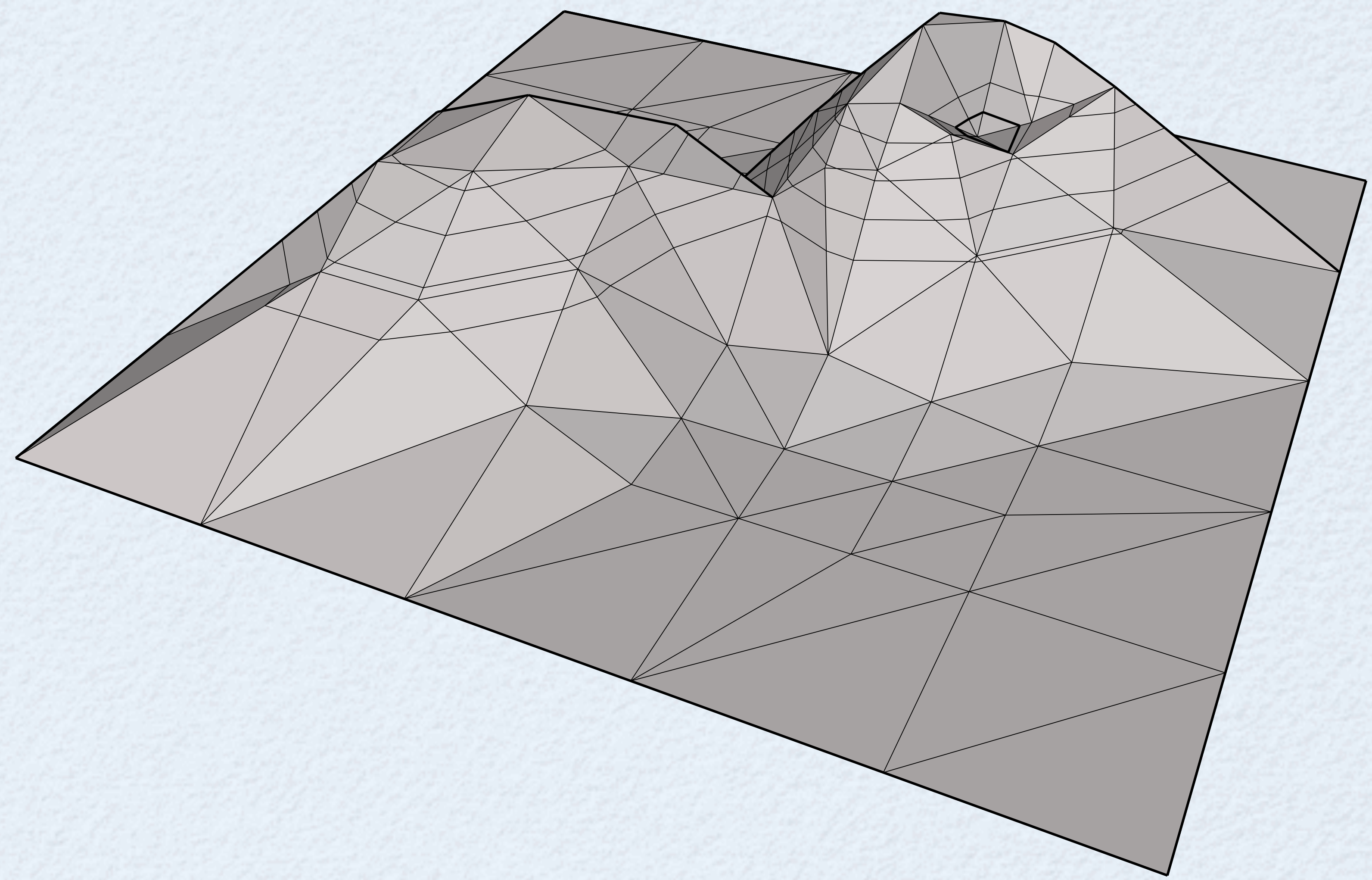
Answering contour queries I/O-efficiently:

	Preprocessing I/Os	Structure Size	Query I/Os
Chiang, Silva'97	$O(\text{Sort}(N))$	$O(N)$	$O(\log_B N + T/B)$
Agarwal, Arge, Murali, Varadrarajan, Vitter'98	$O(N)$	$O(N)$	$O(\log_B N + T/B)$

Unsorted Contours!

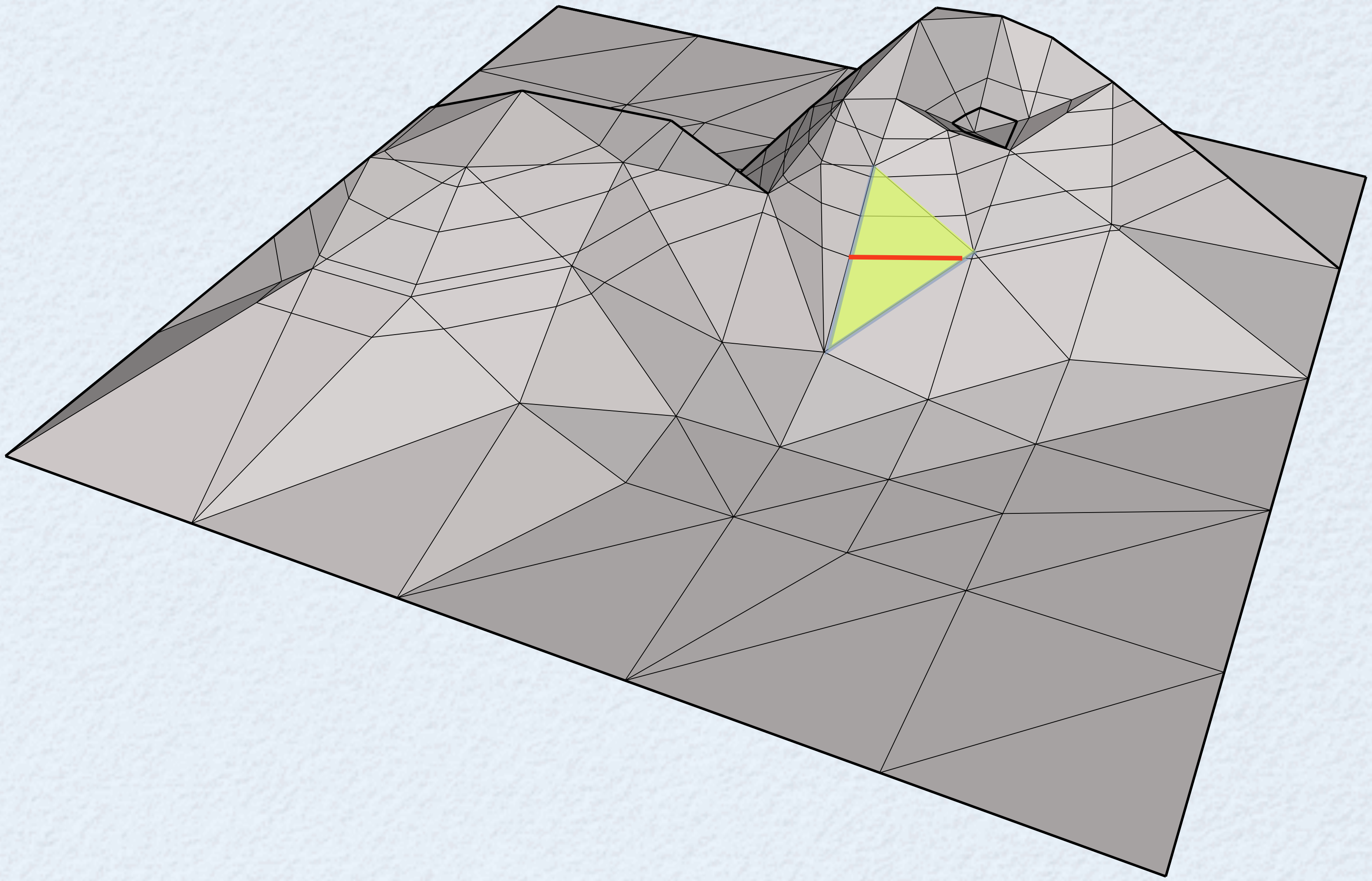
"Very naïve" Algorithm: Trace one contour at a time

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



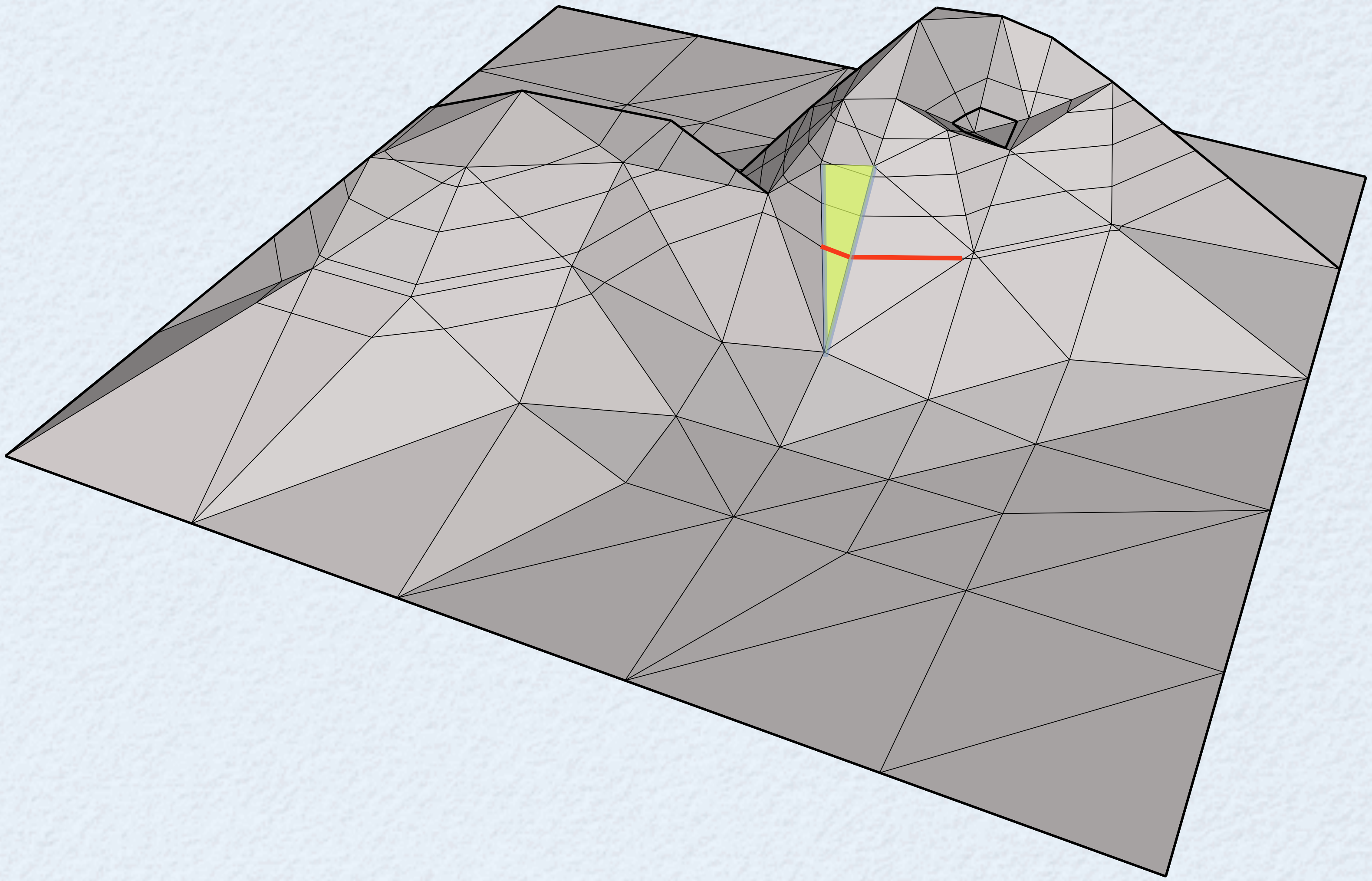
“Very naïve” Algorithm: Trace one contour at a time

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



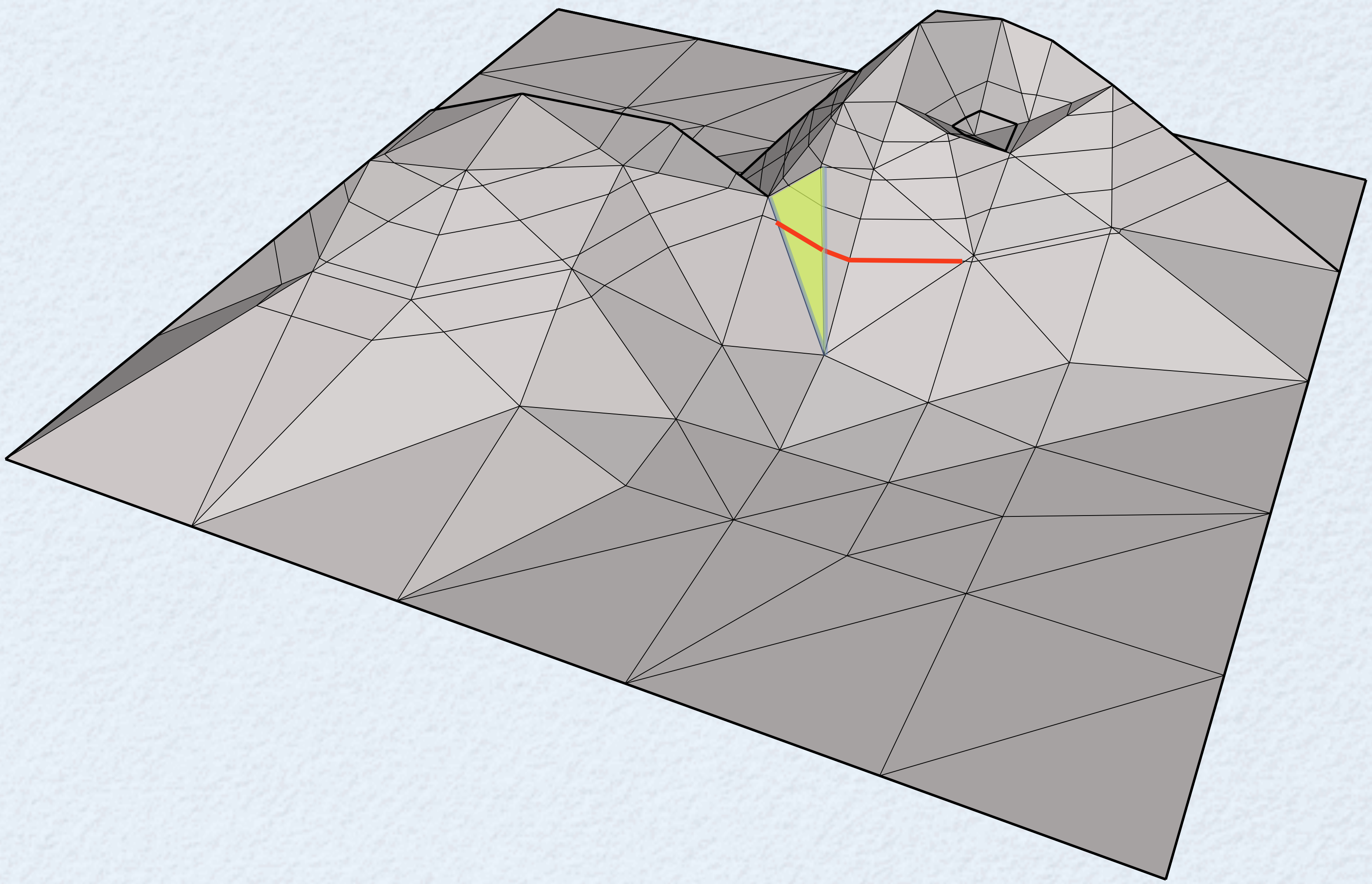
"Very naïve" Algorithm: Trace one contour at a time

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



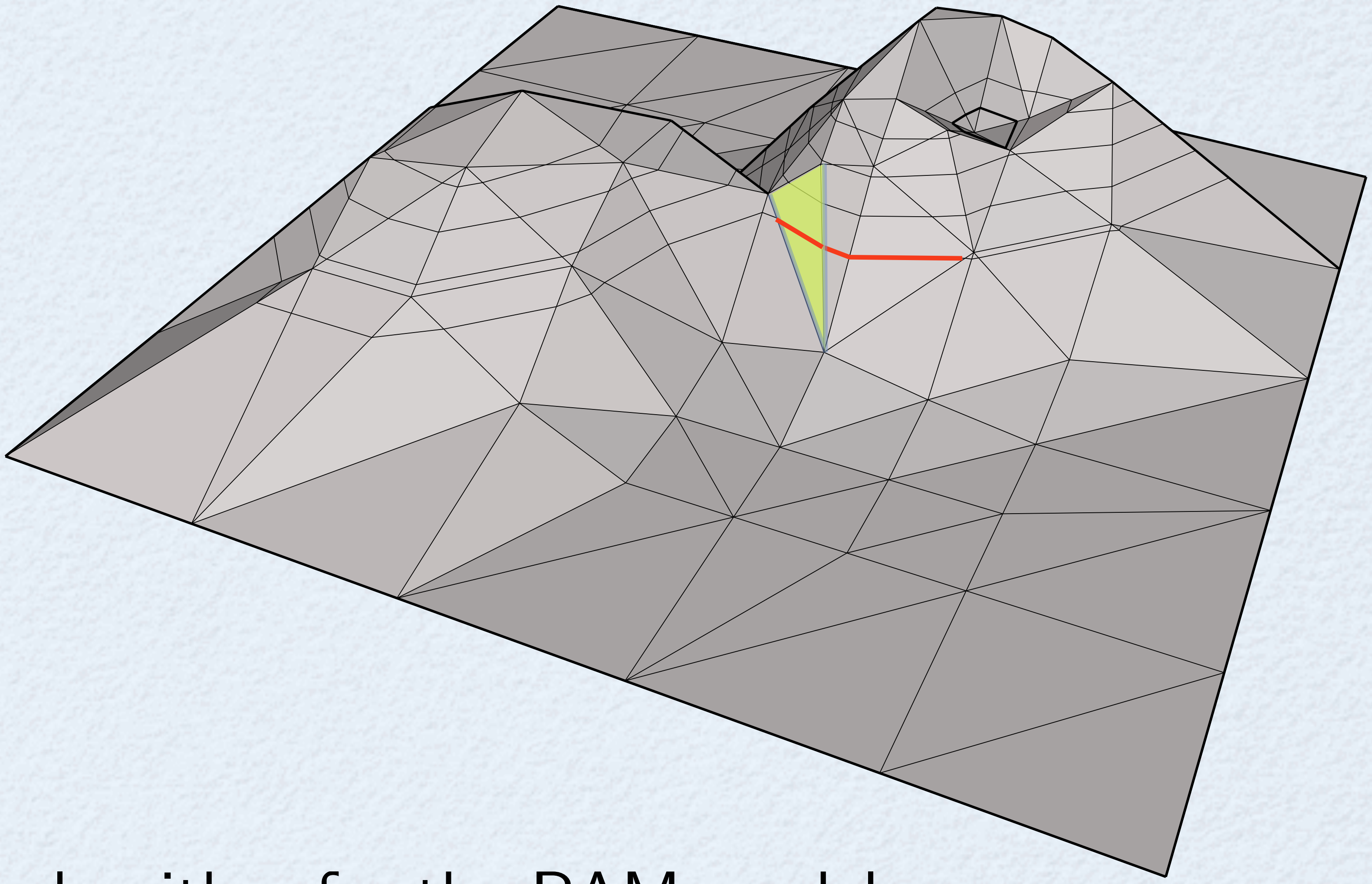
"Very naïve" Algorithm: Trace one contour at a time

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



"Very naïve" Algorithm: Trace one contour at a time

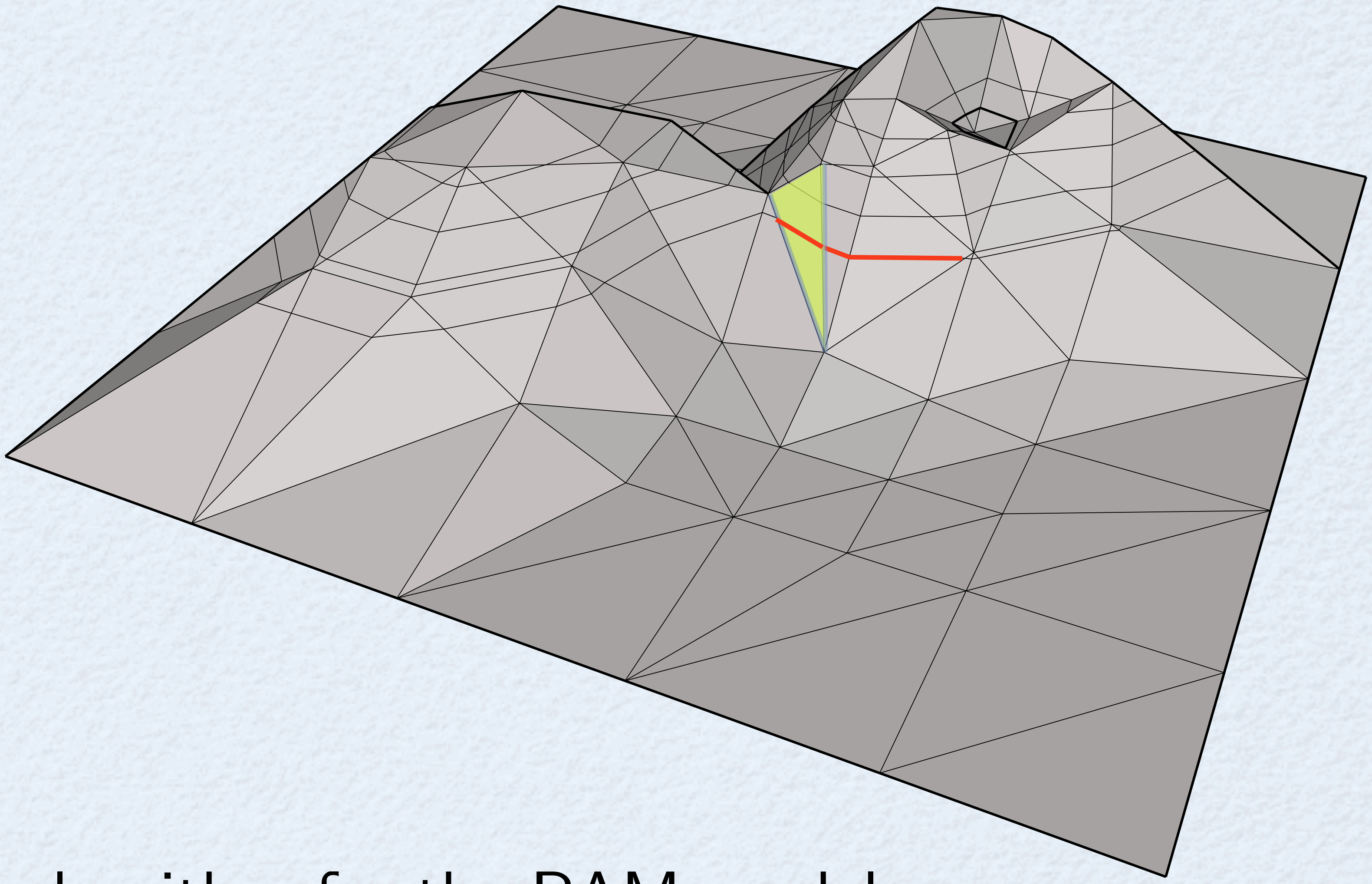
Find one "seed" triangle intersecting each contour and trace out the contour sequentially.



This is essentially the optimal algorithm for the RAM model.

"Very naïve" Algorithm: Trace 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.

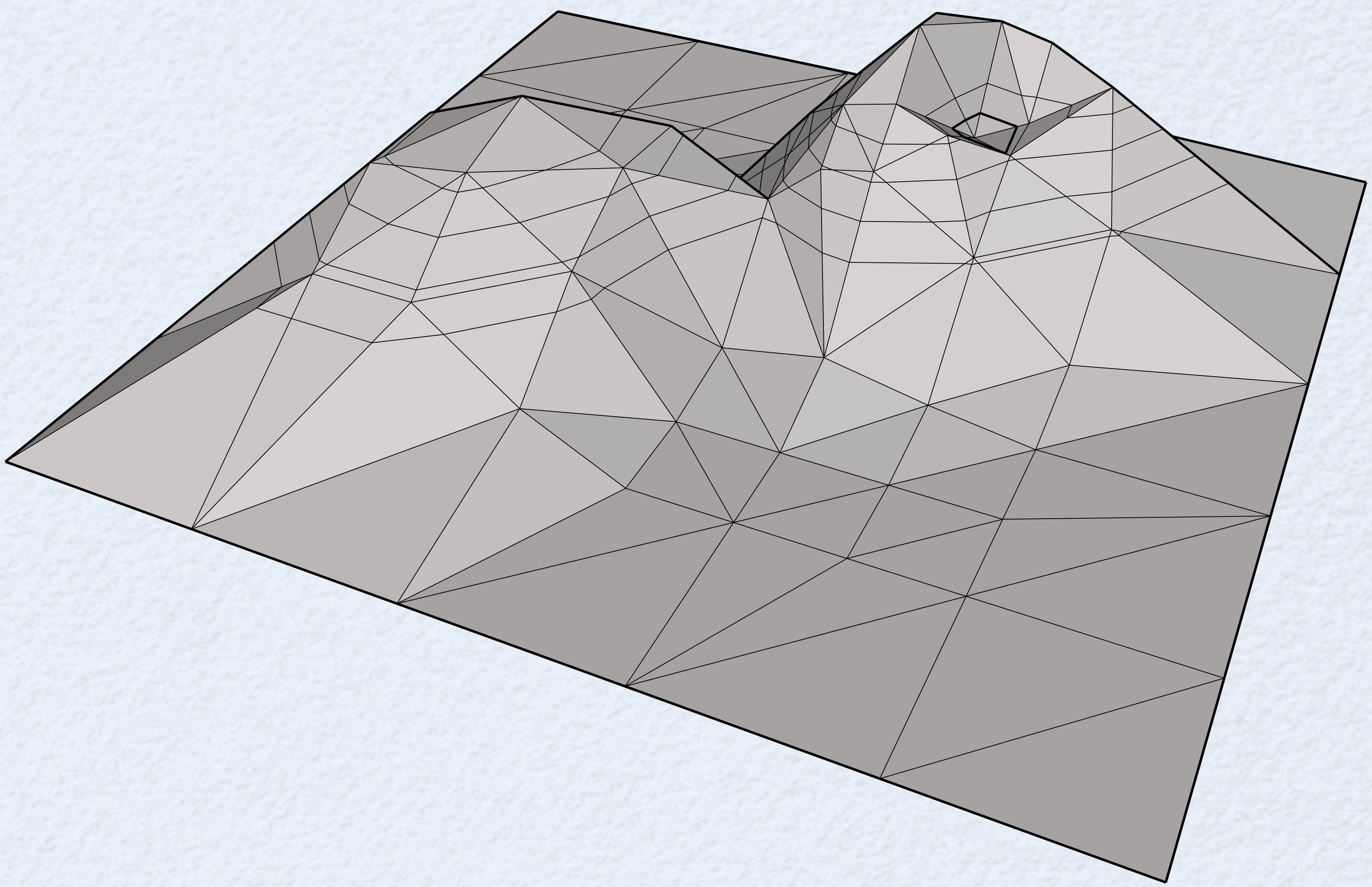
I/O Complexity:

$$O(N/B \cdot \log_B |L| + T).$$

segments in the output

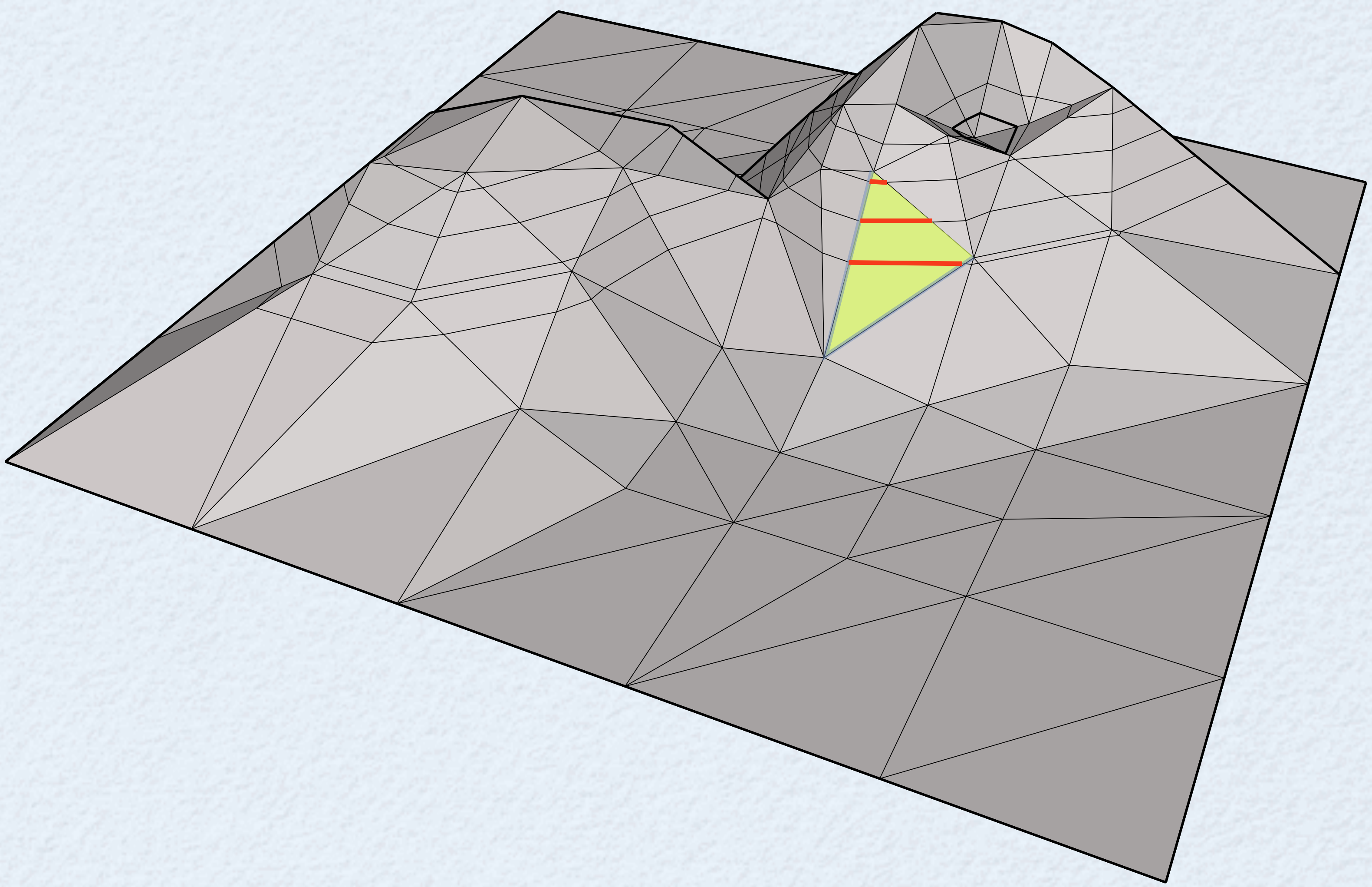
"Less naïve" Algorithm: Generate pieces, sort later

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



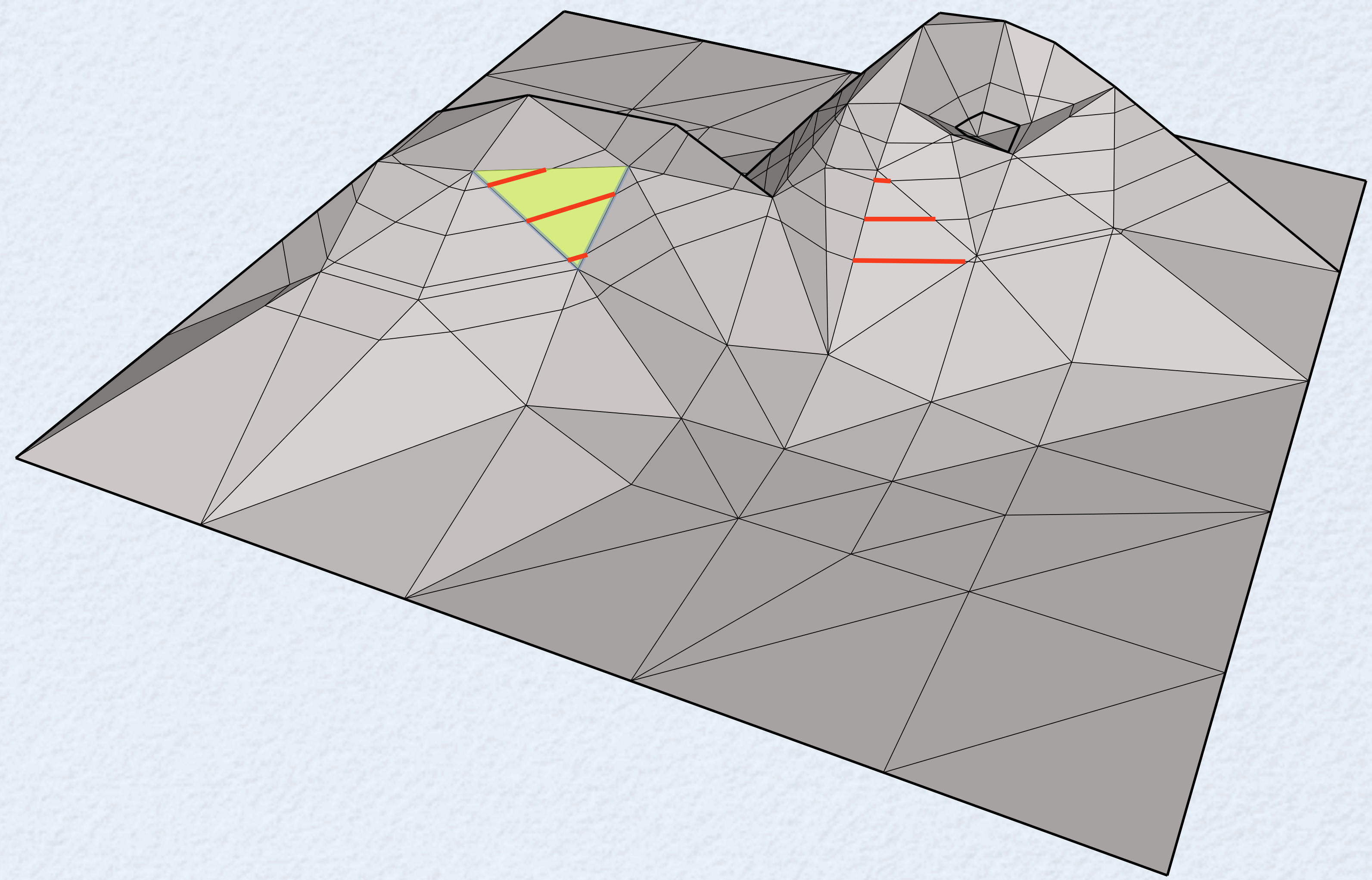
"Less naïve" Algorithm: Generate pieces, sort later

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



“Less naïve” Algorithm: Generate pieces, sort later

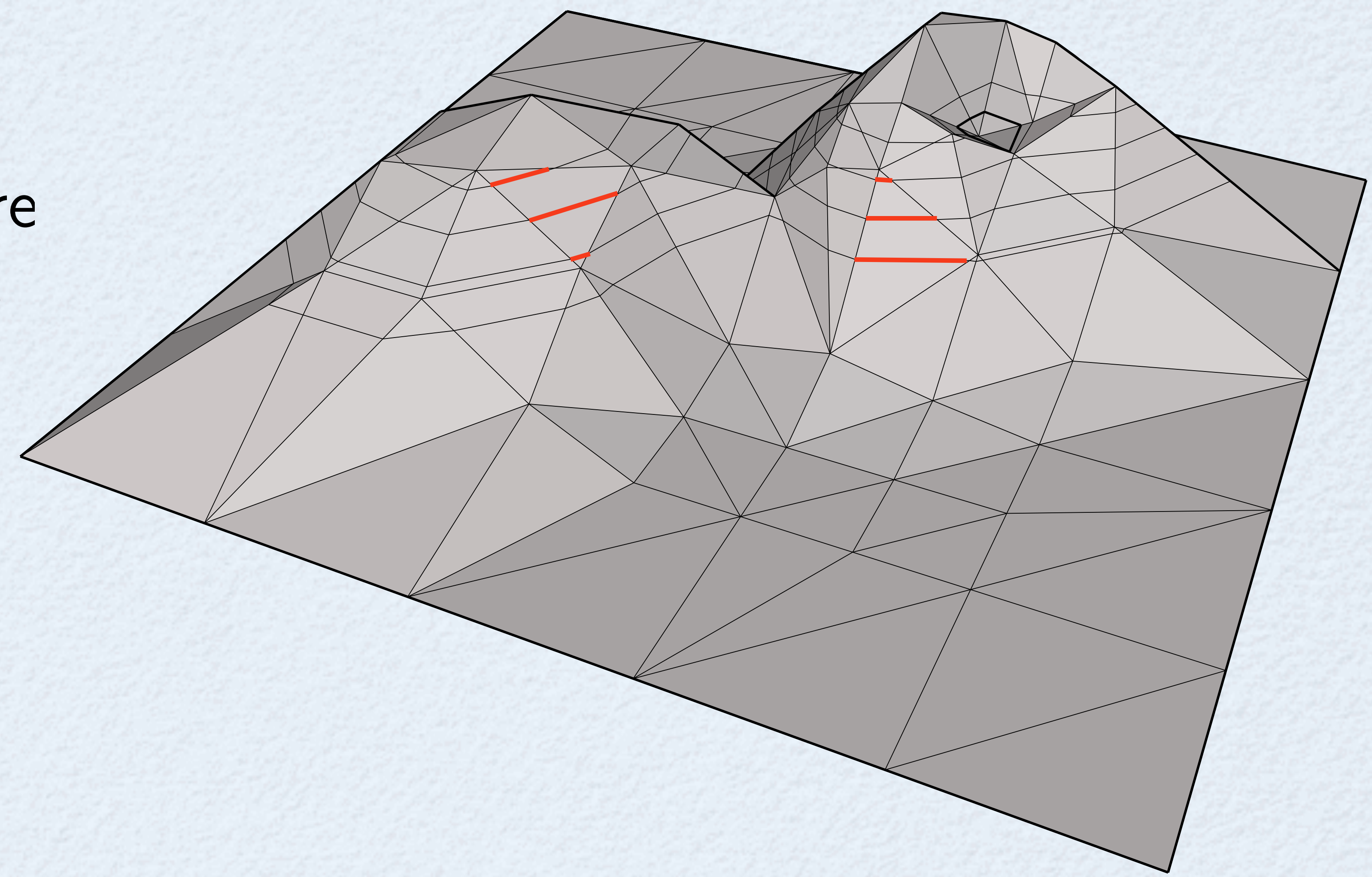
Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.



"Less naïve" Algorithm: Generate pieces, sort later

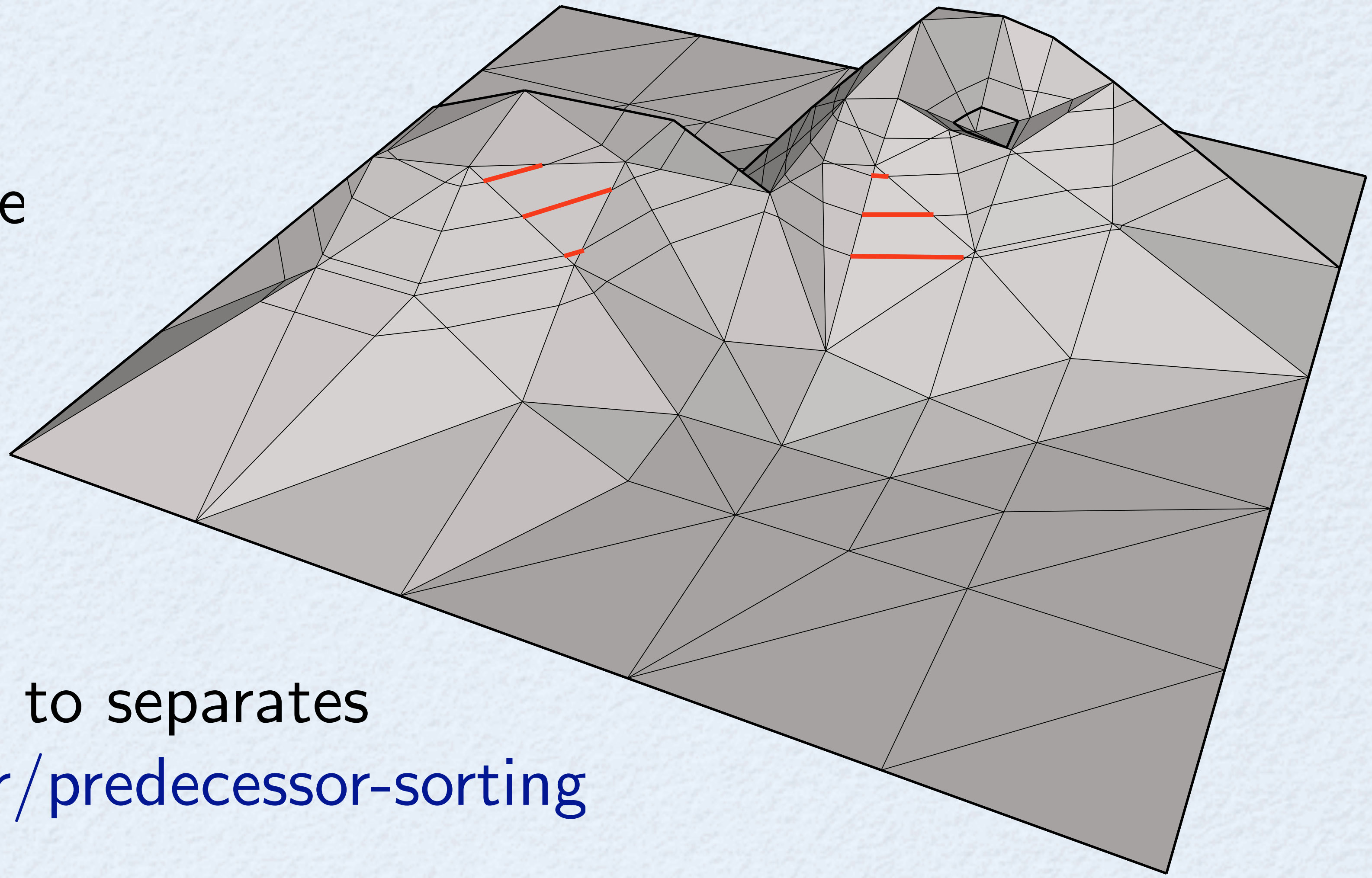
Scan the triangles (in the order laid out on the disk) and generate all segments. Then sort the output.

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



"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.

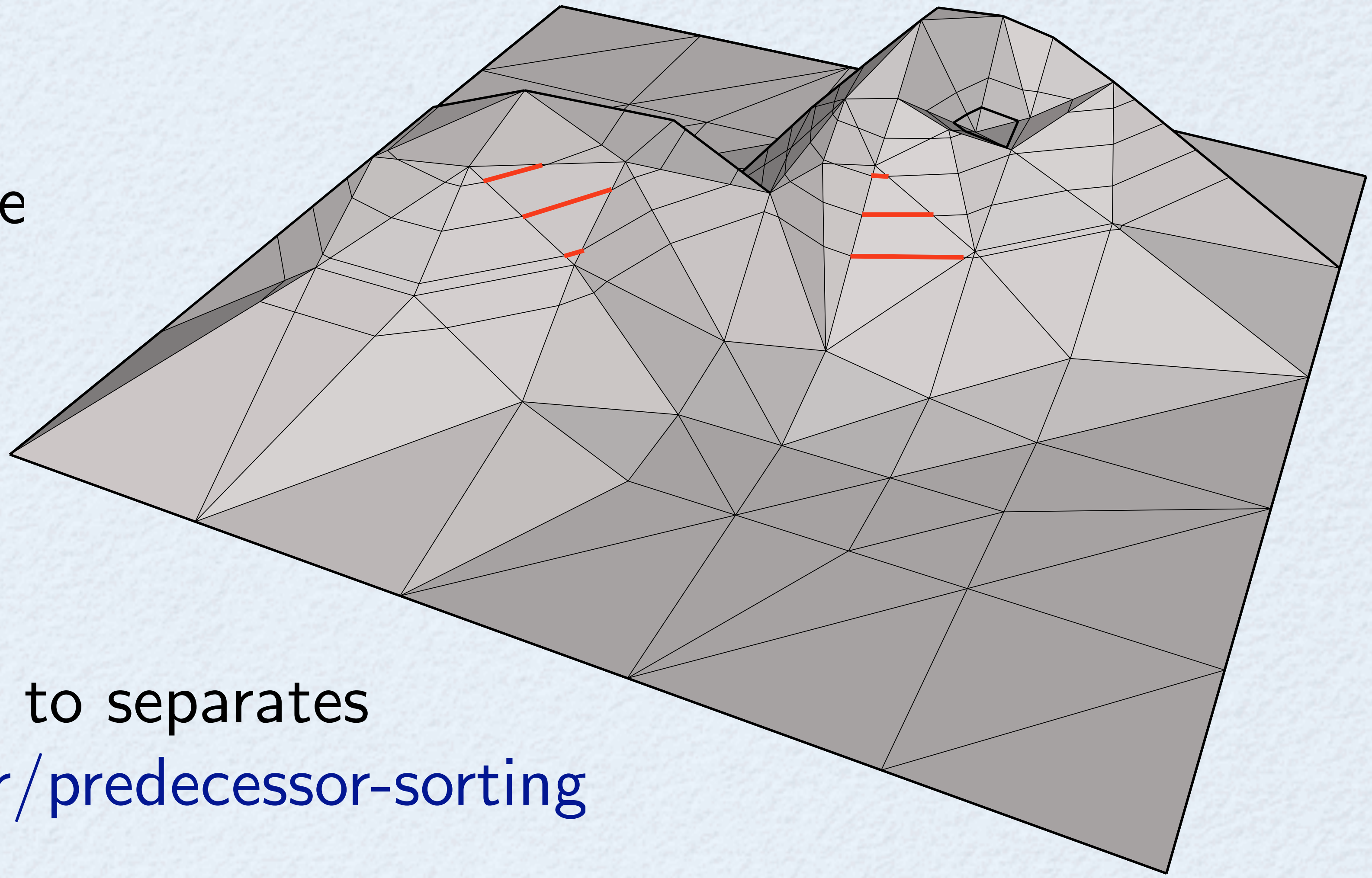


For segment s at level ℓ_i store pair (ℓ_i, s) plus the segments before and after s on contour containing s.

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

"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.

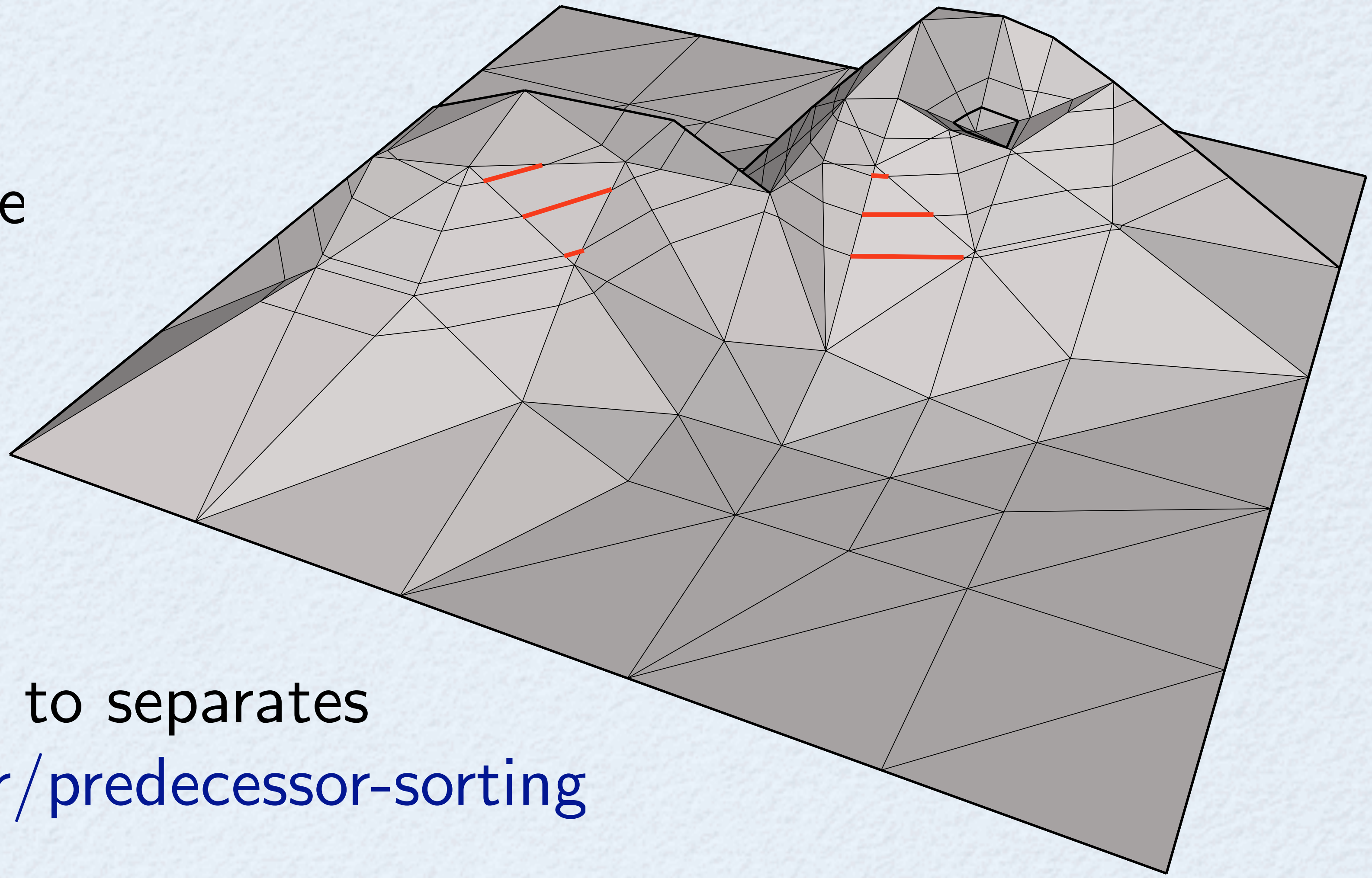
I/O Complexity:

$$O(N/B \cdot \log_B |L| + \mathsf{Sort}(T)).$$

"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 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.

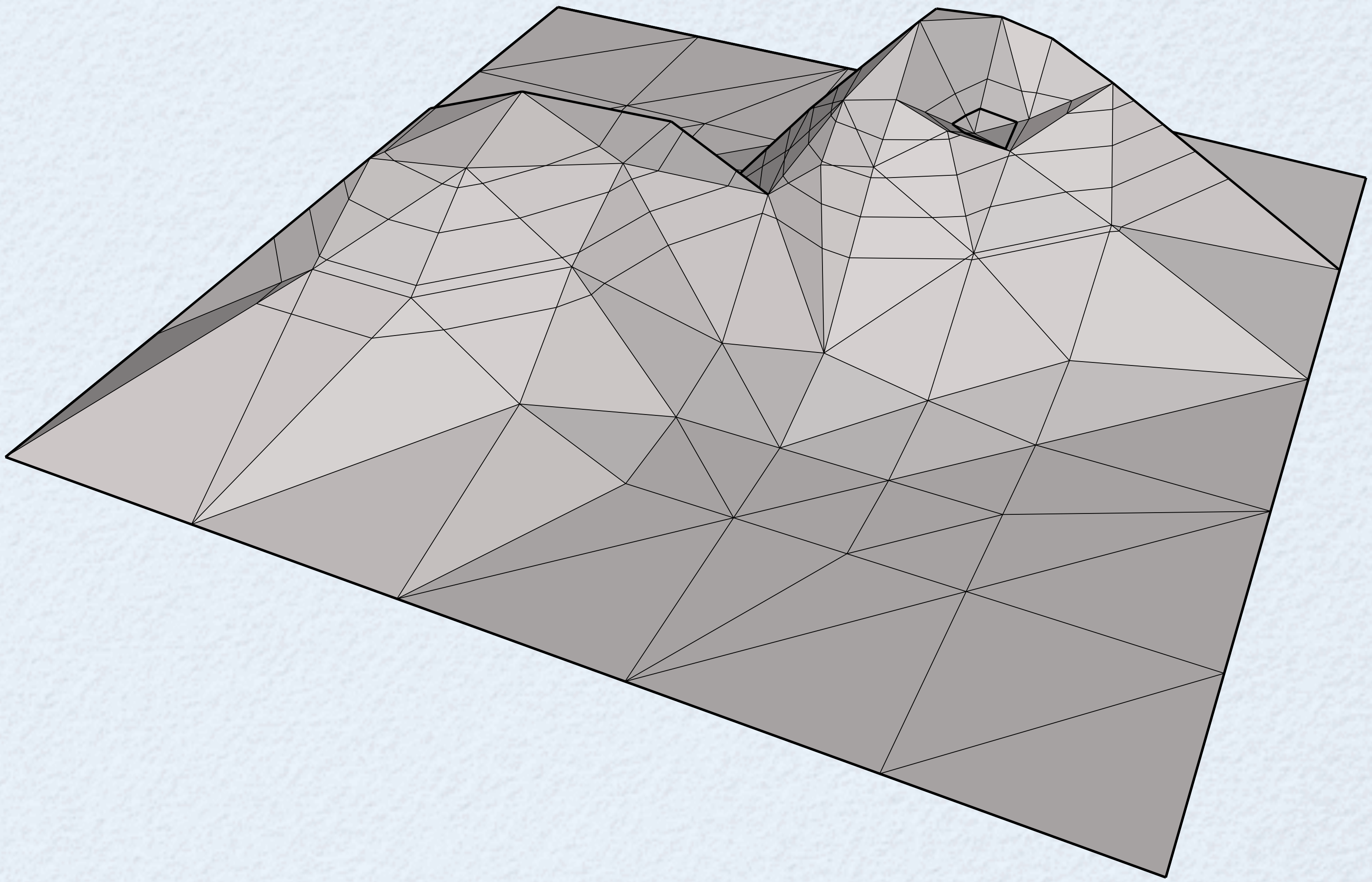
I/O Complexity:

$$O(N/B \cdot \log_B |L| + \mathsf{Sort}(T)).$$

This talk: $O(\mathsf{Sort}(N) + T/B)$.

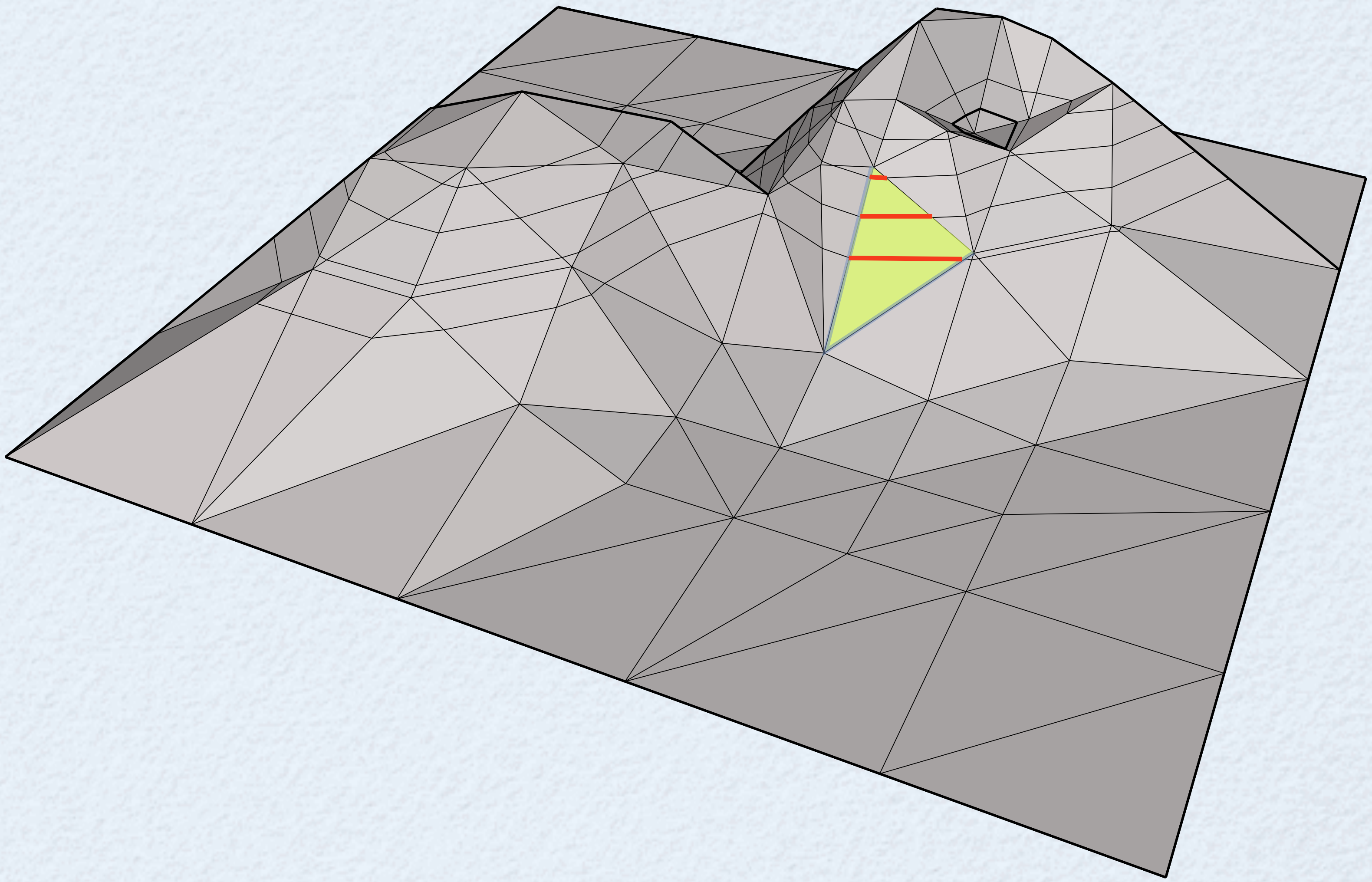
Idea: Grow Contours Contiguously

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.



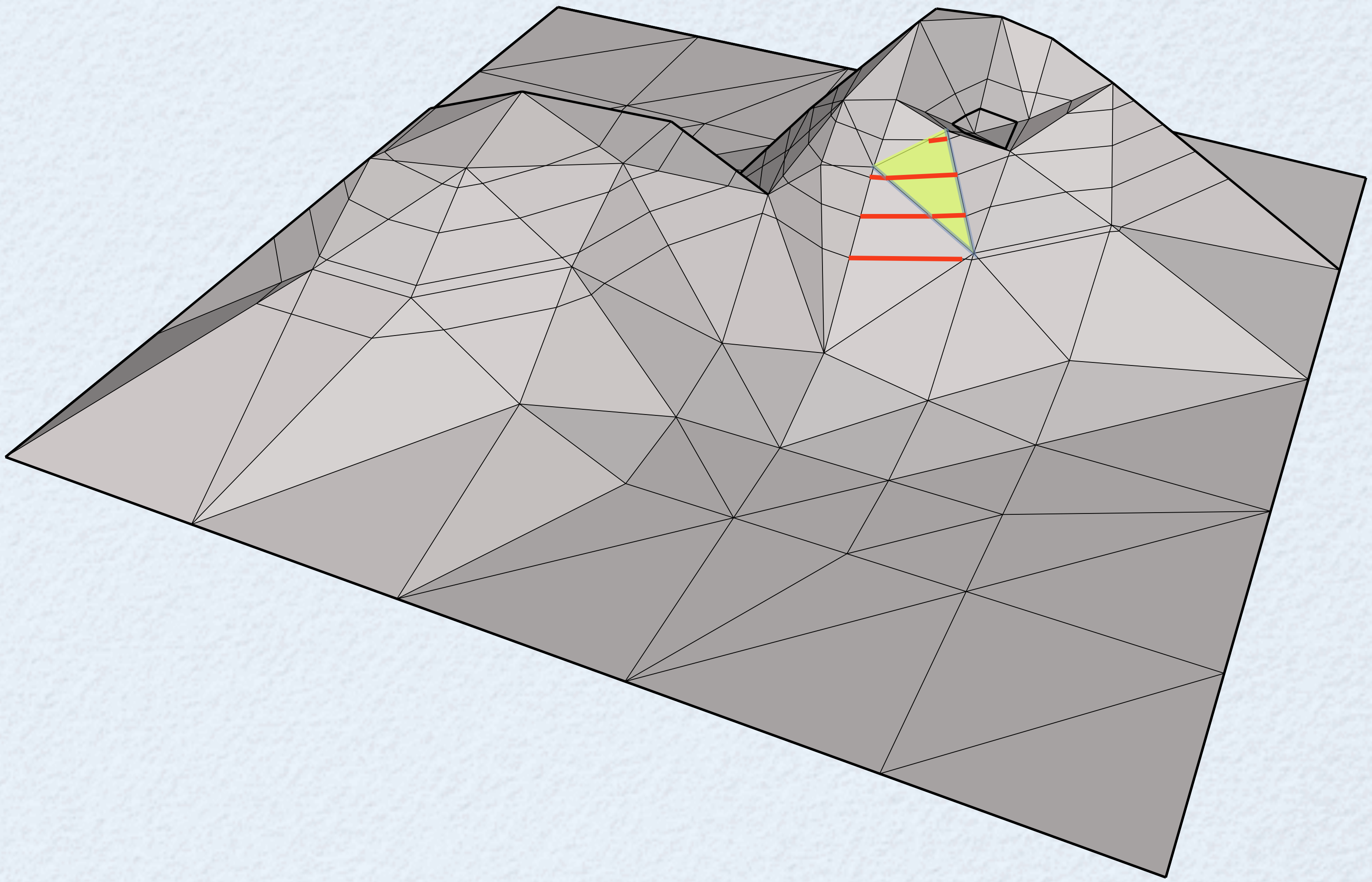
Idea: Grow Contours Contiguously

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.



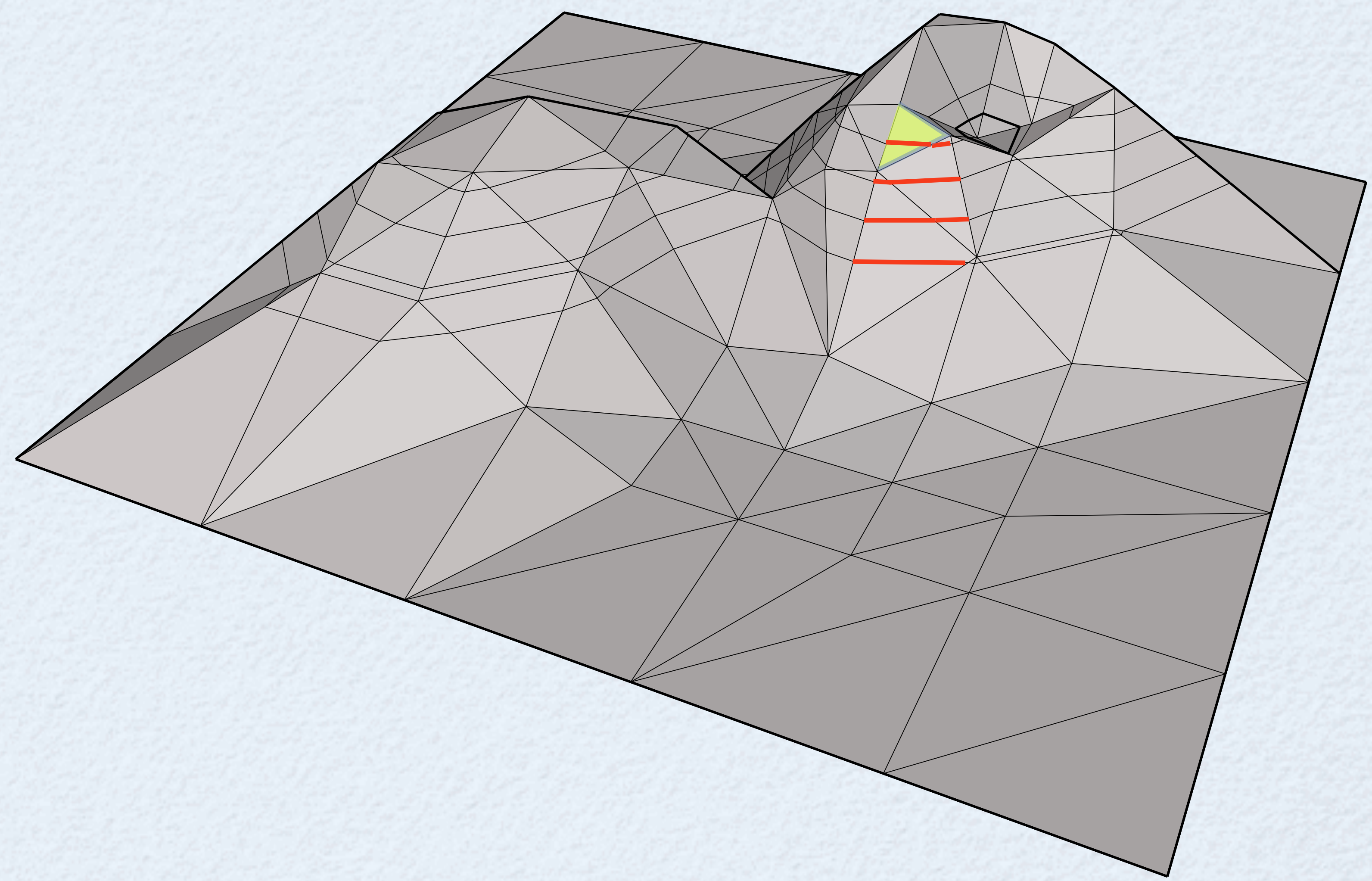
Idea: Grow Contours Contiguously

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.



Idea: Grow Contours Contiguously

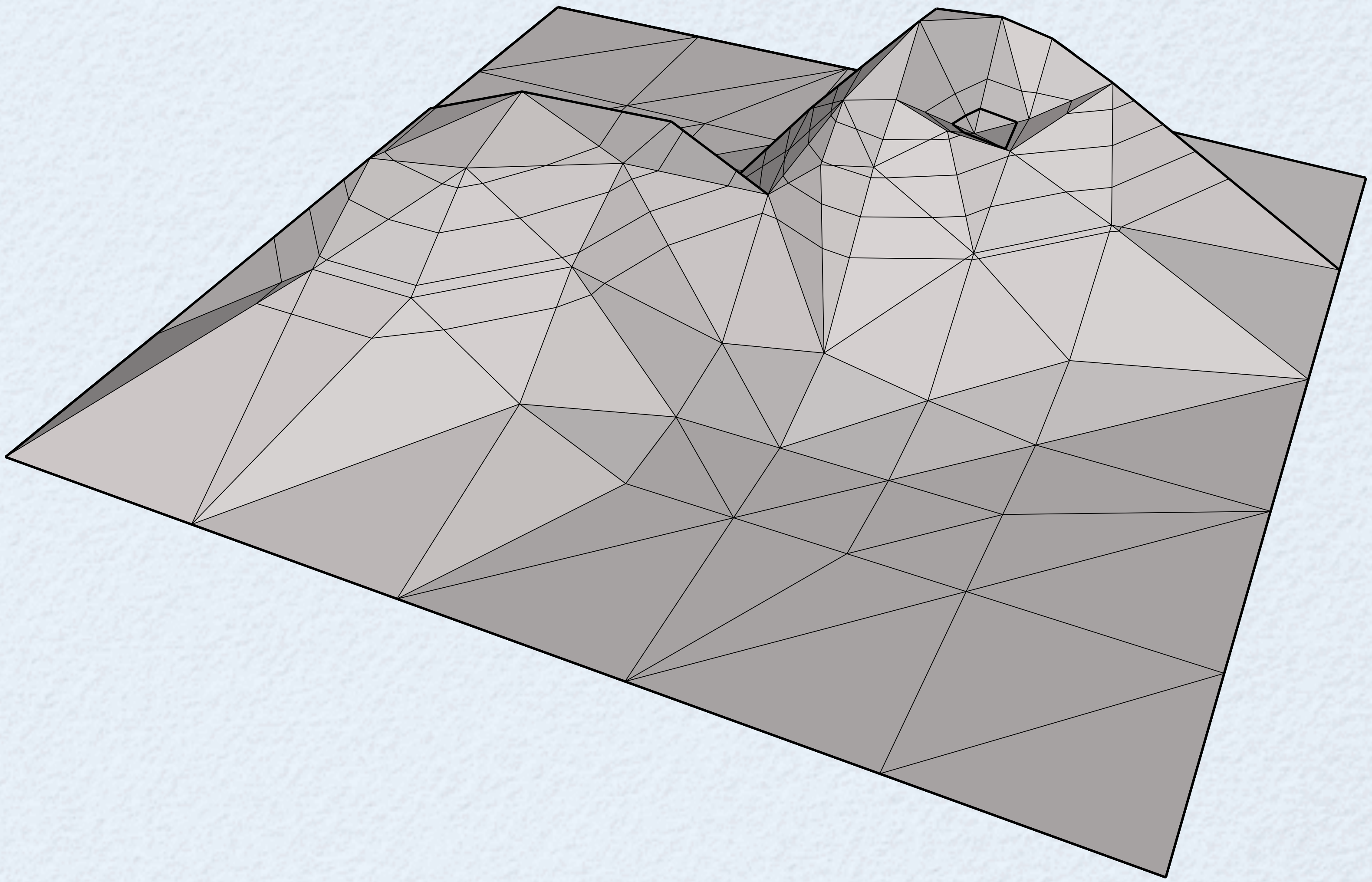
If triangles were ordered on disk such that all partially generated contours in “less naïve” algorithm stayed connected, no succ/pred sorting would be needed.



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.

$\prec$: such an ordering

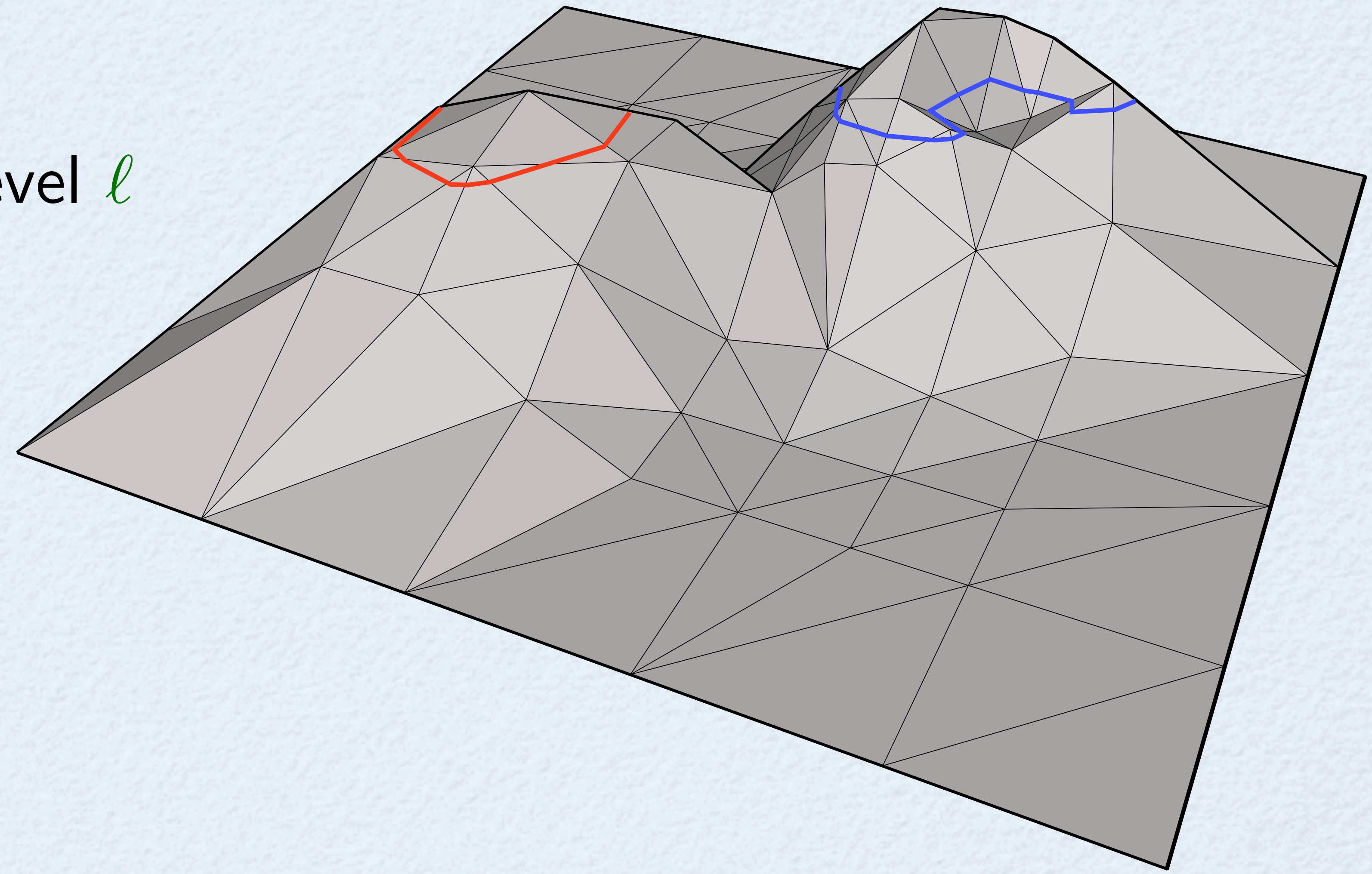


Idea: Grow Contours Contiguously

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.

$\prec$: such an ordering

Δ_ℓ: triangles that intersect level ℓ

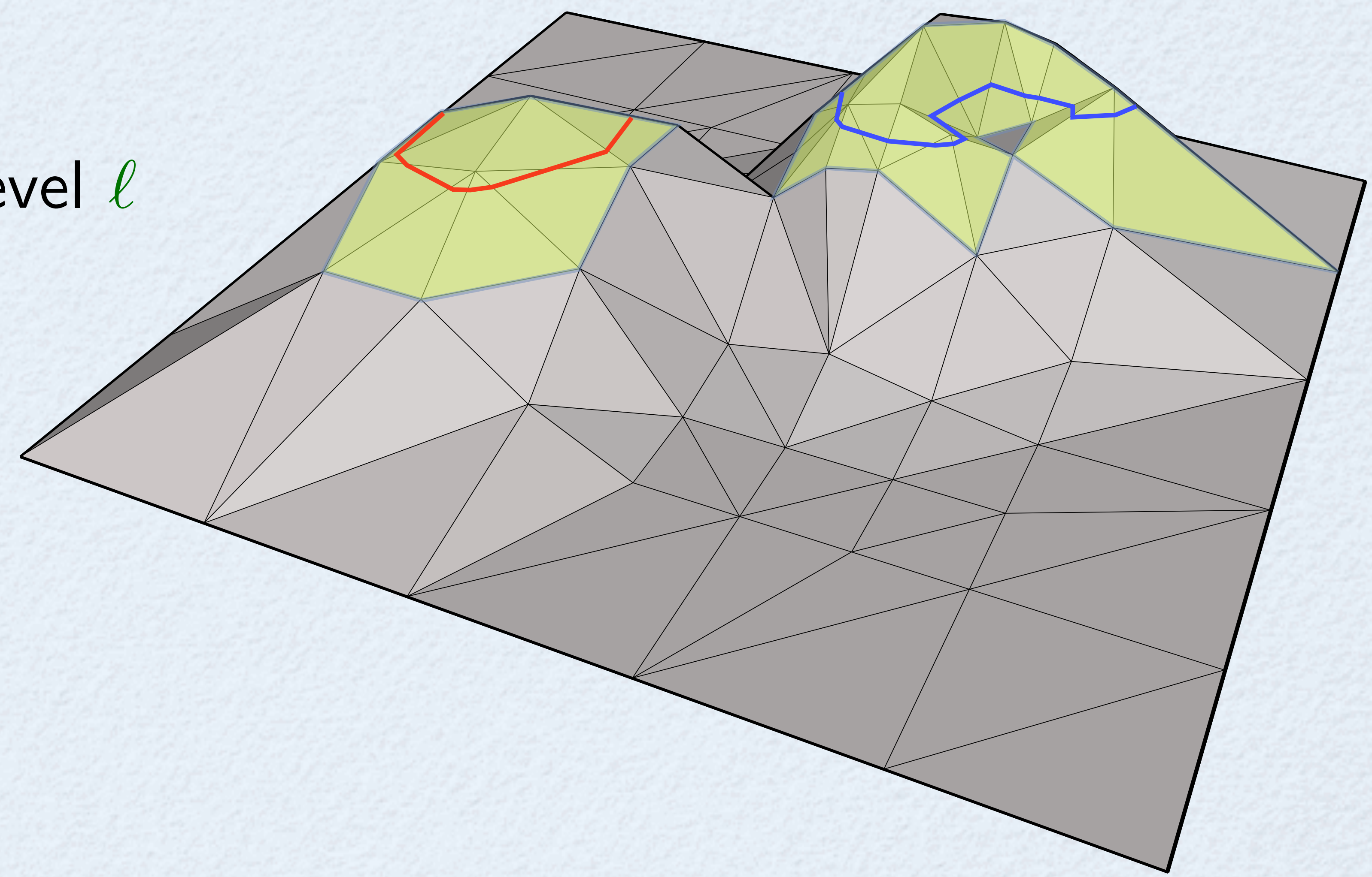


Idea: Grow Contours Contiguously

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.

$\prec$: such an ordering

Δ_ℓ: triangles that intersect level ℓ

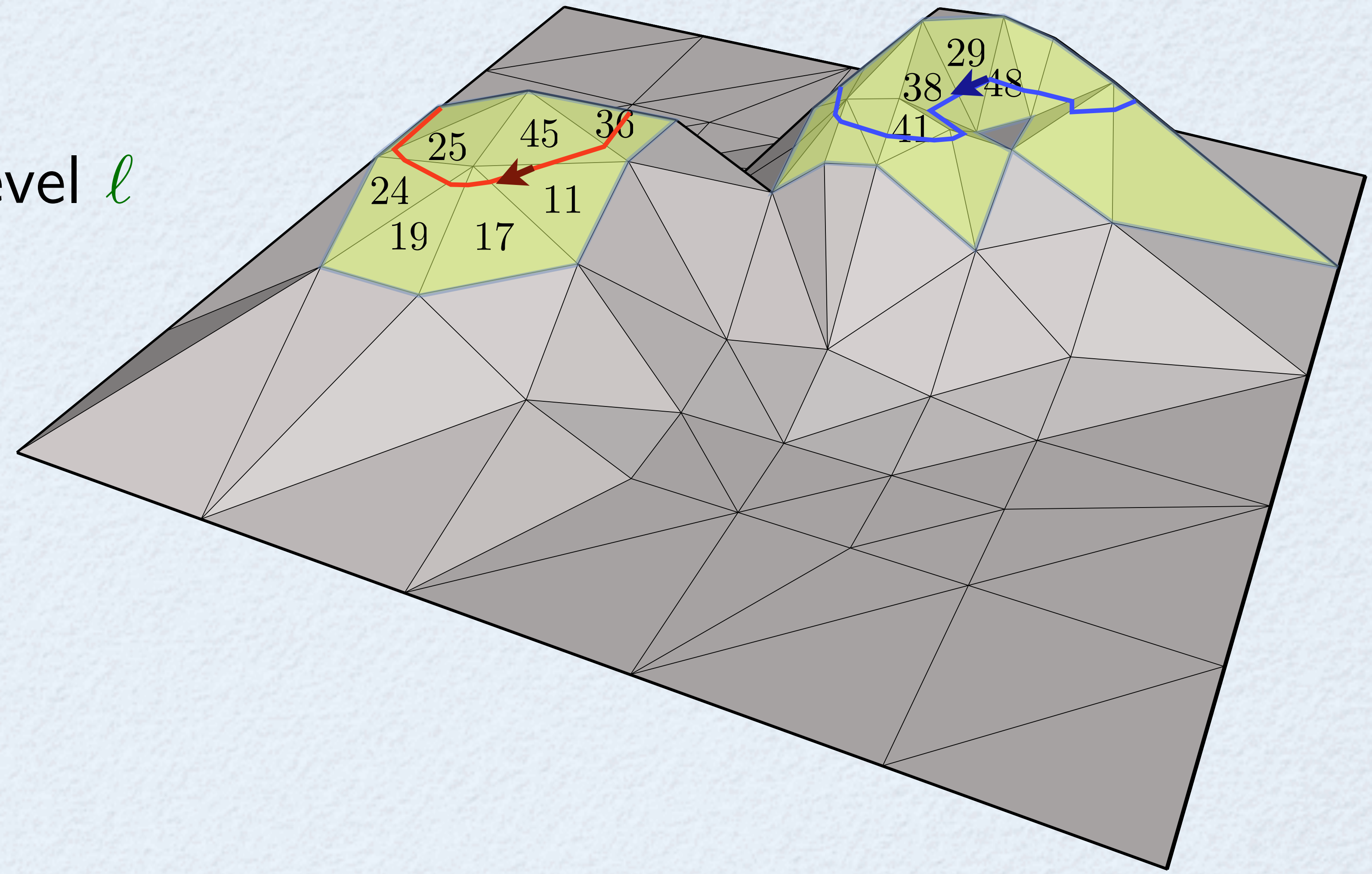


Idea: Grow Contours Contiguously

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.

$\prec$: such an ordering

Δ_ℓ: triangles that intersect level ℓ

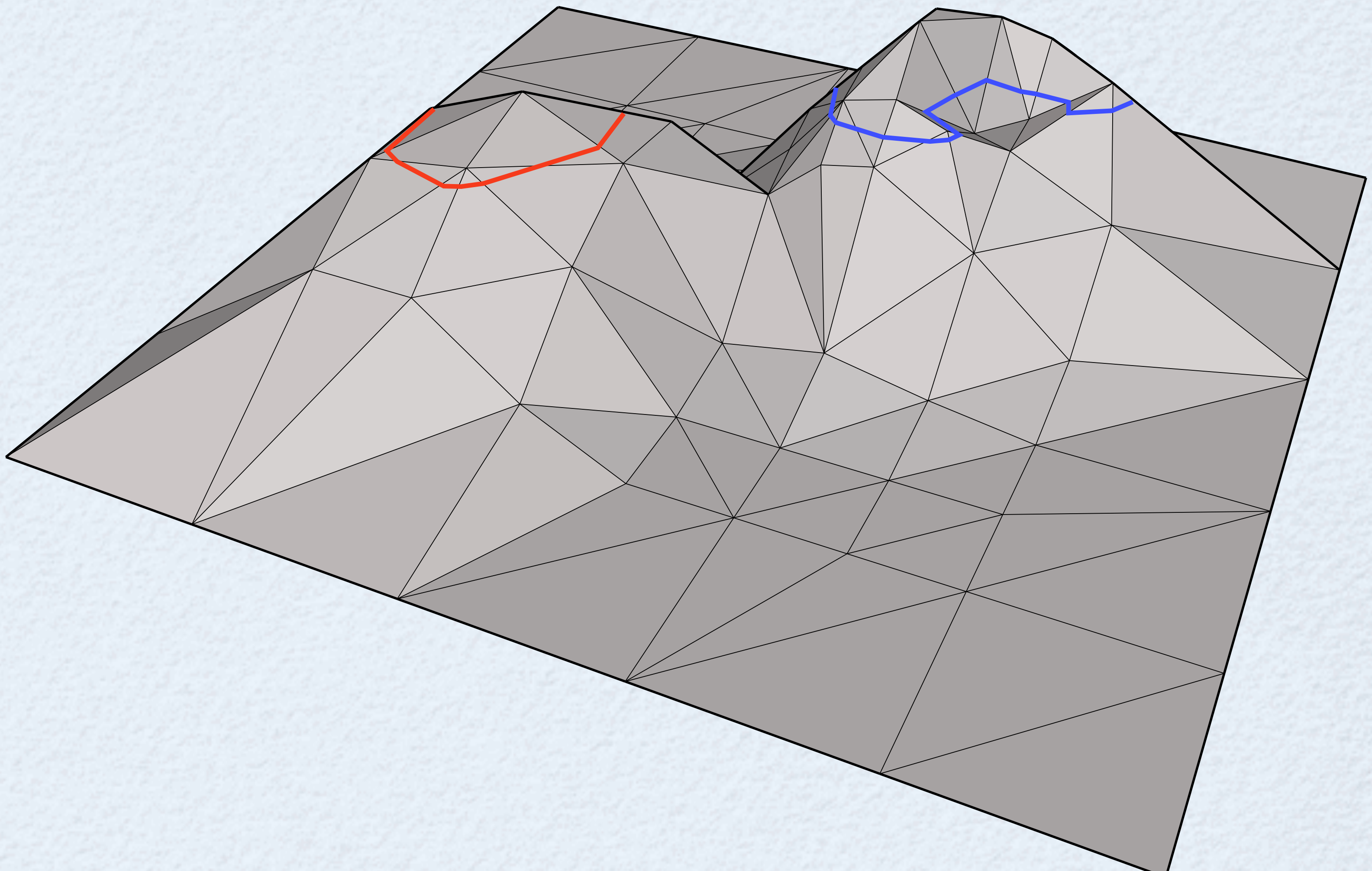


The restriction of $\prec$ to Δ_ℓ traverses each contour of $\mathbb{M}$ in circular order.

Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is
a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

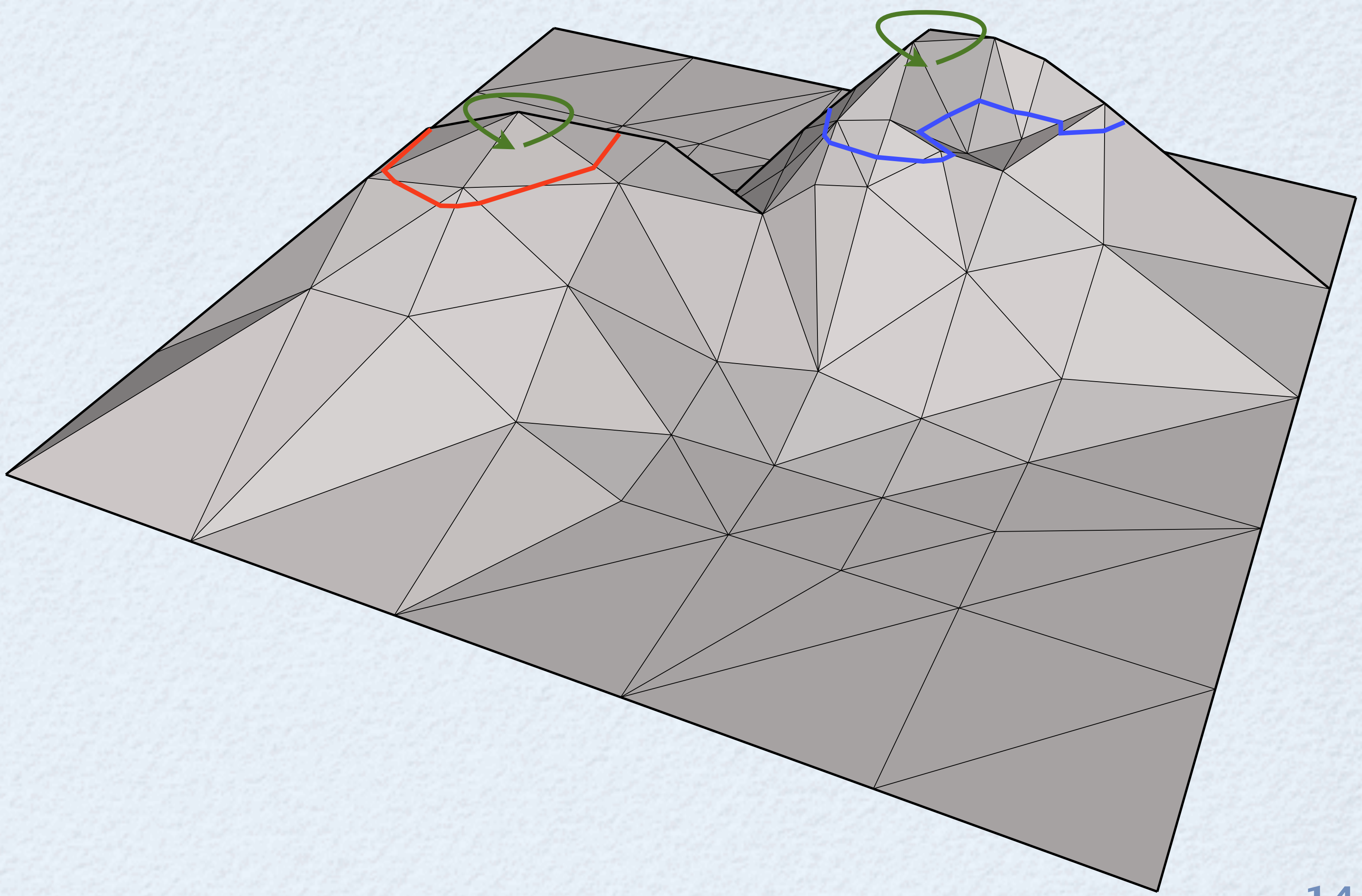
1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.



Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.

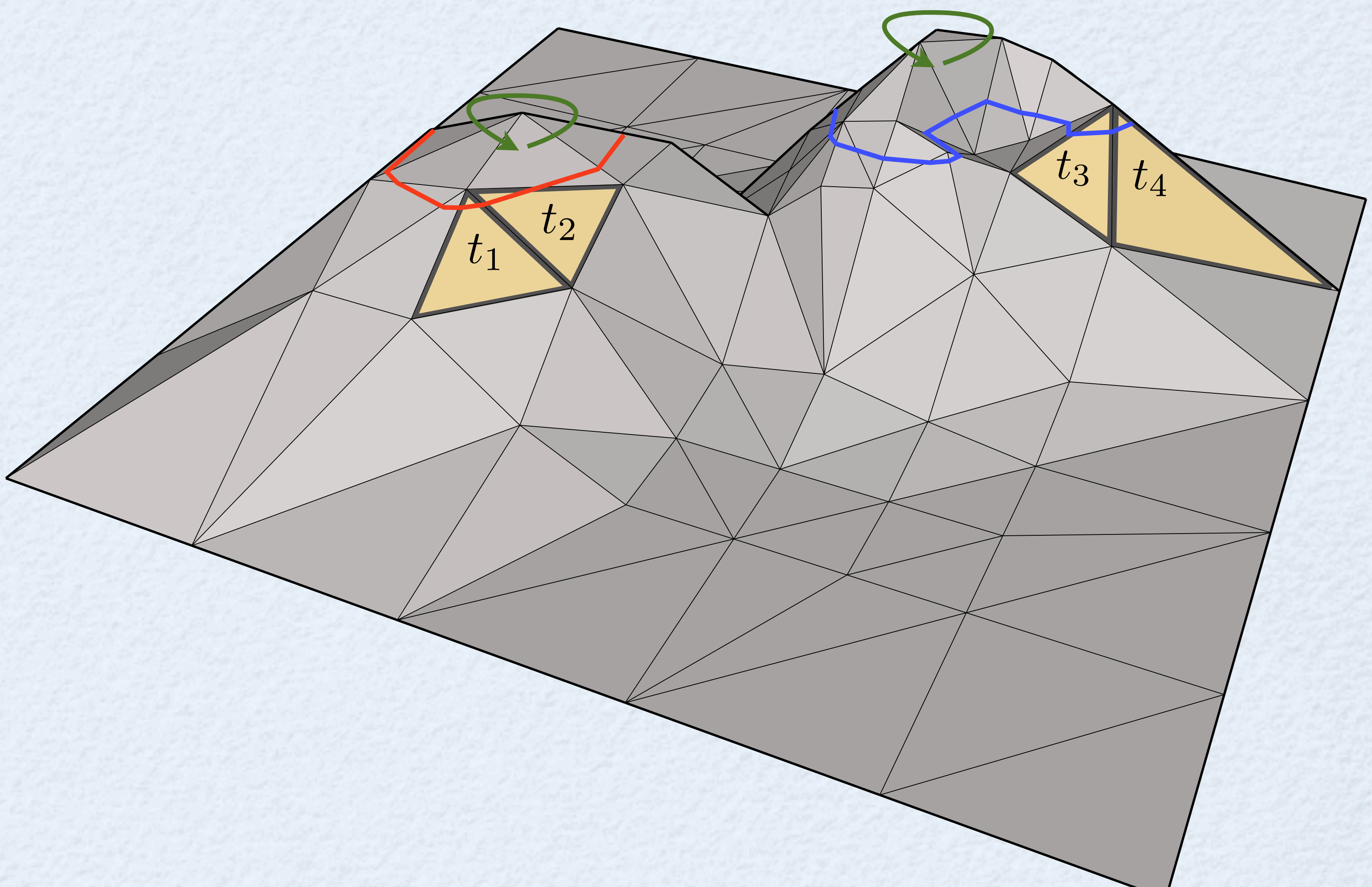


Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \quad \Longrightarrow \quad t_1 \prec t_4 \prec t_2$$

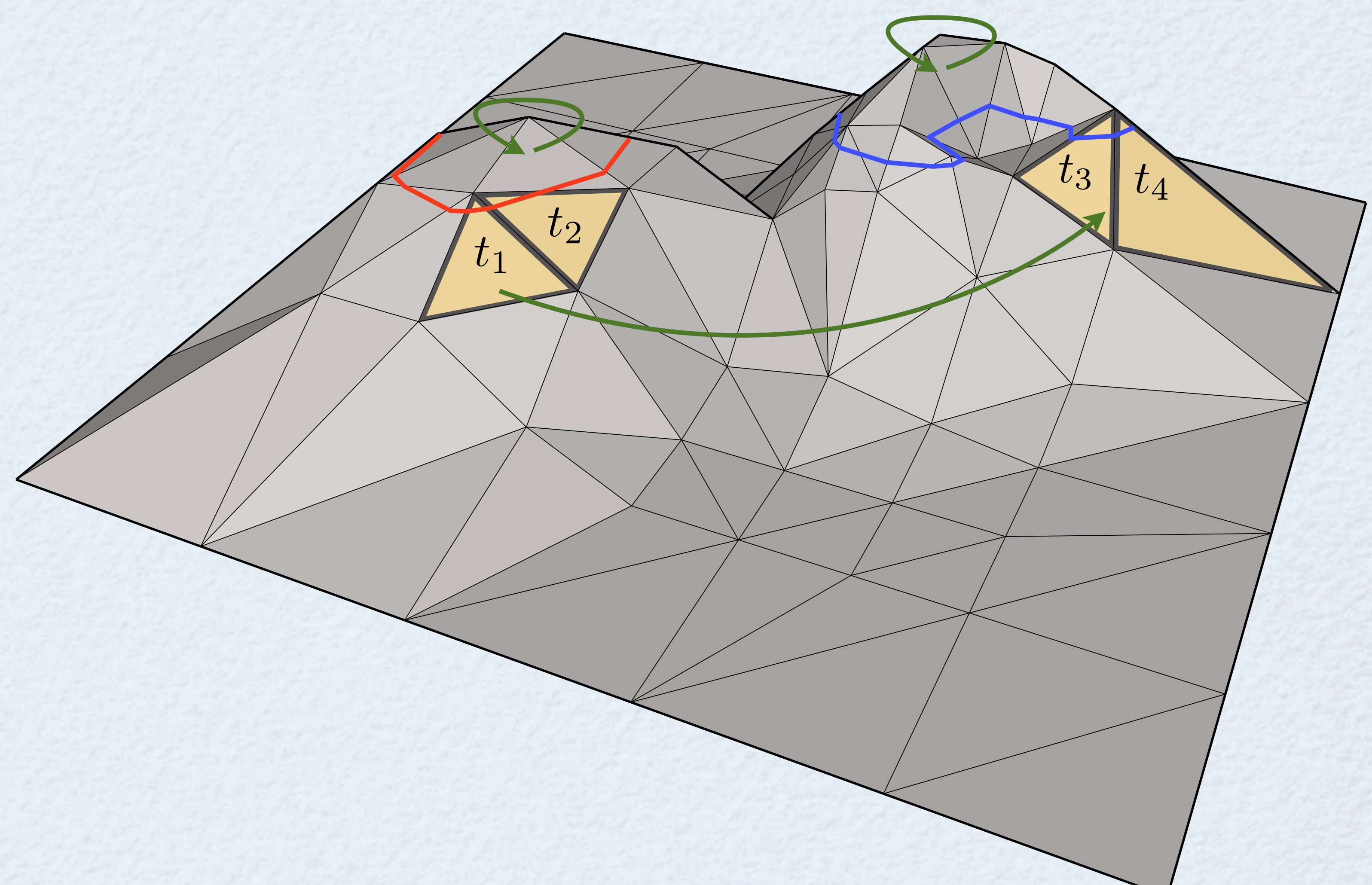


Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \implies t_1 \prec t_4 \prec t_2$$

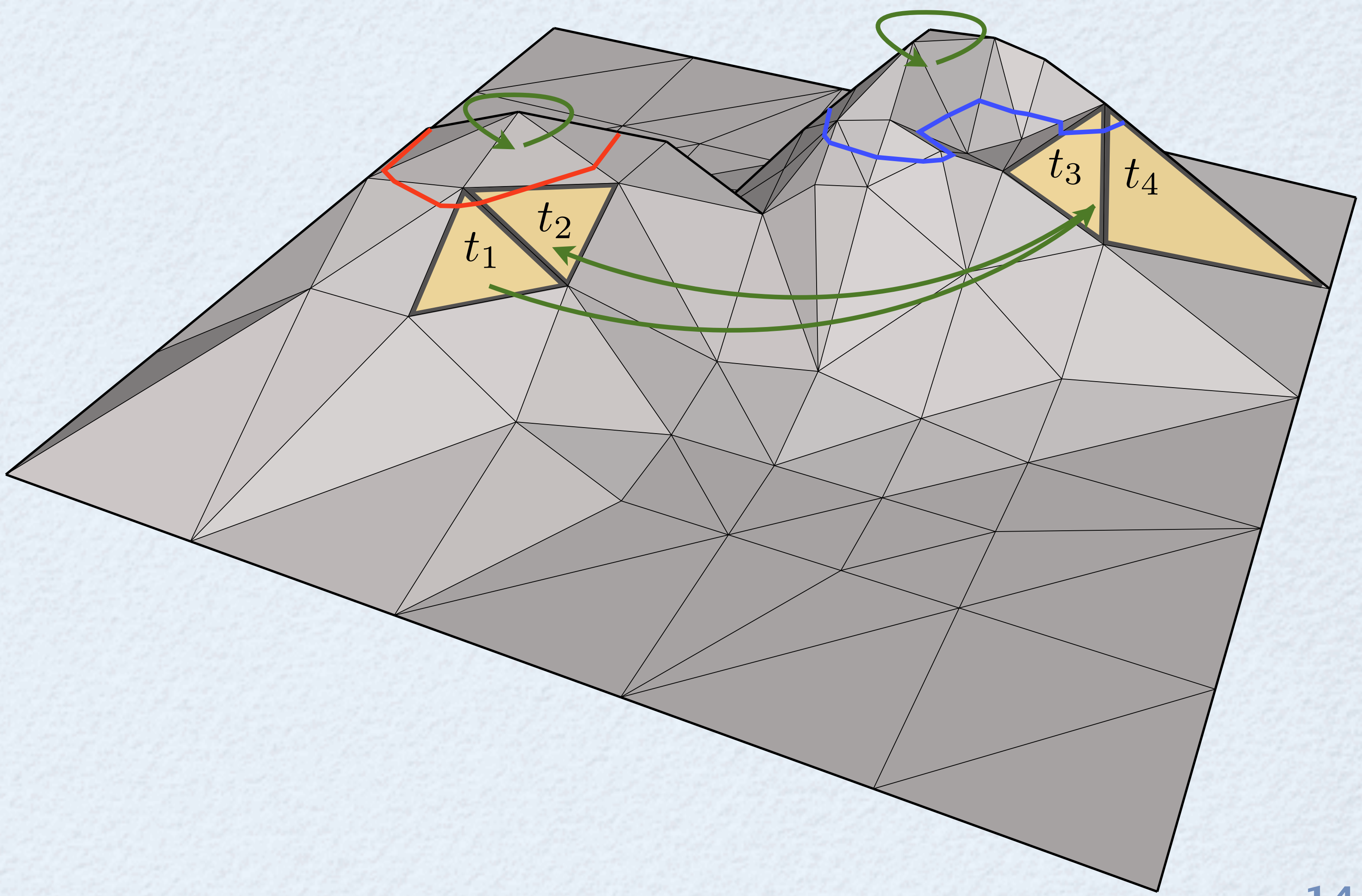


Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \quad \Longrightarrow \quad t_1 \prec t_4 \prec t_2$$

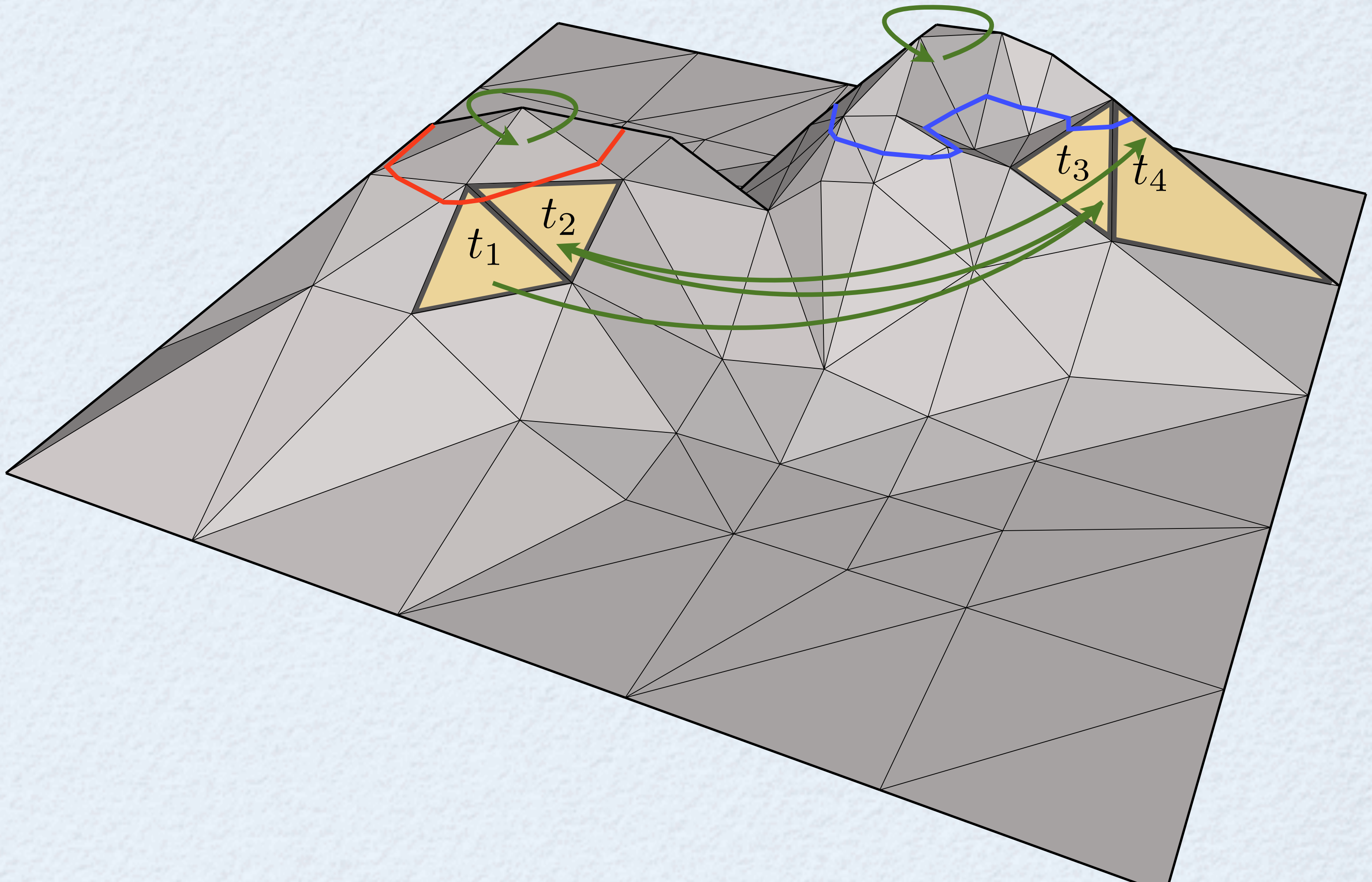


Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \quad \Longrightarrow \quad t_1 \prec t_4 \prec t_2$$

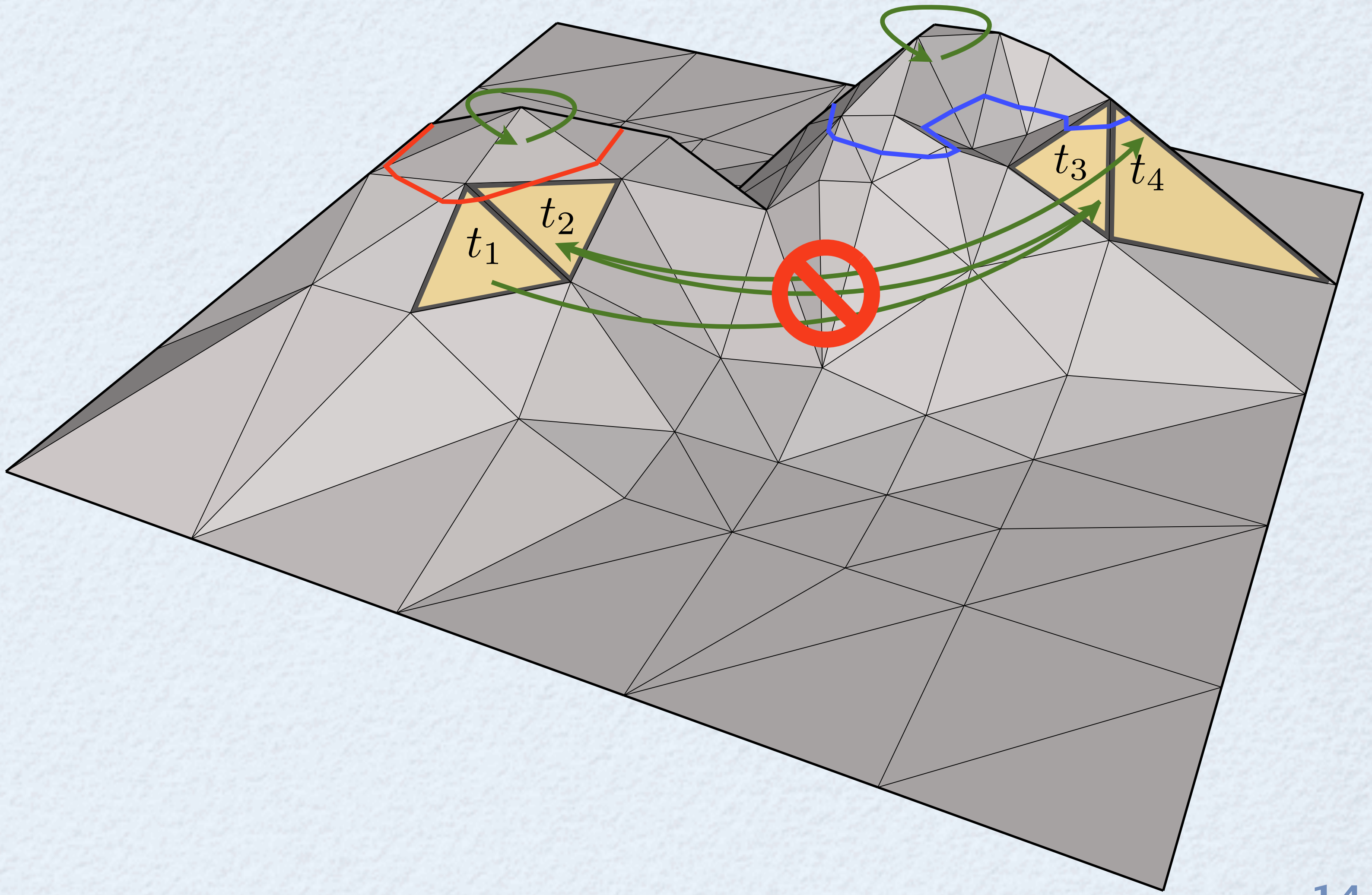


Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \implies t_1 \prec t_4 \prec t_2$$



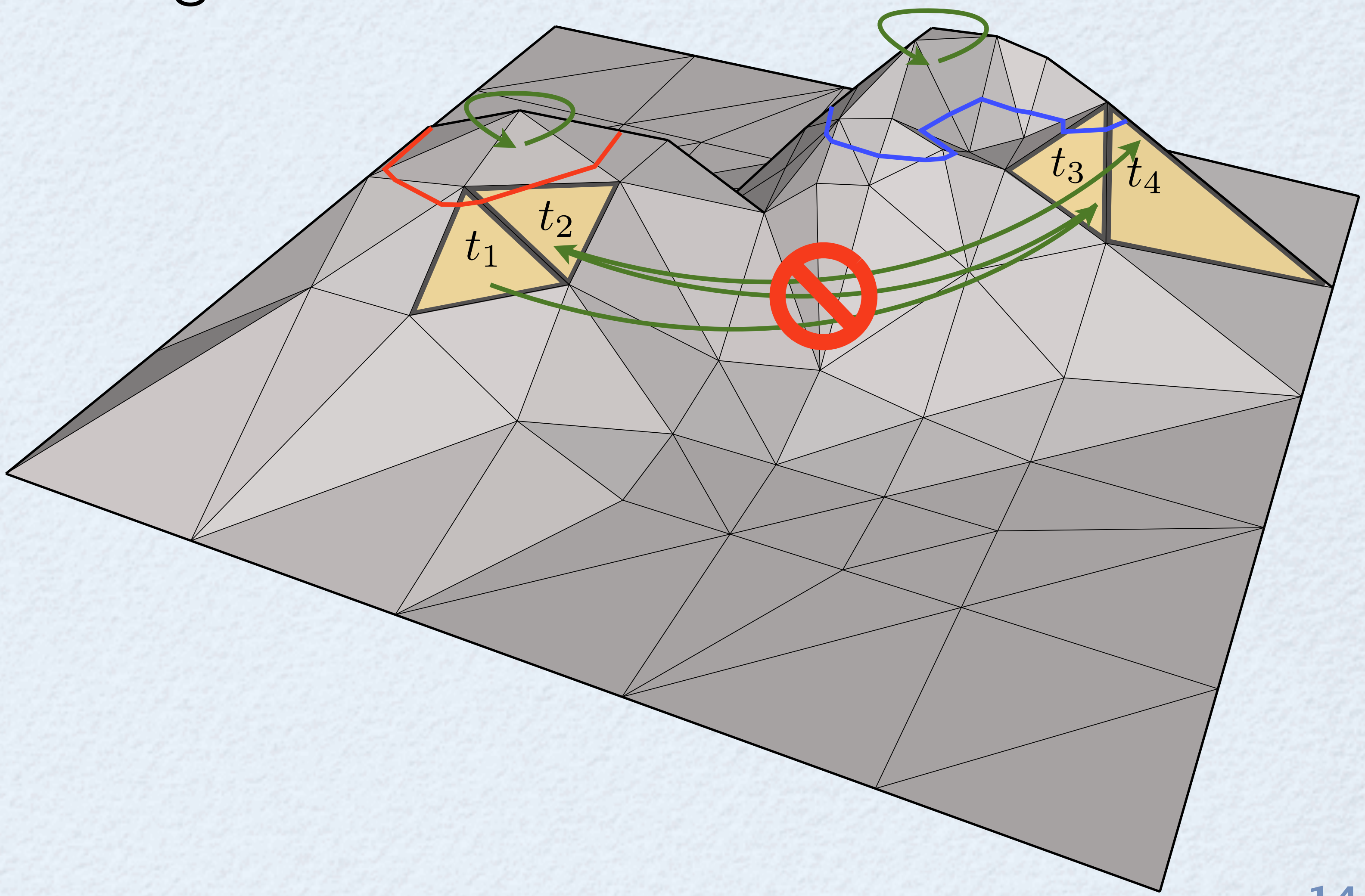
Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \implies t_1 \prec t_4 \prec t_2$$

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



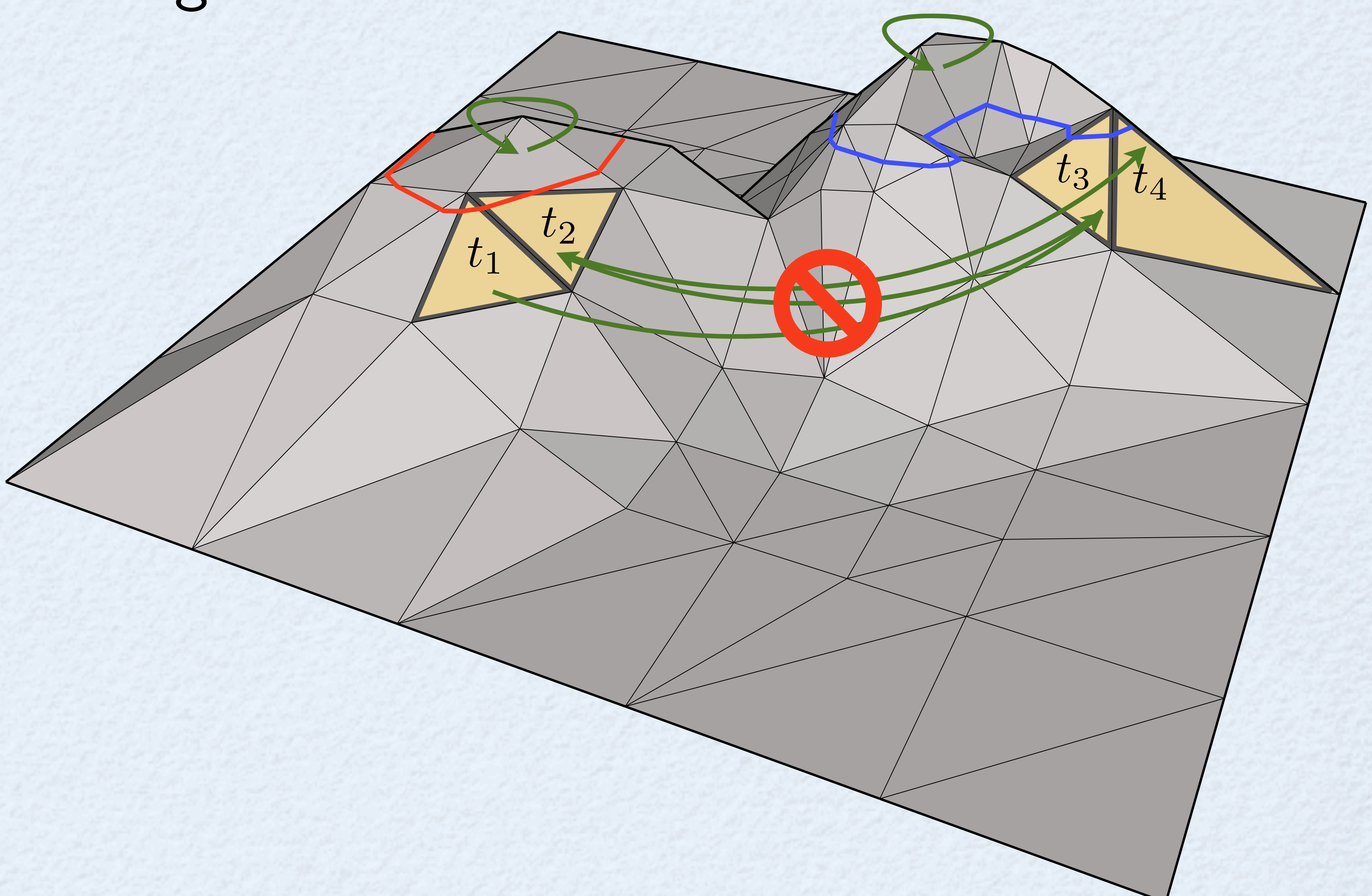
Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is, and can be found in $O(\text{Sort}(N))$ I/Os, a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \quad \Longrightarrow \quad t_1 \prec t_4 \prec t_2$$

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



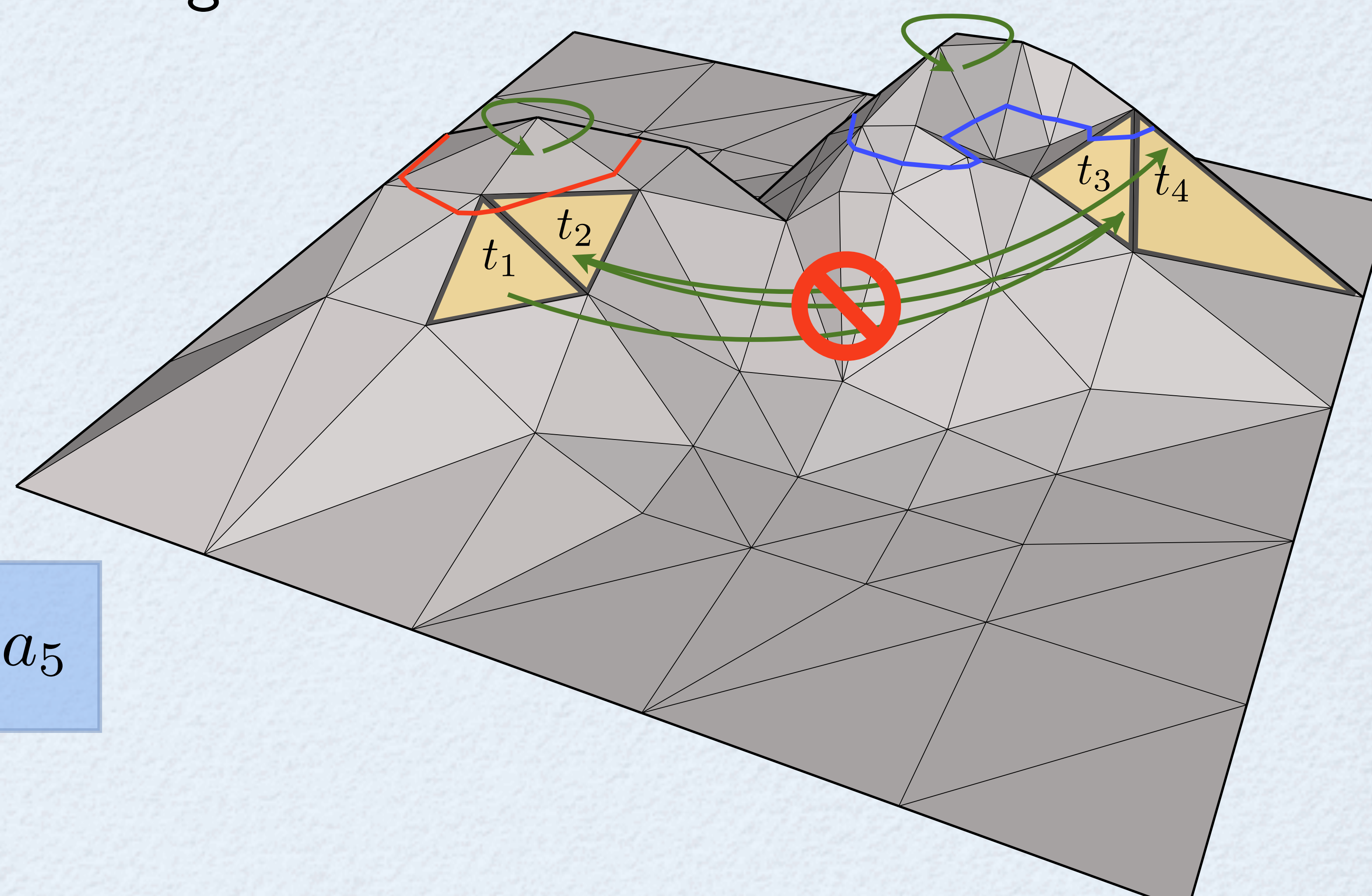
Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is , and can be found in $O(\text{Sort}(N))$ I/Os, a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \quad \Longrightarrow \quad t_1 \prec t_4 \prec t_2$$

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



$a_1 a_2 b_1 c_1 c_2 c_3 b_2 b_3 d_1 d_2 b_4 a_3 a_4 a_5$

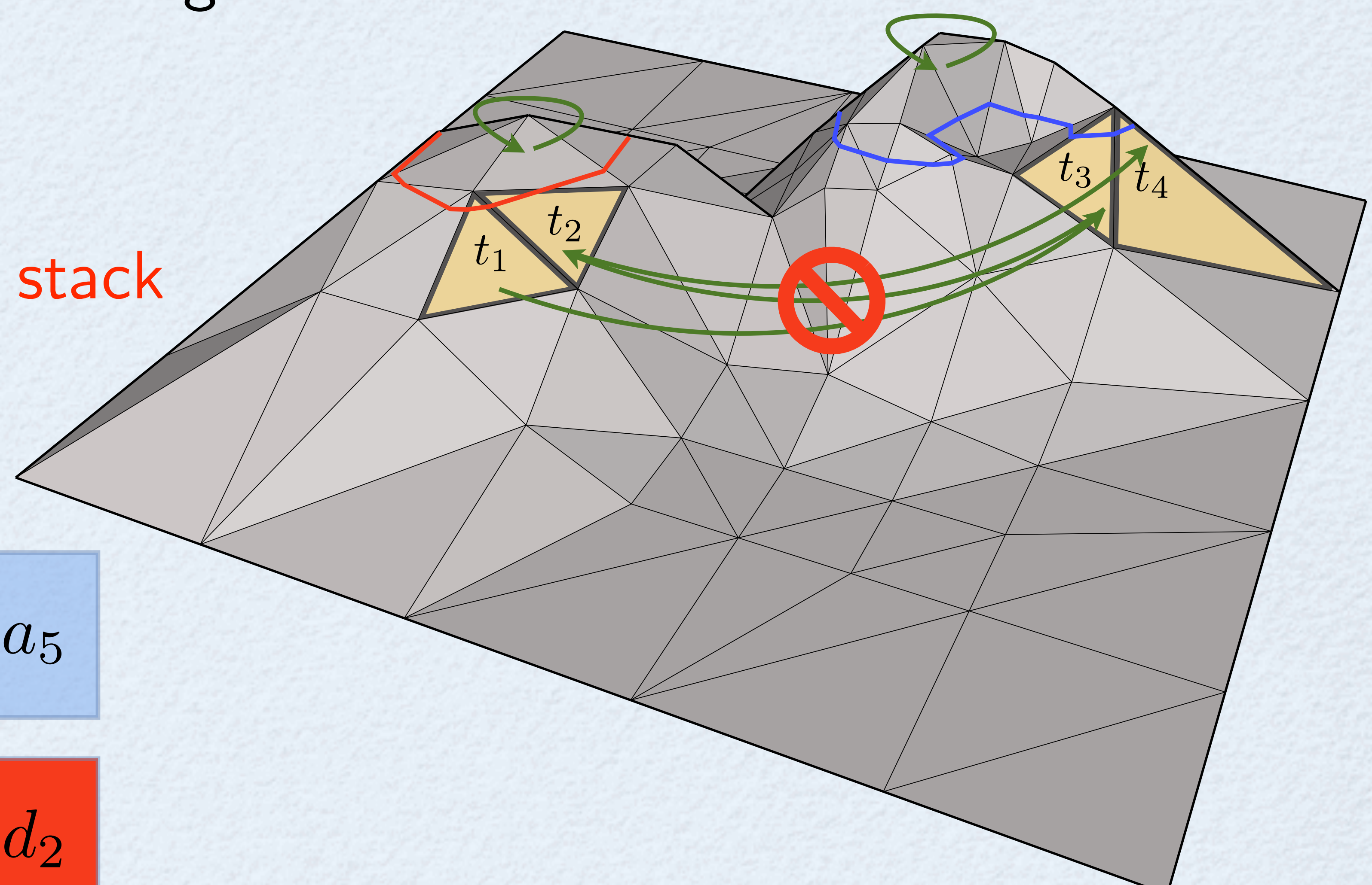
Level Ordering Theorem

Theorem. For any terrain $\mathbb{M}$, there is , and can be found in $O(\text{Sort}(N))$ I/Os, a total ordering "$\prec$" of triangles of $\mathbb{M}$, s.t. for any ℓ:

1. Triangles of each contour in Δ_ℓ are $\prec$-sorted in cw or ccw order.
2. For contours C and D of Δ_ℓ and $t_1, t_2 \in C$ and $t_3, t_4 \in D$:

$$t_1 \prec t_3 \prec t_2 \implies t_1 \prec t_4 \prec t_2$$

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



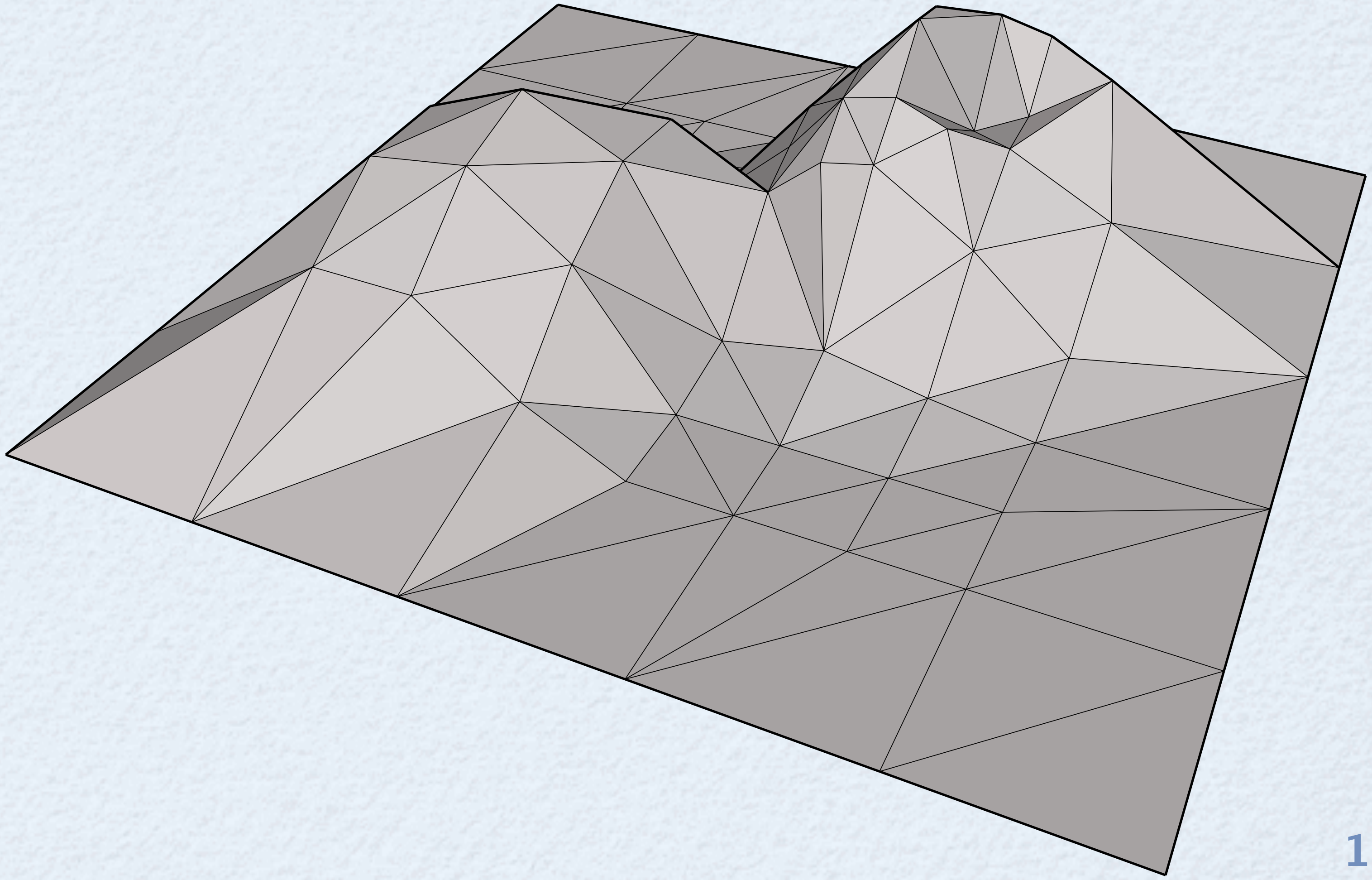
Can separate contours using a stack in $O(T/B)$ I/Os.

$a_1 a_2 b_1 c_1 c_2 c_3 b_2 b_3 d_1 d_2 b_4 a_3 a_4 a_5$

$a_1 a_2 a_3 a_4 a_5 b_1 b_2 b_3 b_4 c_1 c_2 c_3 d_1 d_2$

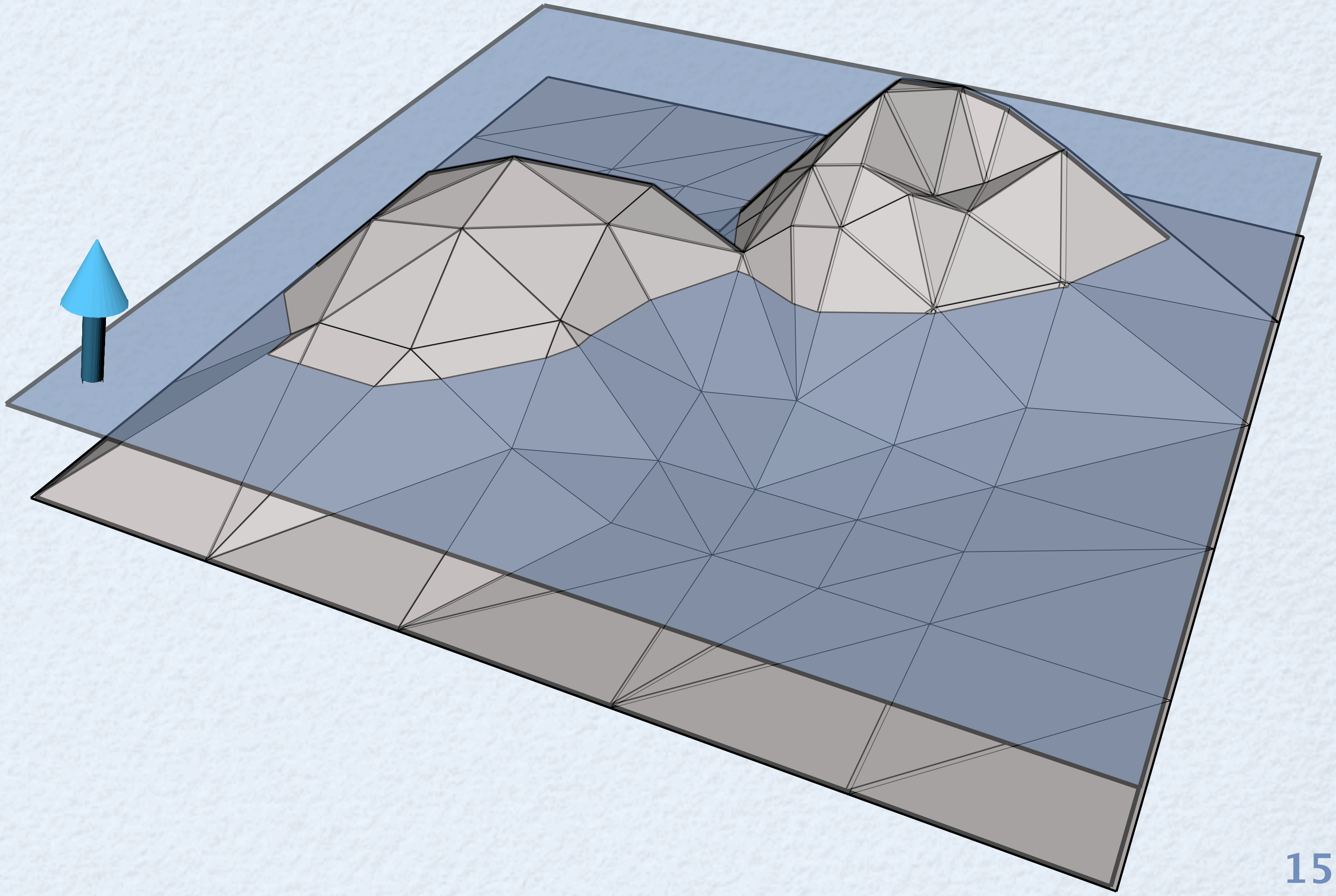
The Algorithm

1. Sweep the terrain by a horizontal plane in the $+z$ direction.



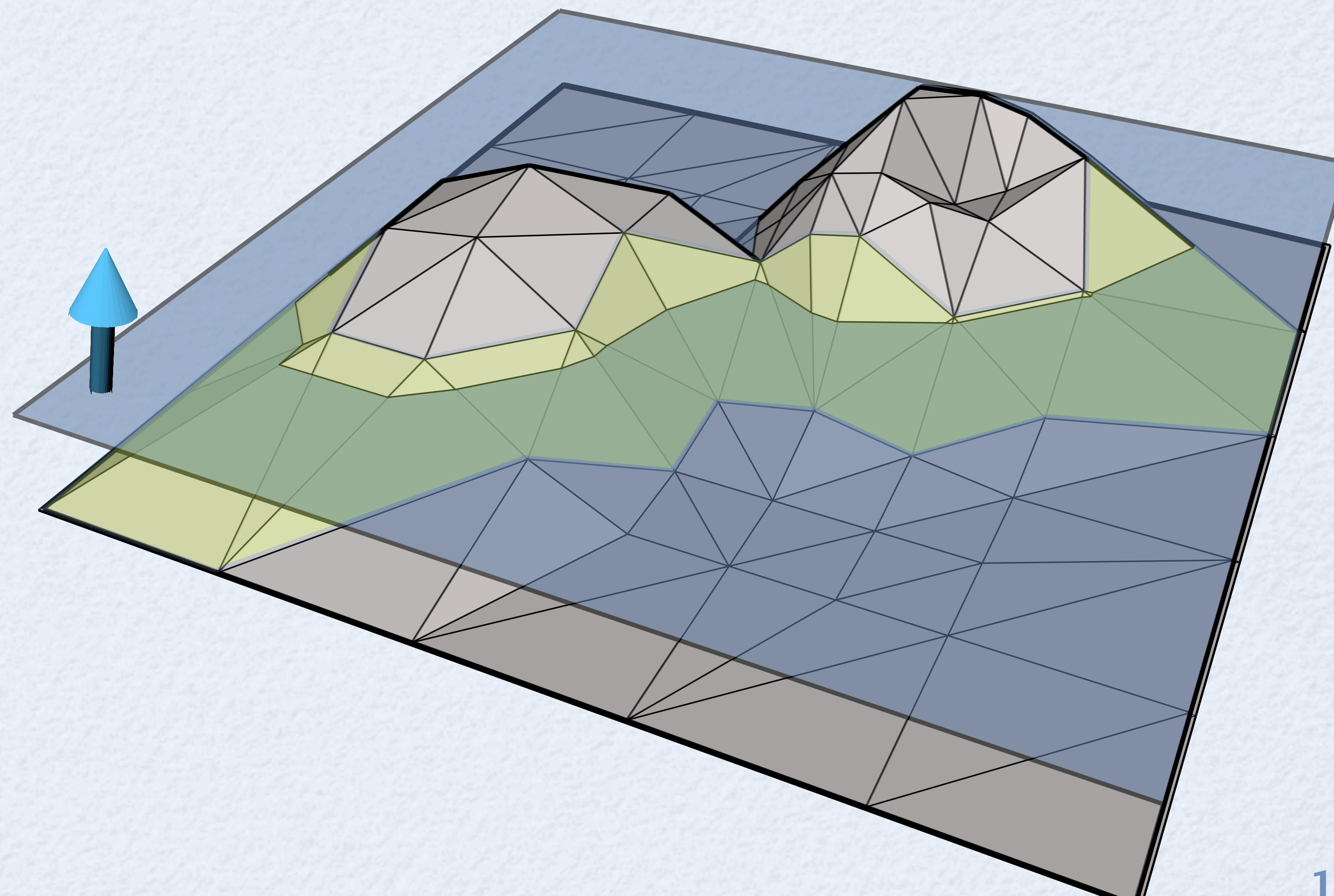
The Algorithm

1. Sweep the terrain by a horizontal plane in the $+z$ direction.



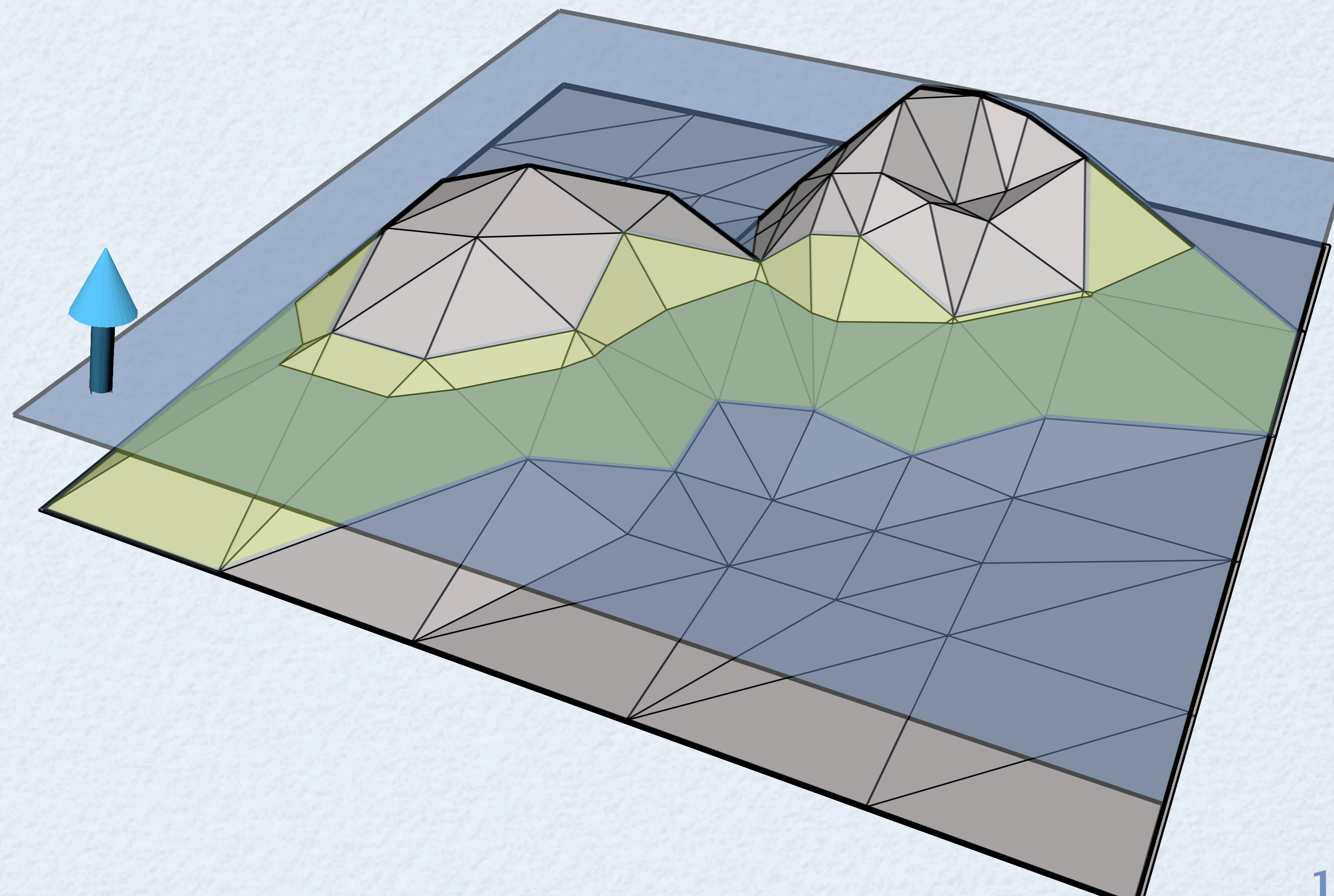
The Algorithm

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 $\prec$.



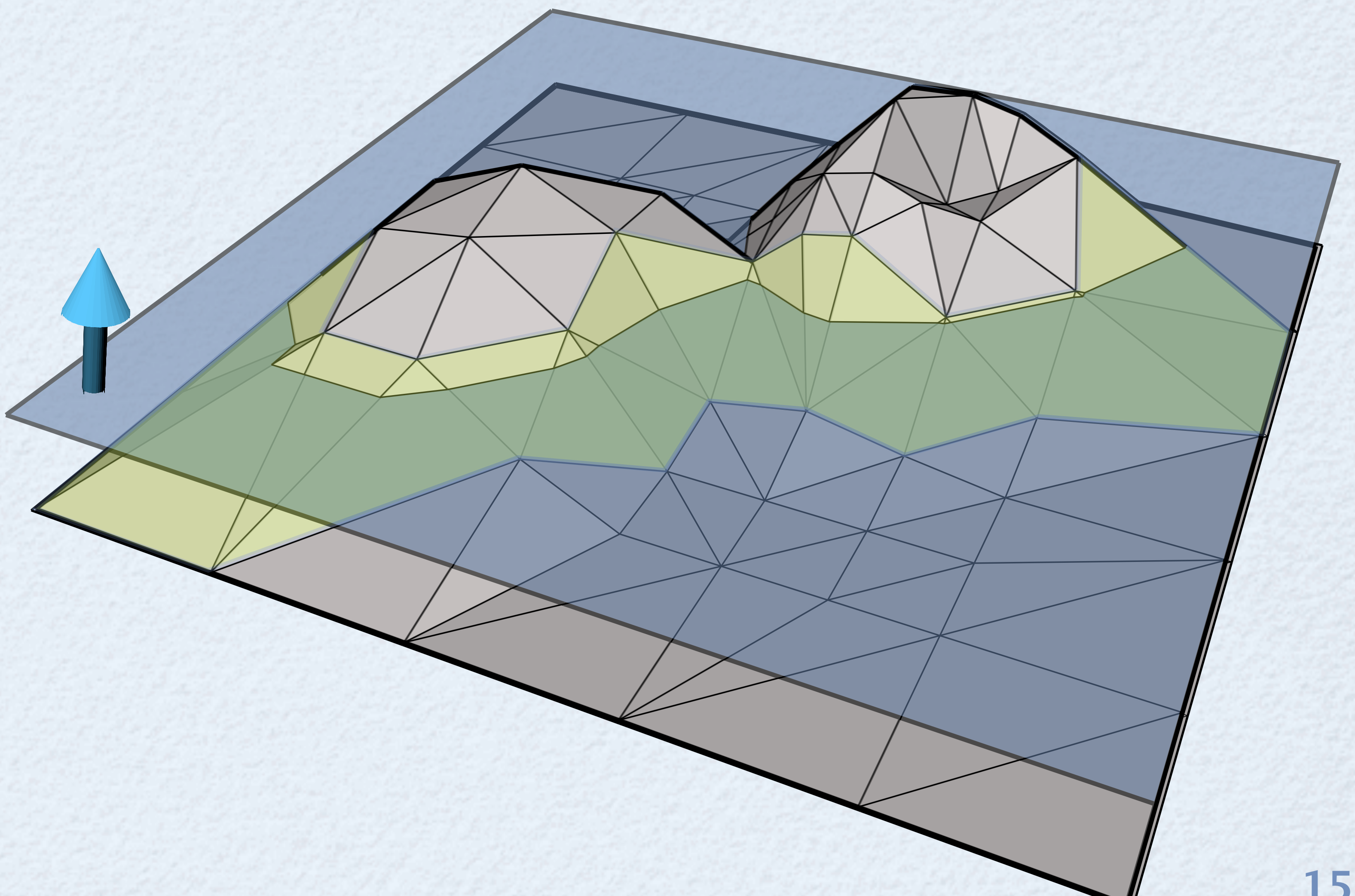
The Algorithm

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 $+z$ direction. Sort(N)
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

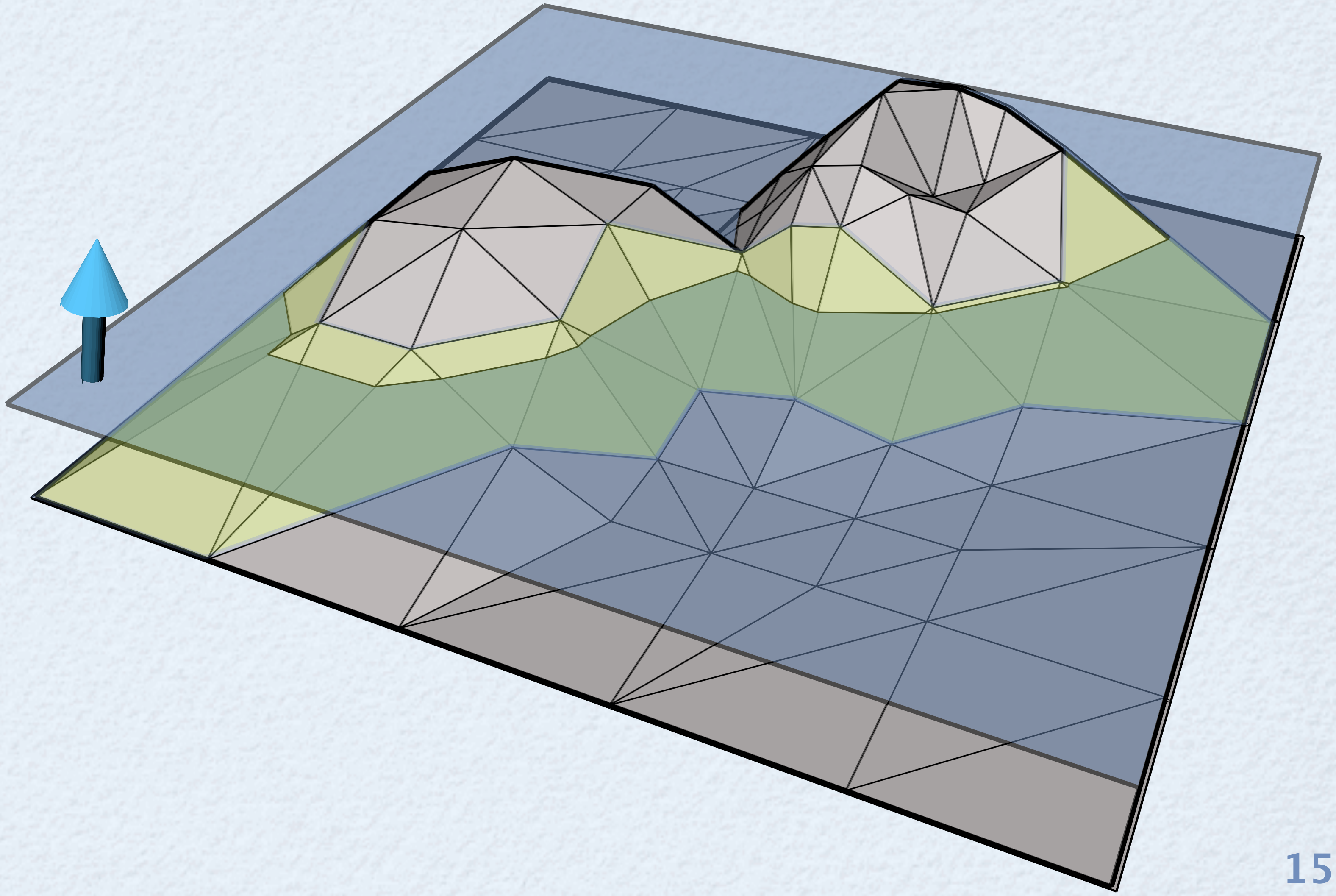
1. Sweep the terrain by a horizontal plane in the $+z$ direction.

Sort(N)

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

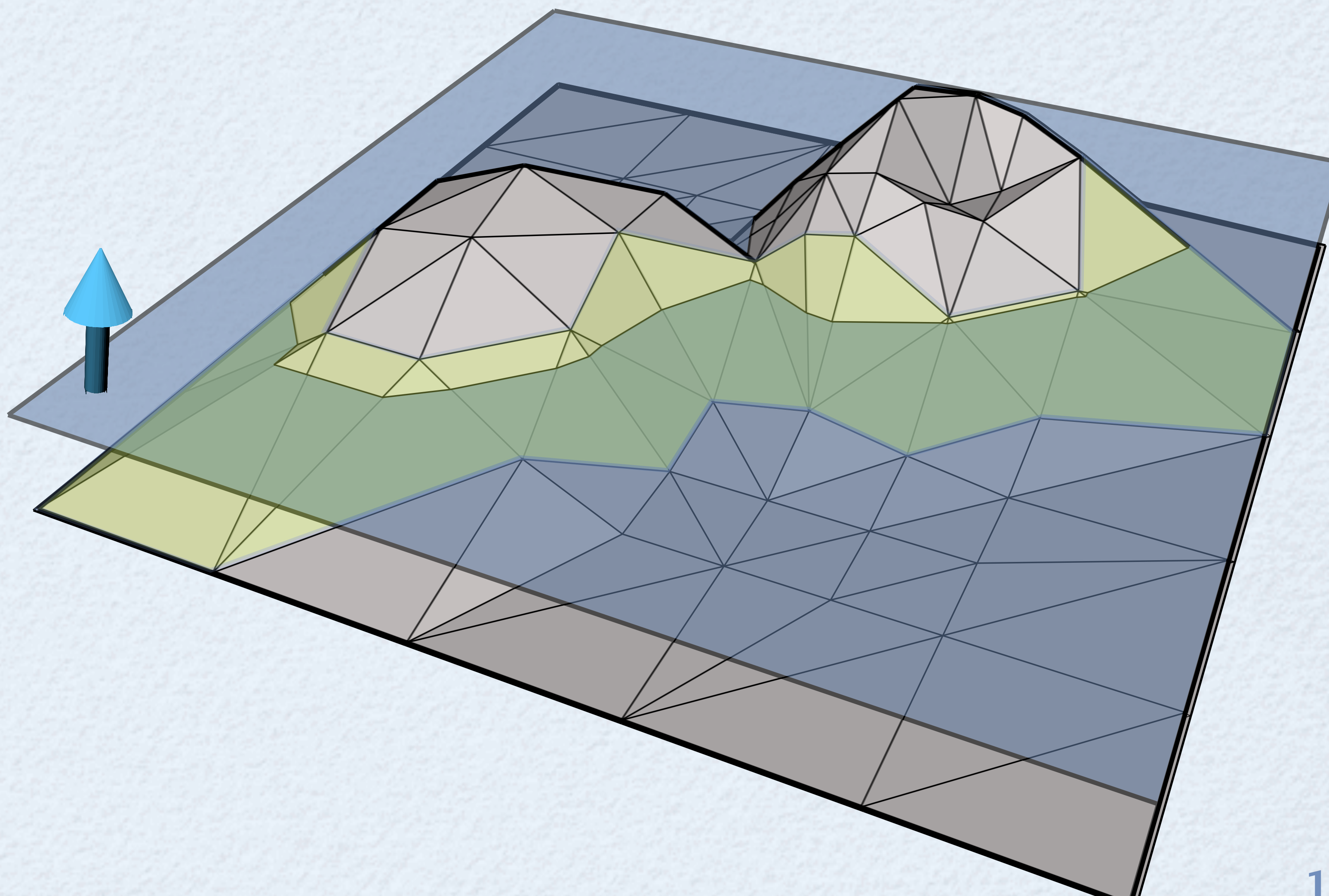
Buffer Tree [Arge'95]

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



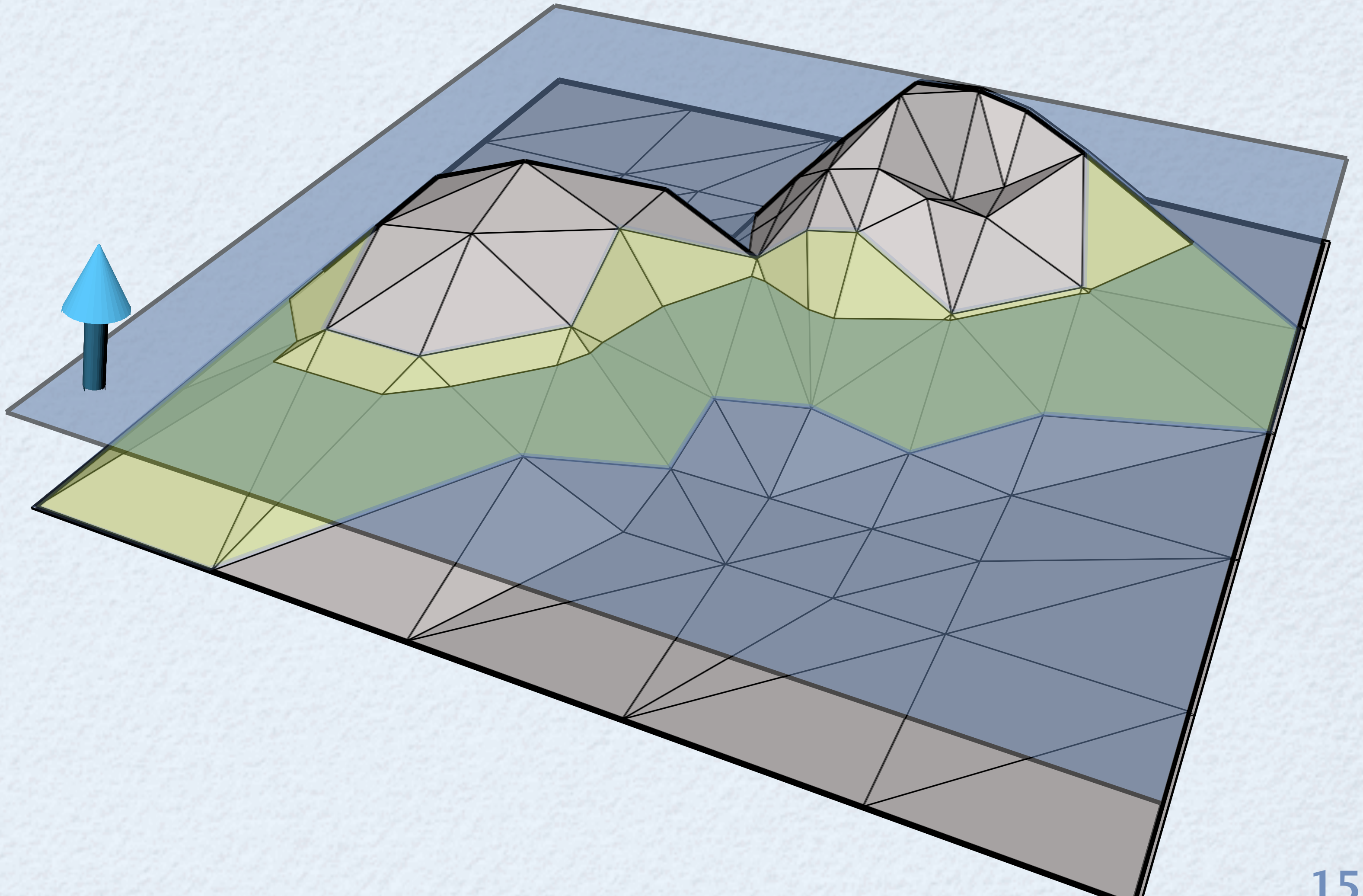
The Algorithm

1. Sweep the terrain by a horizontal plane in the $+z$ direction. Sort(N)
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$. Amortized Sort(N) Buffer Tree [Arge'95]
3. When passing target level $\ell_i \in L$, dump contents of tree to disk.



The Algorithm

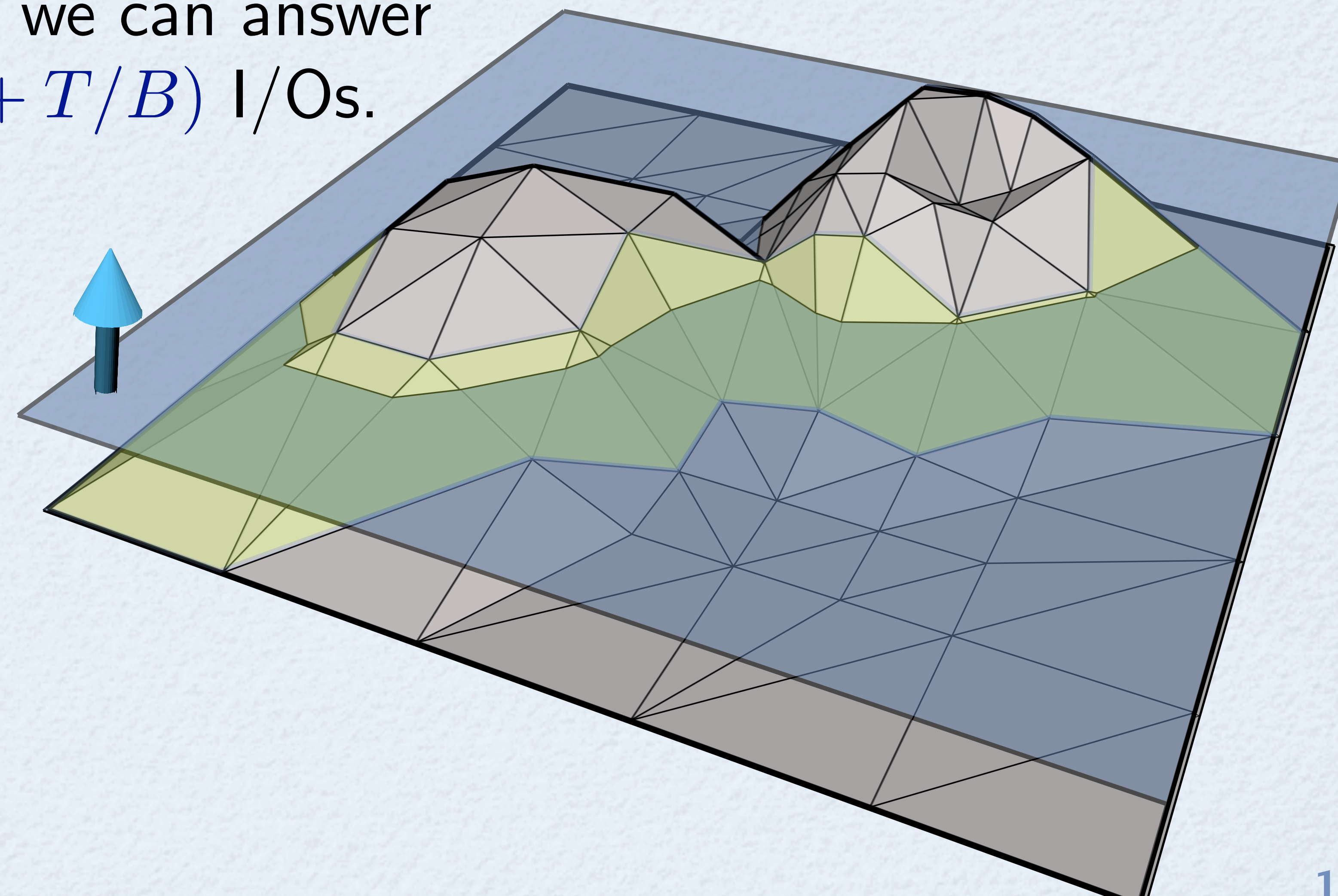
1. Sweep the terrain by a horizontal plane in the $+z$ direction. Sort(N)
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$. Amortized Sort(N) Buffer Tree [Arge'95]
3. When passing target level $\ell_i \in L$, dump contents of tree to disk. T/B



The Algorithm

1. Sweep the terrain by a horizontal plane in the $+z$ direction. Sort(N)
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$. Amortized Sort(N) Buffer Tree [Arge'95]
3. When passing target level $\ell_i \in L$, dump contents of tree to disk. T/B

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

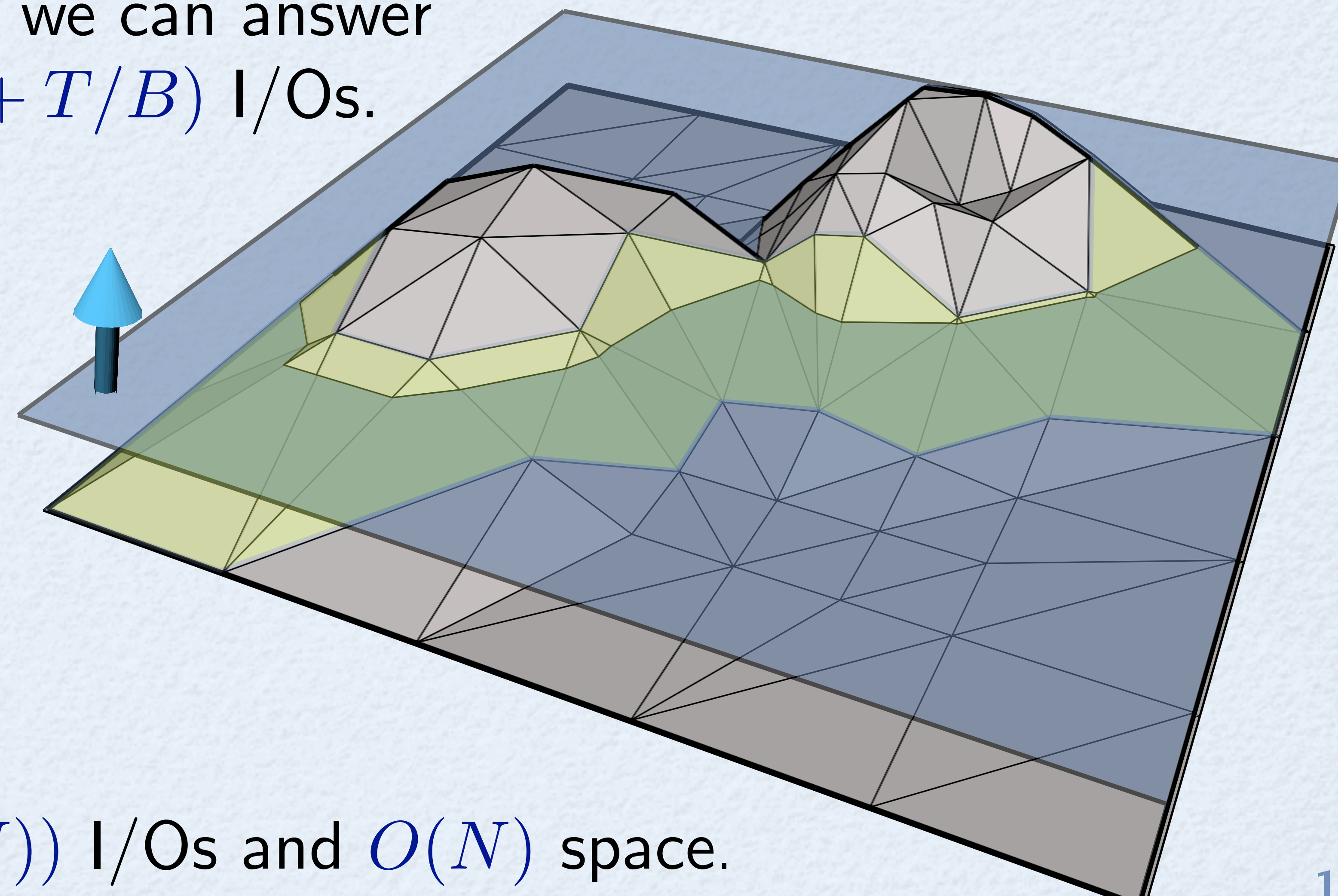


The Algorithm

1. Sweep the terrain by a horizontal plane in the $+z$ direction. ← $\text{Sort}(N)$
2. Keep triangles that intersect the sweep plane in a search tree ordered by $\prec$. ← Amortized $\text{Sort}(N)$

   Buffer Tree [Arge'95]
3. When passing target level $\ell_i \in L$, dump contents of tree to disk. ← T/B

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.

Buffer Tree

Buffer Tree is an I/O-efficient search tree introduced by Arge [Arge'95].

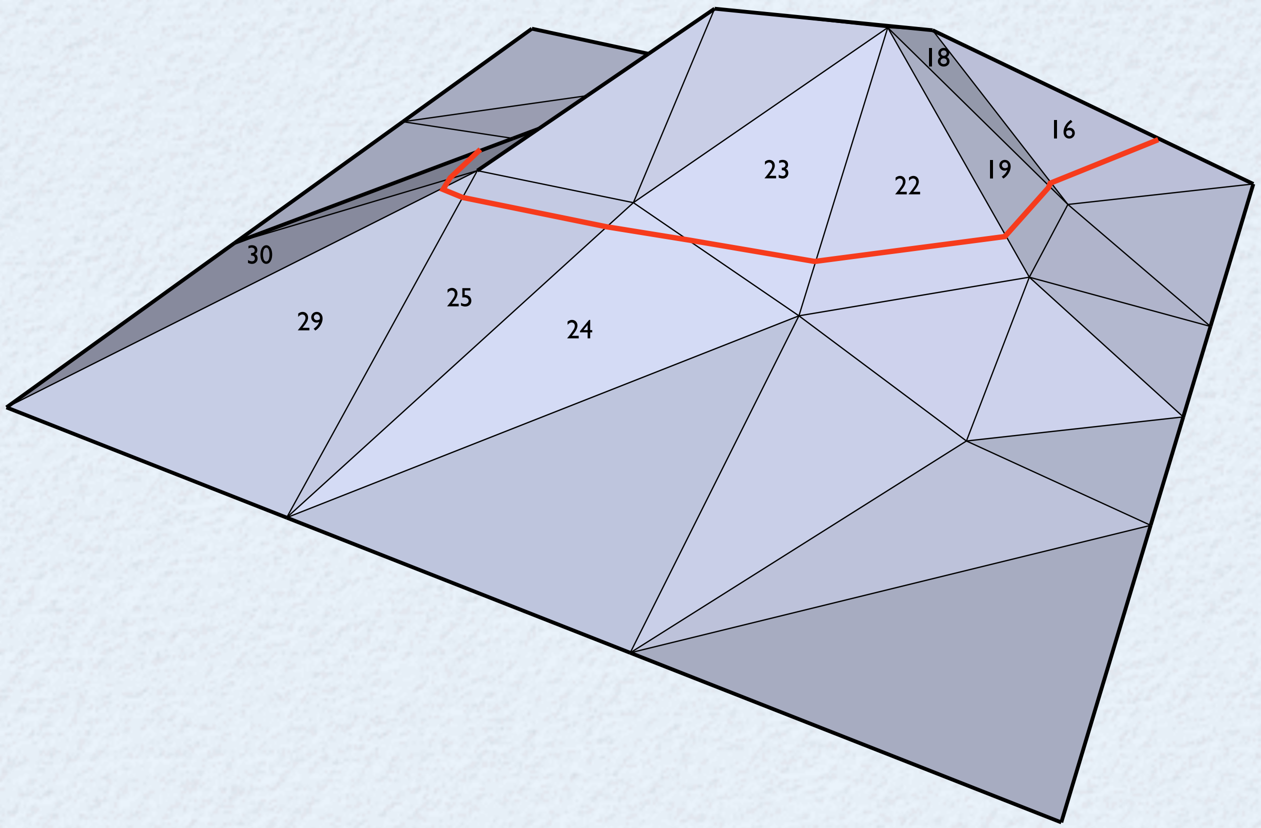
The amortized cost of N intermixed insert and delete operations on an initially empty buffer tree is $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right) = O(\mathsf{Sort}(N))$ I/Os.

Buffer Tree

Buffer Tree is an I/O-efficient search tree introduced by Arge [Arge'95].

The amortized cost of N intermixed insert and delete operations on an initially empty buffer tree is $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right) = O(\text{Sort}(N))$ I/Os.

Flushing the tree writes the entire content of the tree in sorted order on disk, and this takes $O(N/B)$ I/Os.

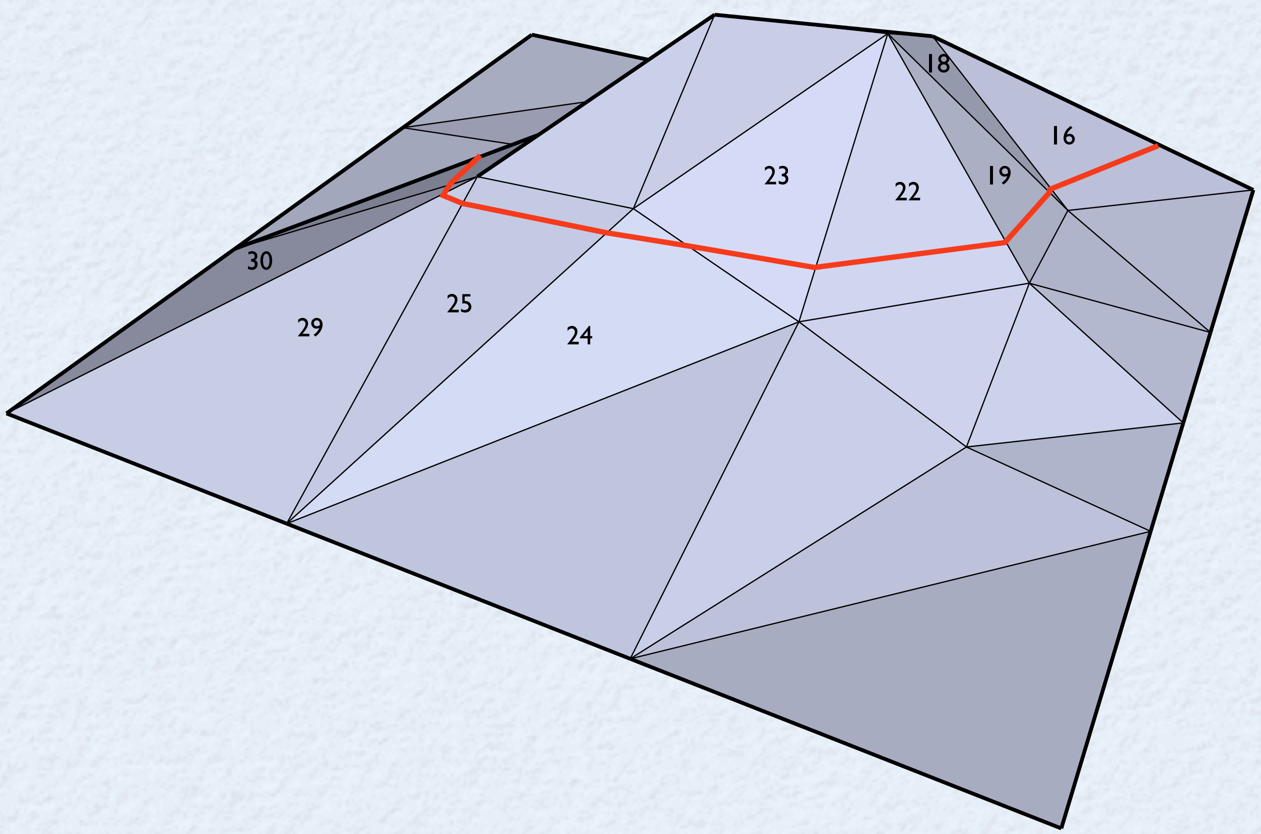


Buffer Tree

Buffer Tree is an I/O-efficient search tree introduced by Arge [Arge'95].

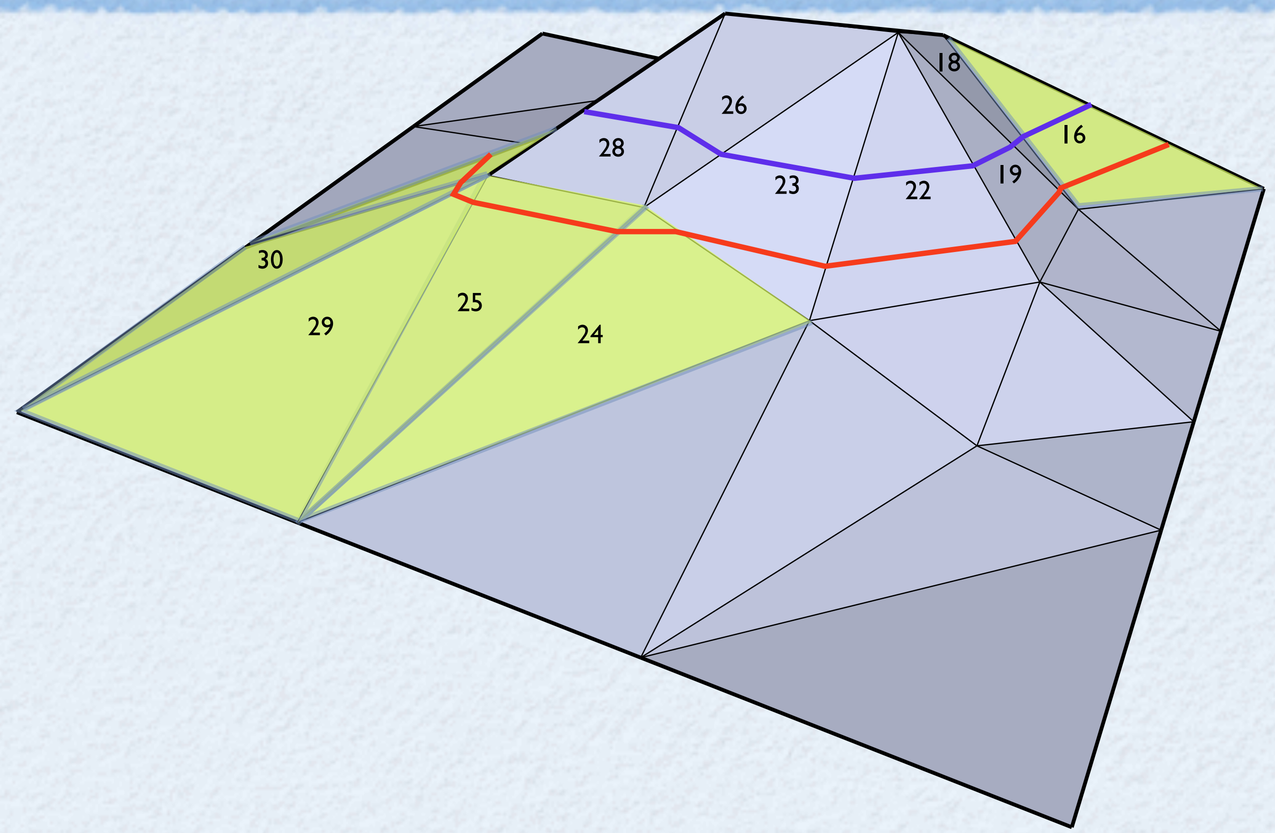
The amortized cost of N intermixed insert and delete operations on an initially empty buffer tree is $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right) = O(\mathsf{Sort}(N))$ I/Os.

Flushing the tree writes the entire content of the tree in sorted order on disk, and this takes $O(N/B)$ I/Os.



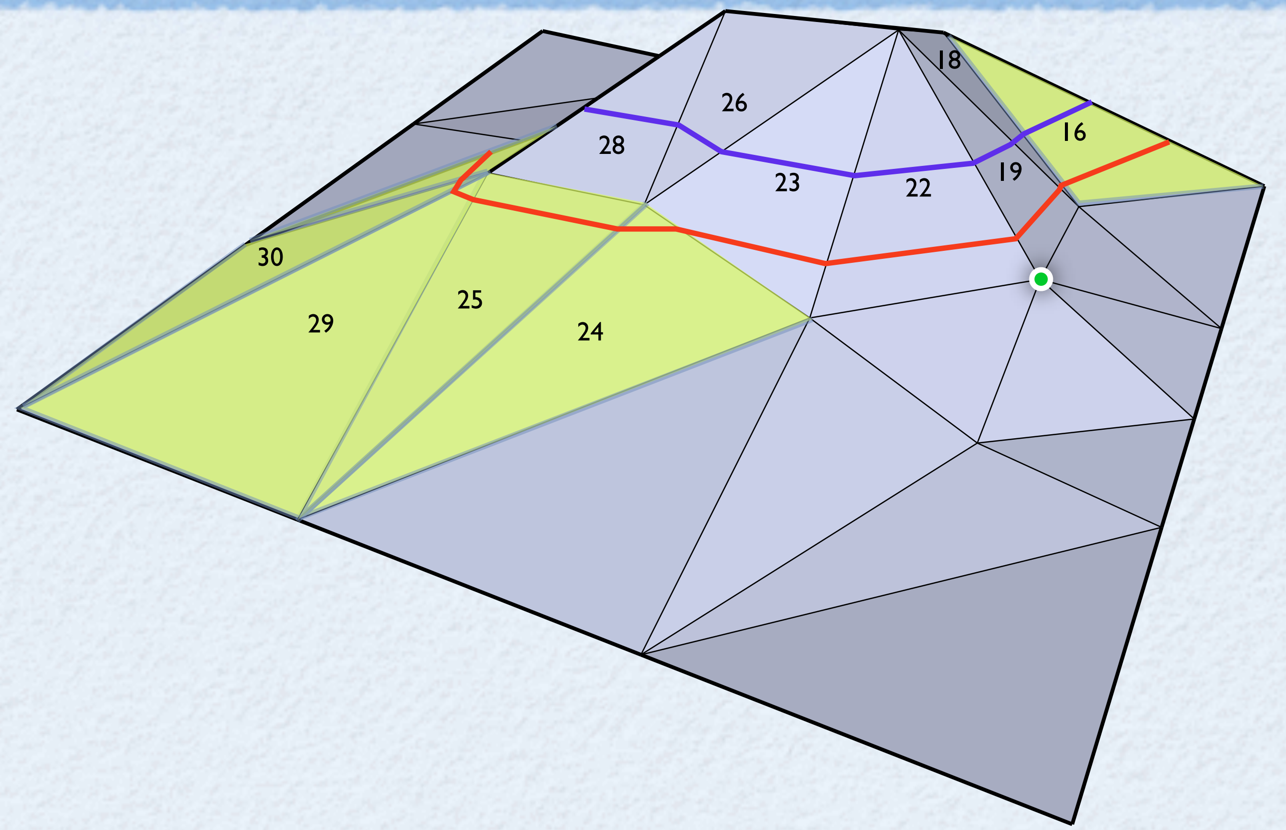
It is possible to make a persistent tree such that each version can be retrieved in $O(\log_B N)$ I/Os.

In More Detail



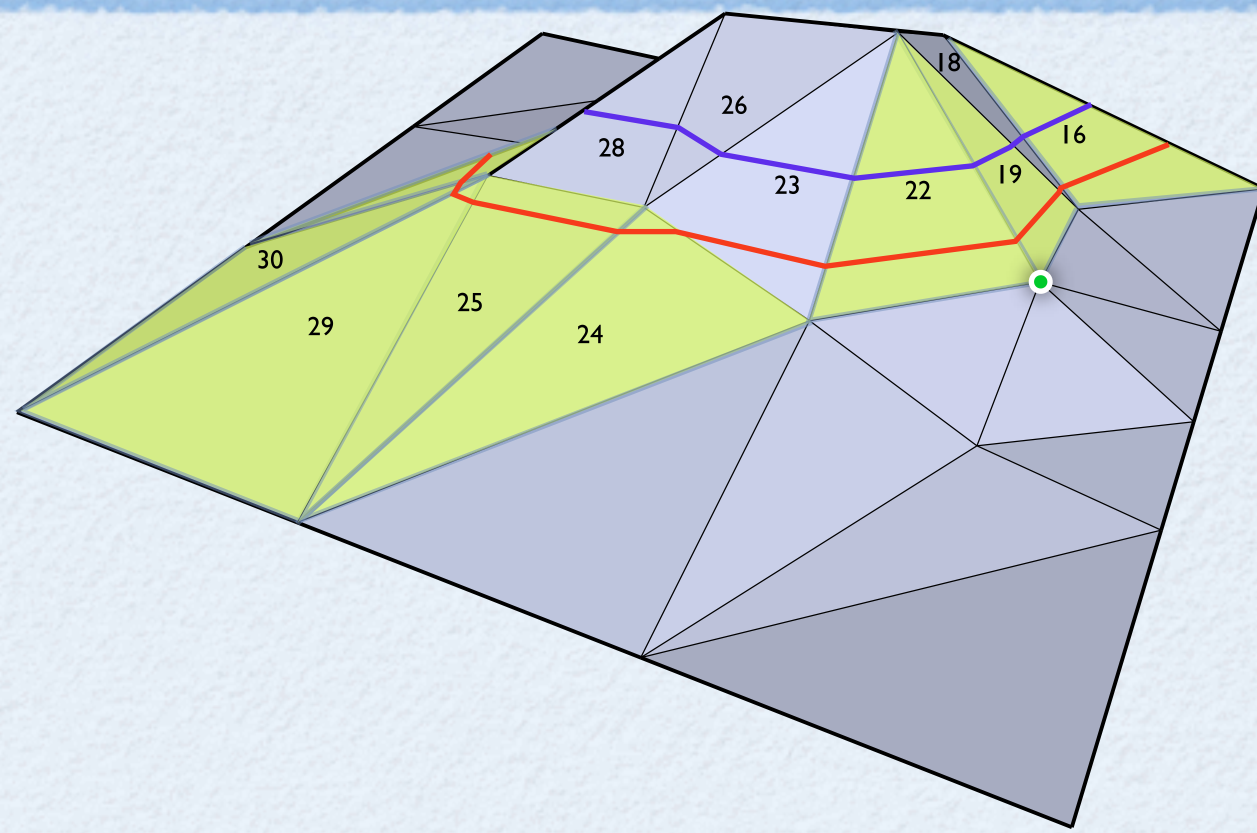
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



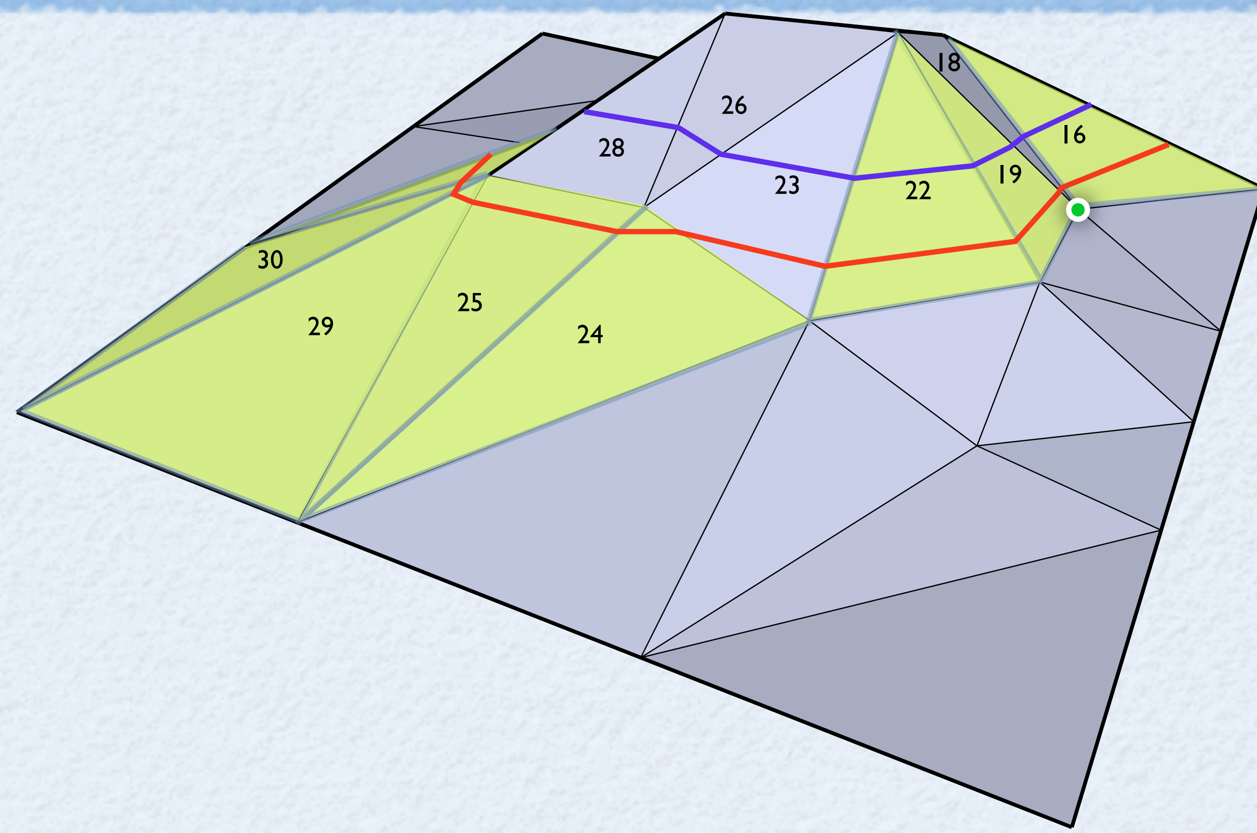
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



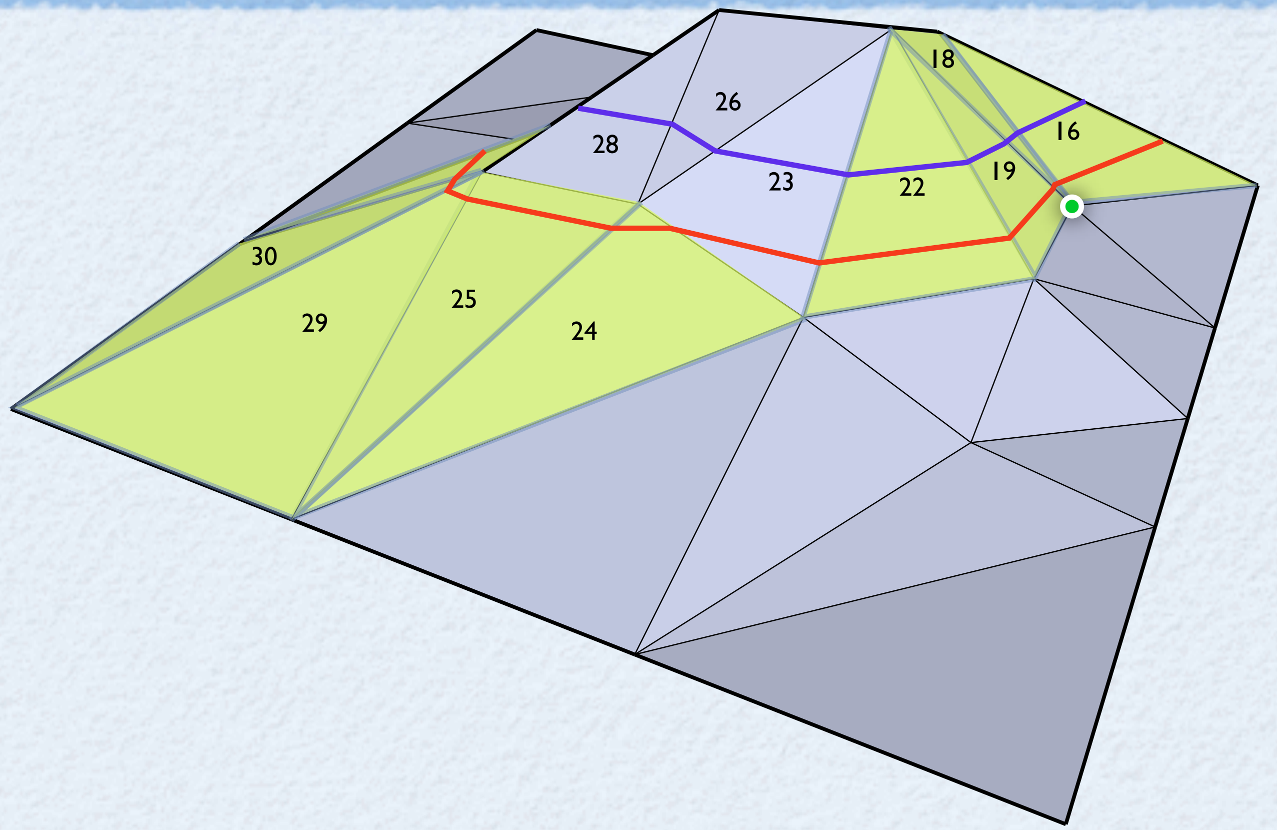
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



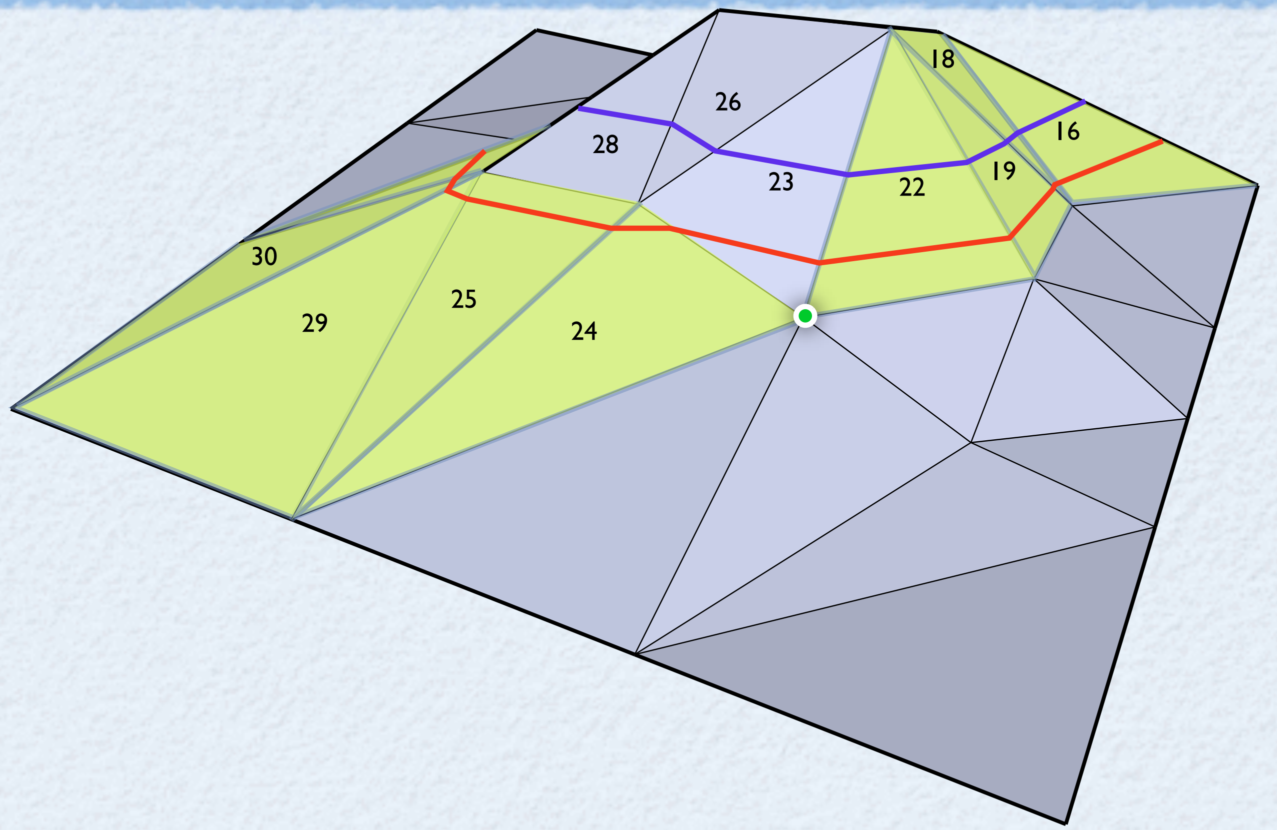
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



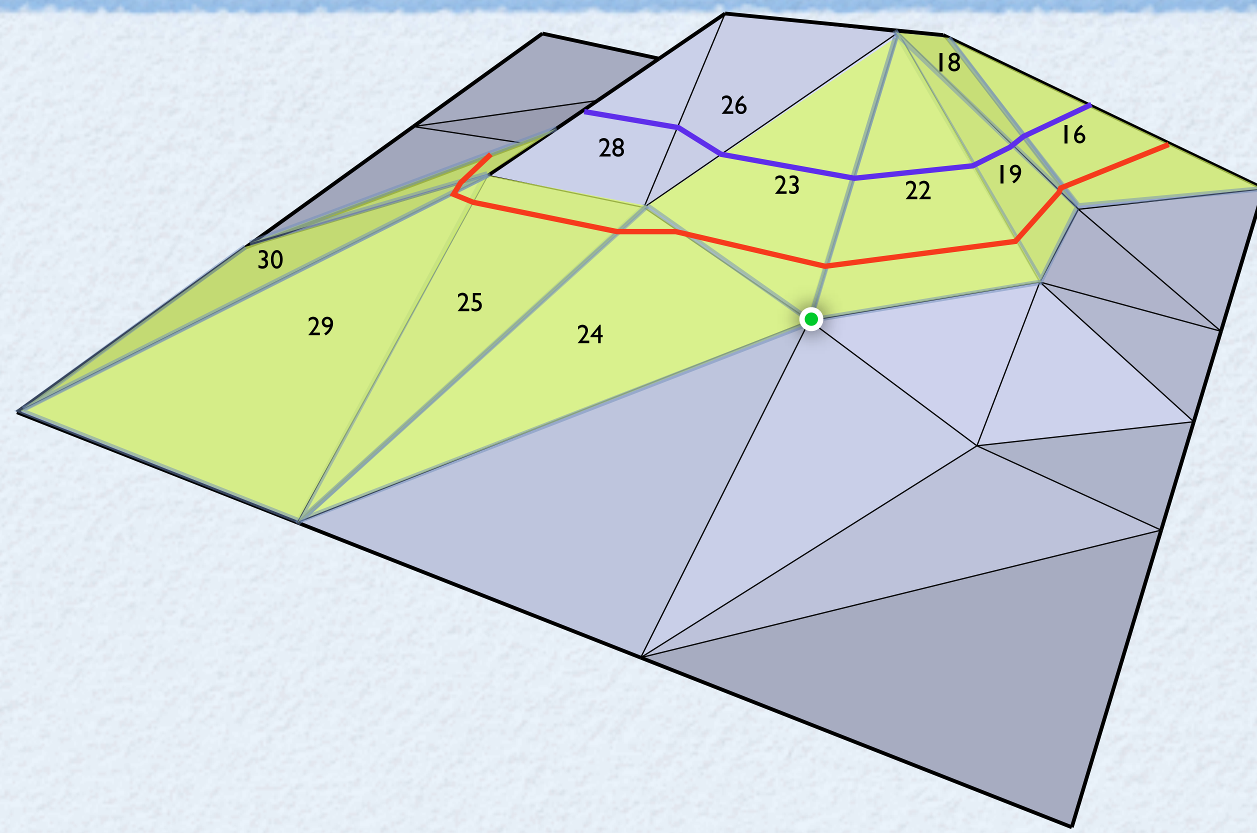
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4.   If $h(v) > \ell$:
5.   Flush the buffer tree; set ℓ to the next level in L.
6.   For each triangle t for which v is the lowest vertex:
7.   If t intersects level ℓ insert t into the buffer tree.
8.   For each triangle t for which v is the heighest vertex:
9.   Delete t from the buffer tree.

In More Detail



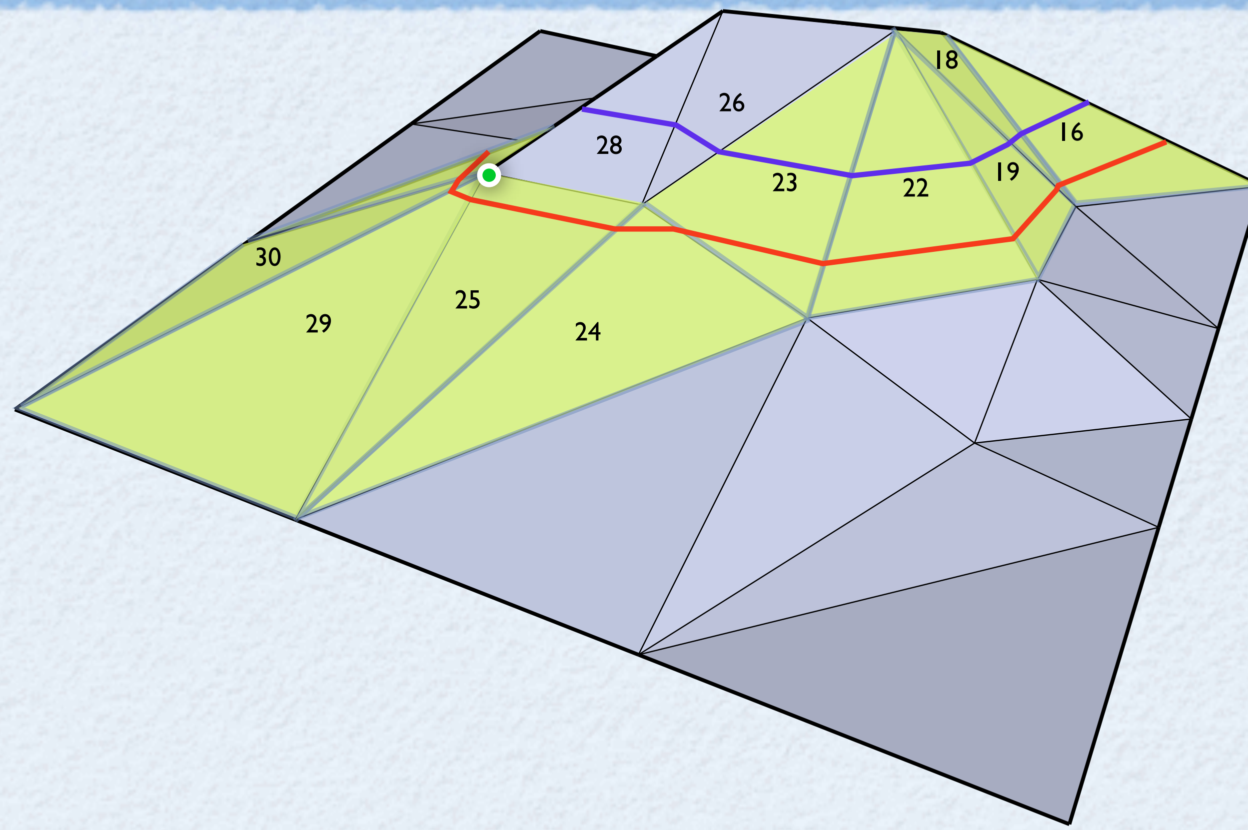
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



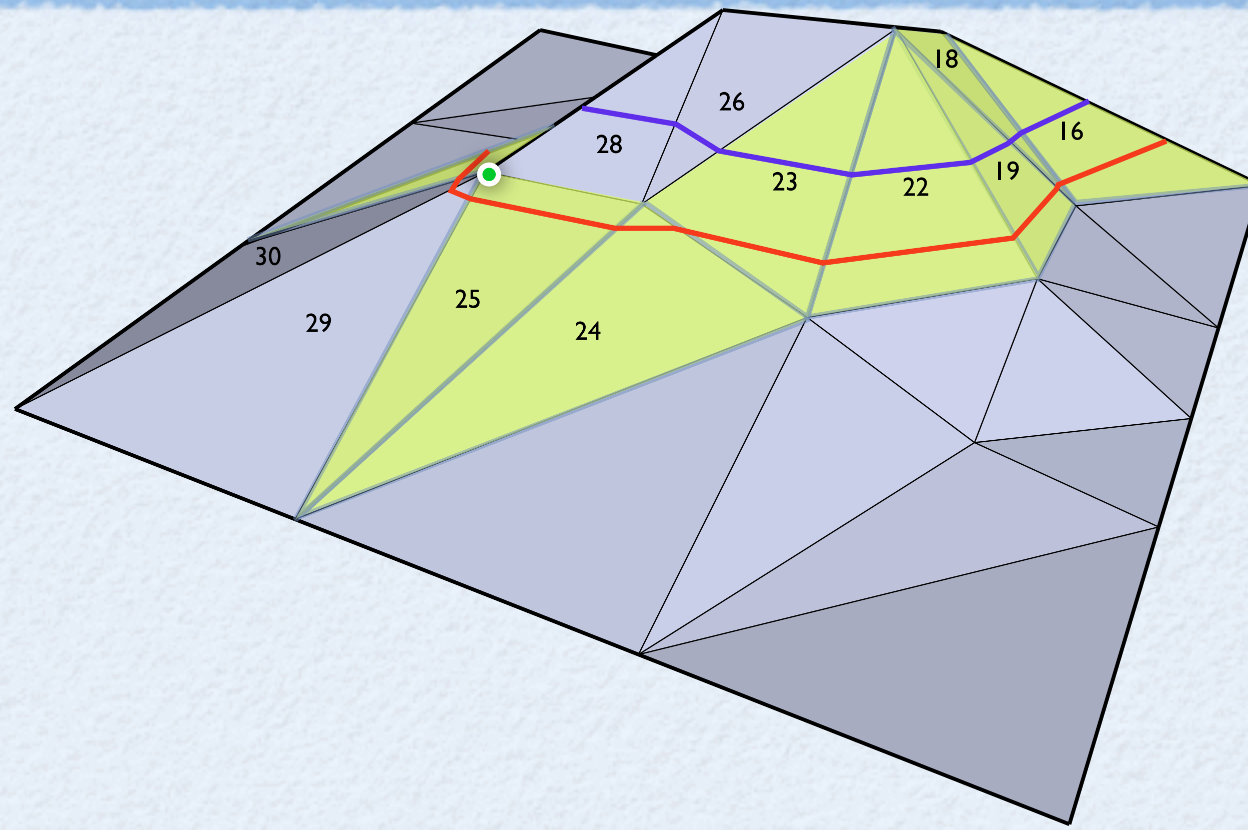
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



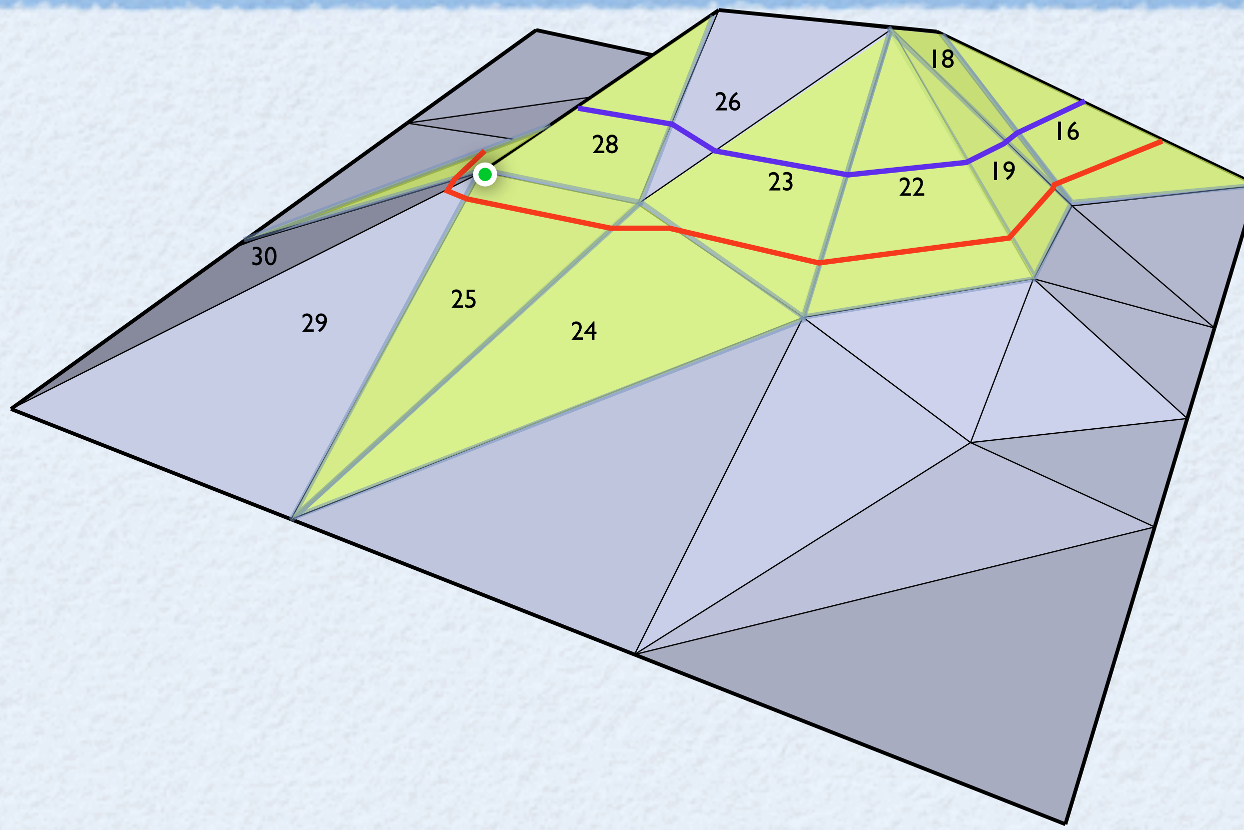
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



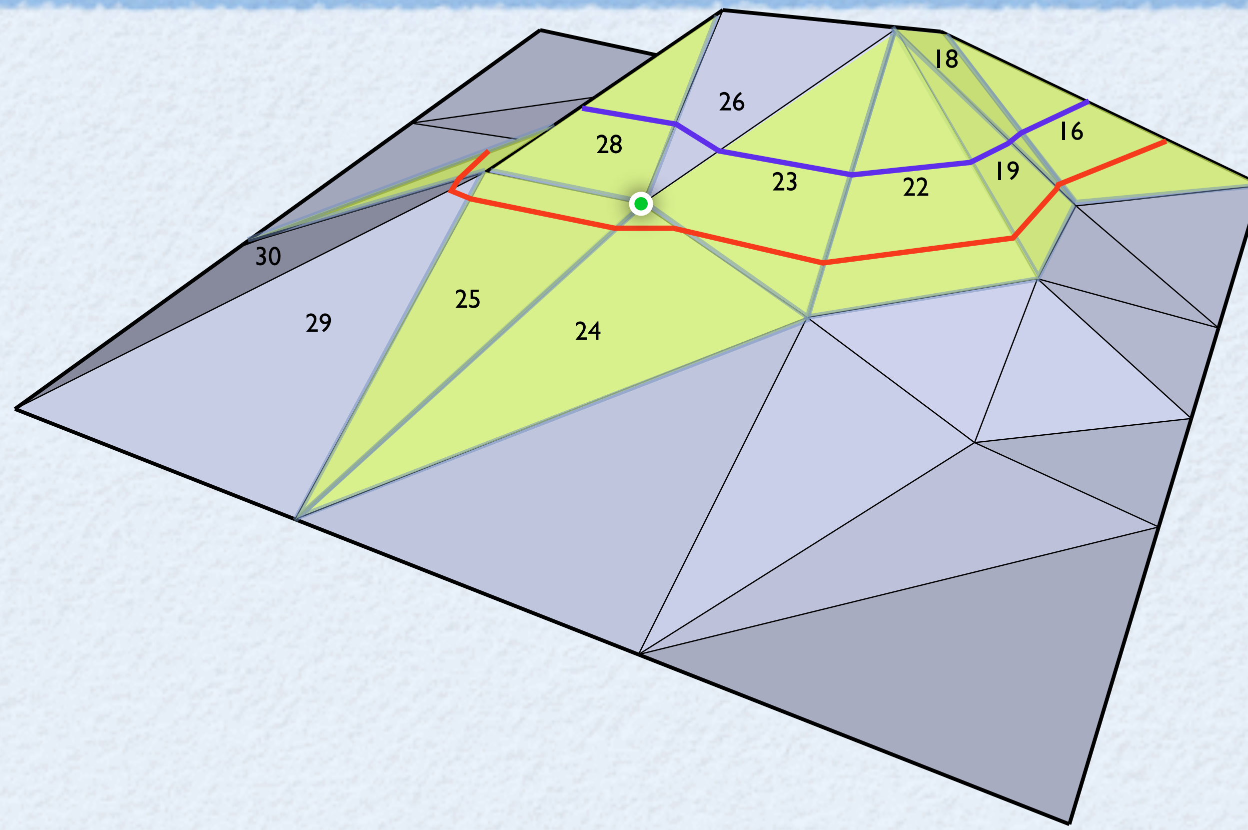
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



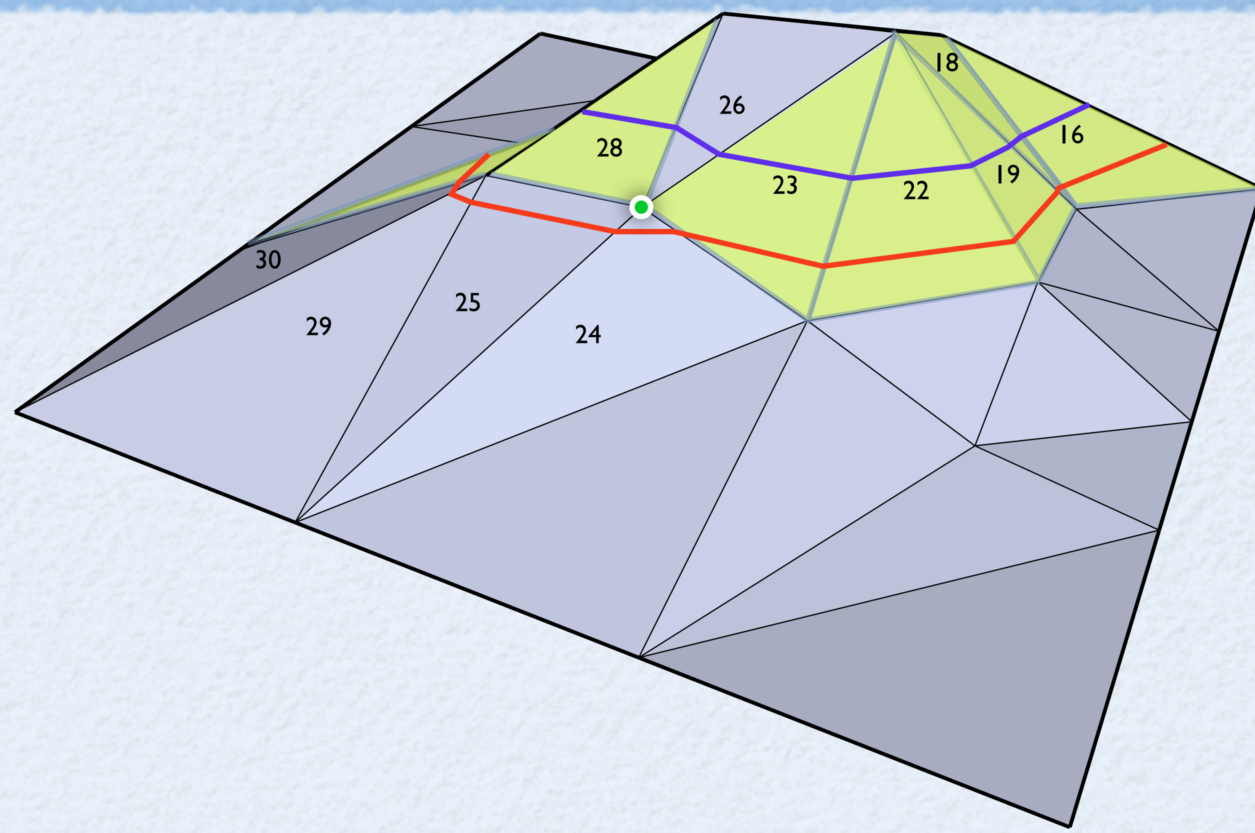
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



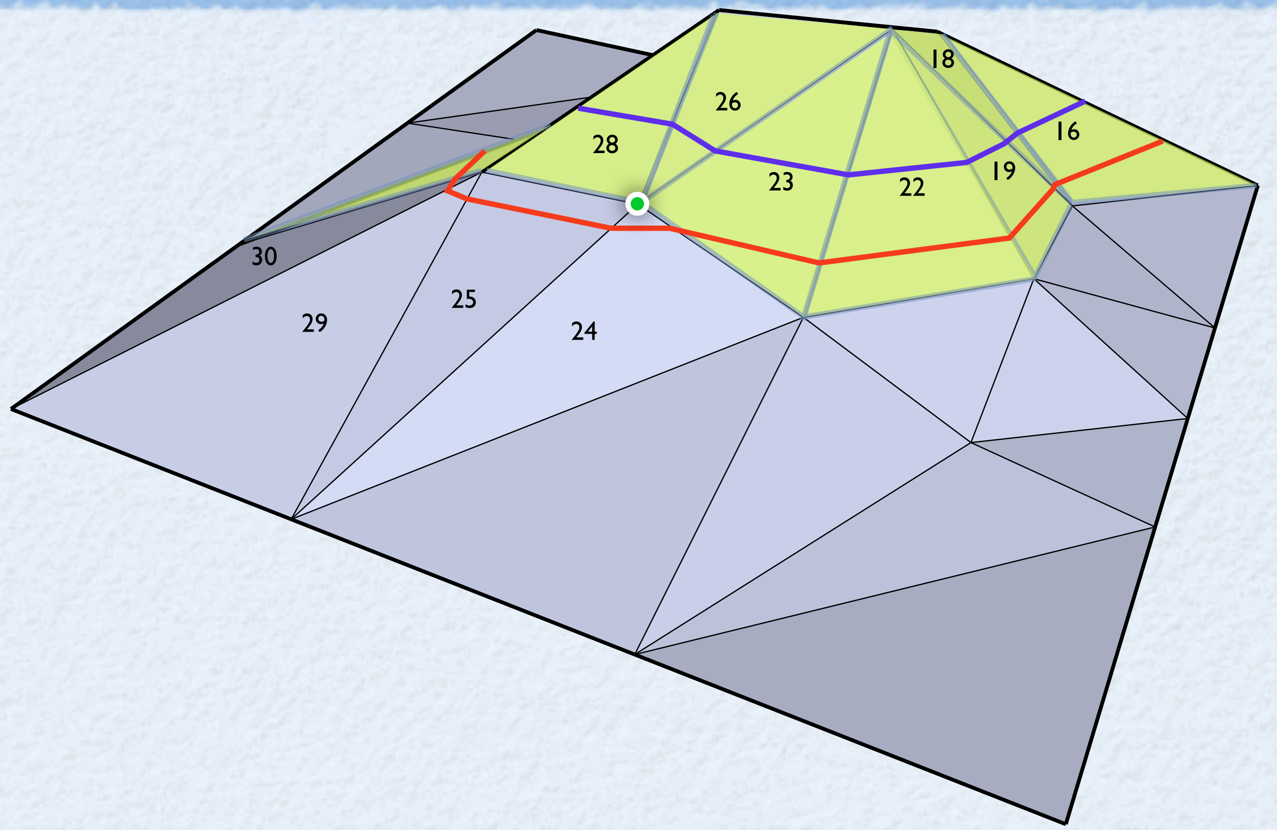
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



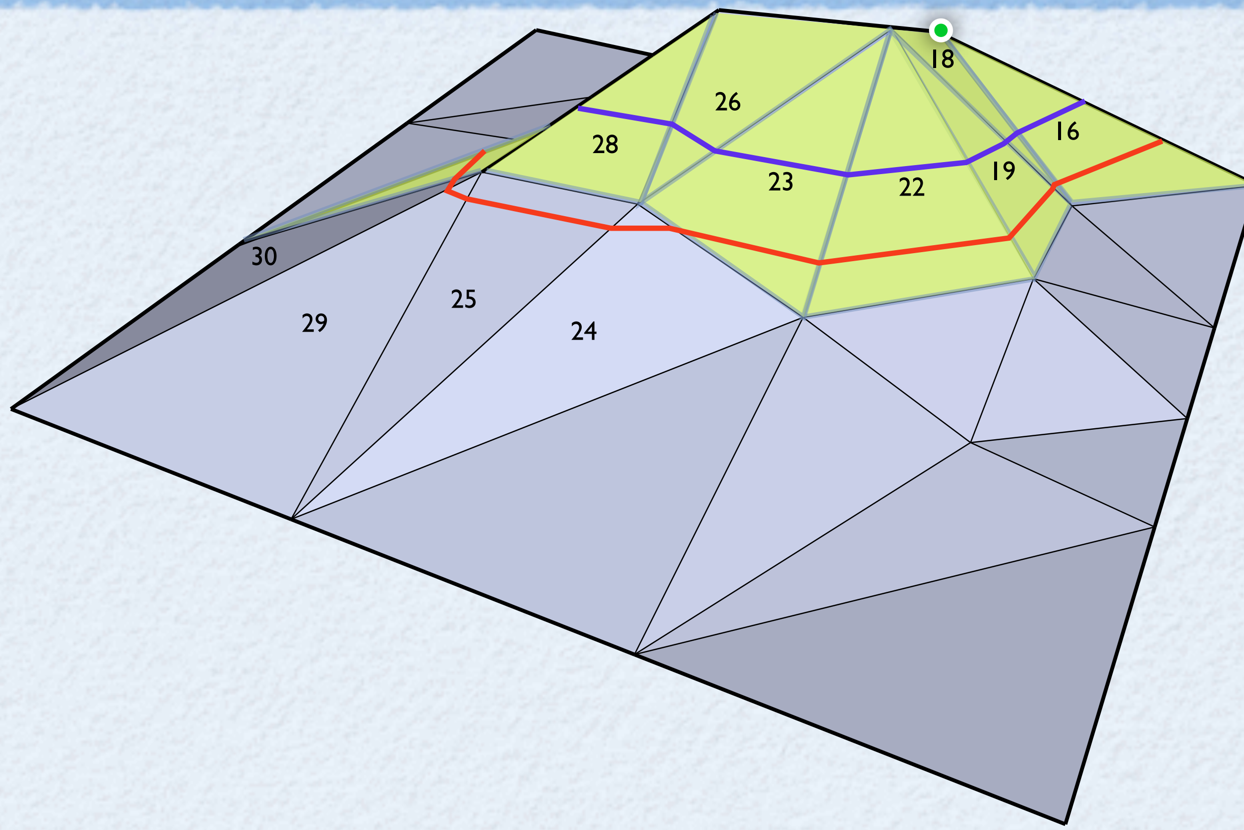
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



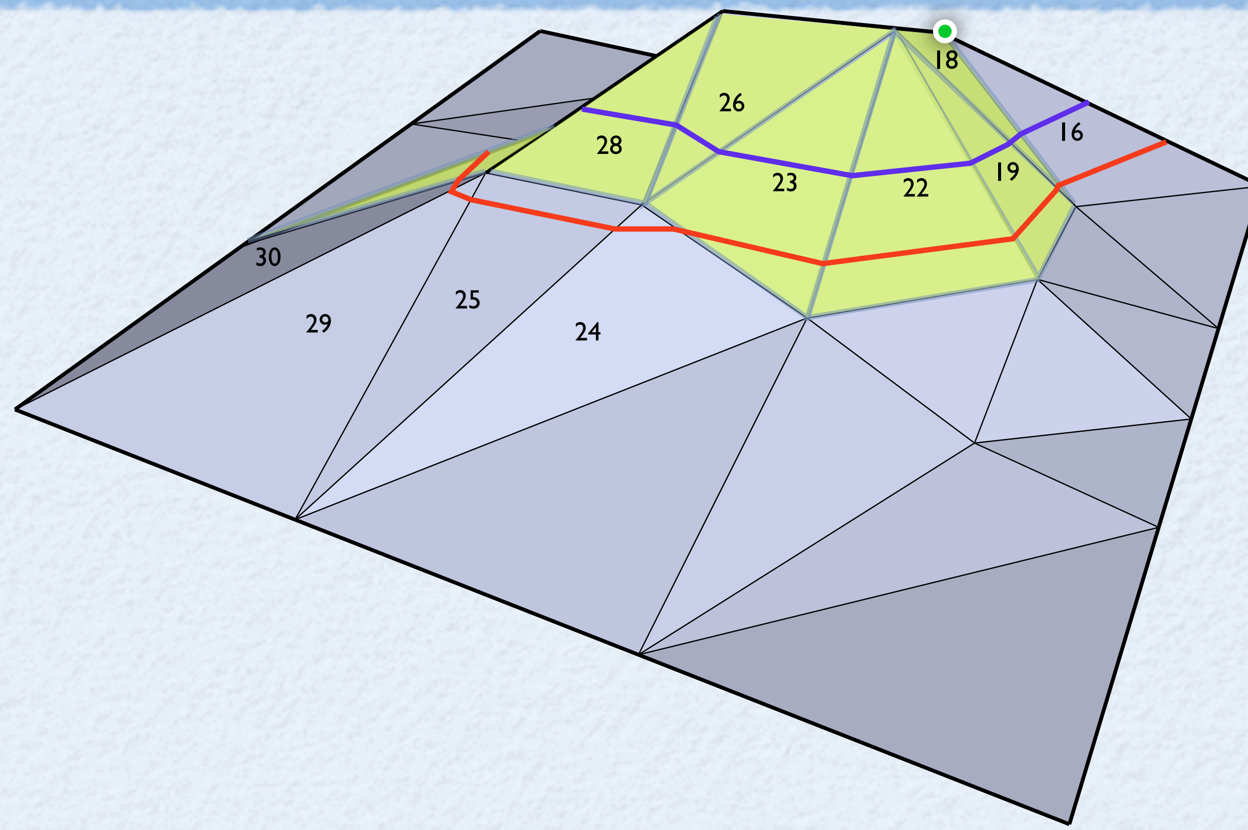
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail



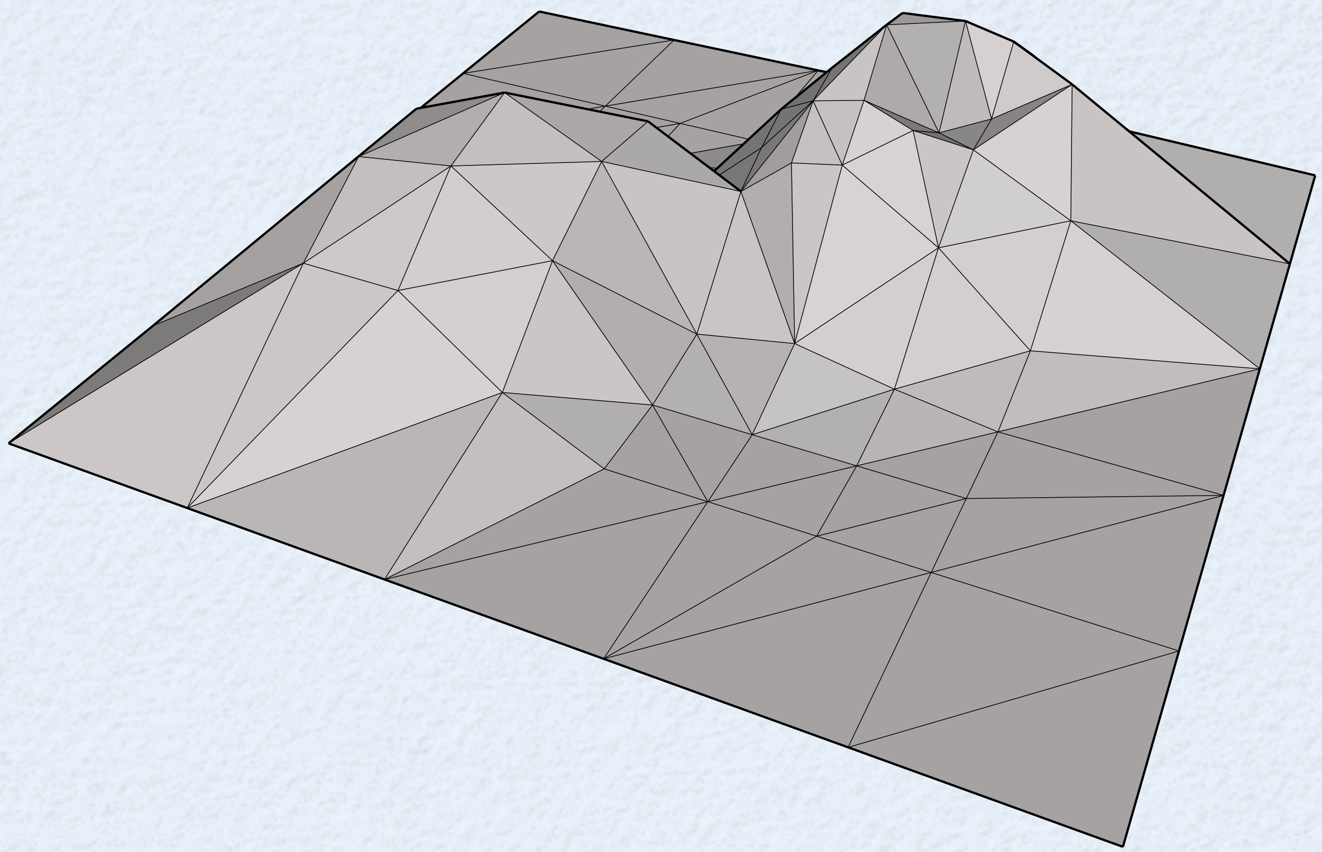
0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

In More Detail

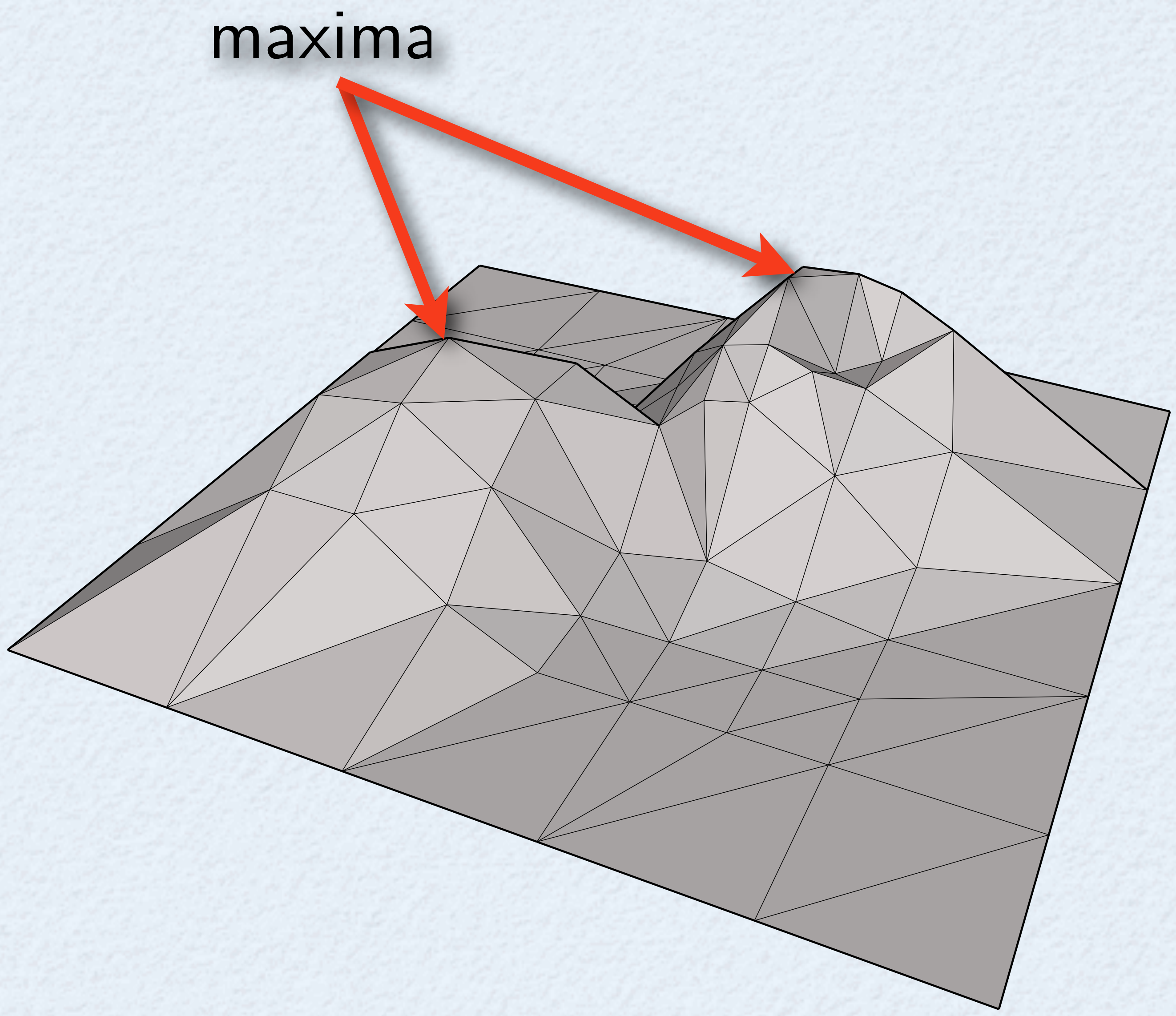


0. Compute a level ordering of $\mathbb{M}$ and the rank of each triangle.
1. Sort the vertices in the ascending order of their heights.
2. Let $\ell = \ell_1$ be the first level of interest in $L = \{\ell_1, \ldots, \ell_k\}$.
3. Scan the sorted list of vertices: at vertex v
4. If $h(v) > \ell$:
5. Flush the buffer tree; set ℓ to the next level in L.
6. For each triangle t for which v is the lowest vertex:
7. If t intersects level ℓ insert t into the buffer tree.
8. For each triangle t for which v is the heighest vertex:
9. Delete t from the buffer tree.

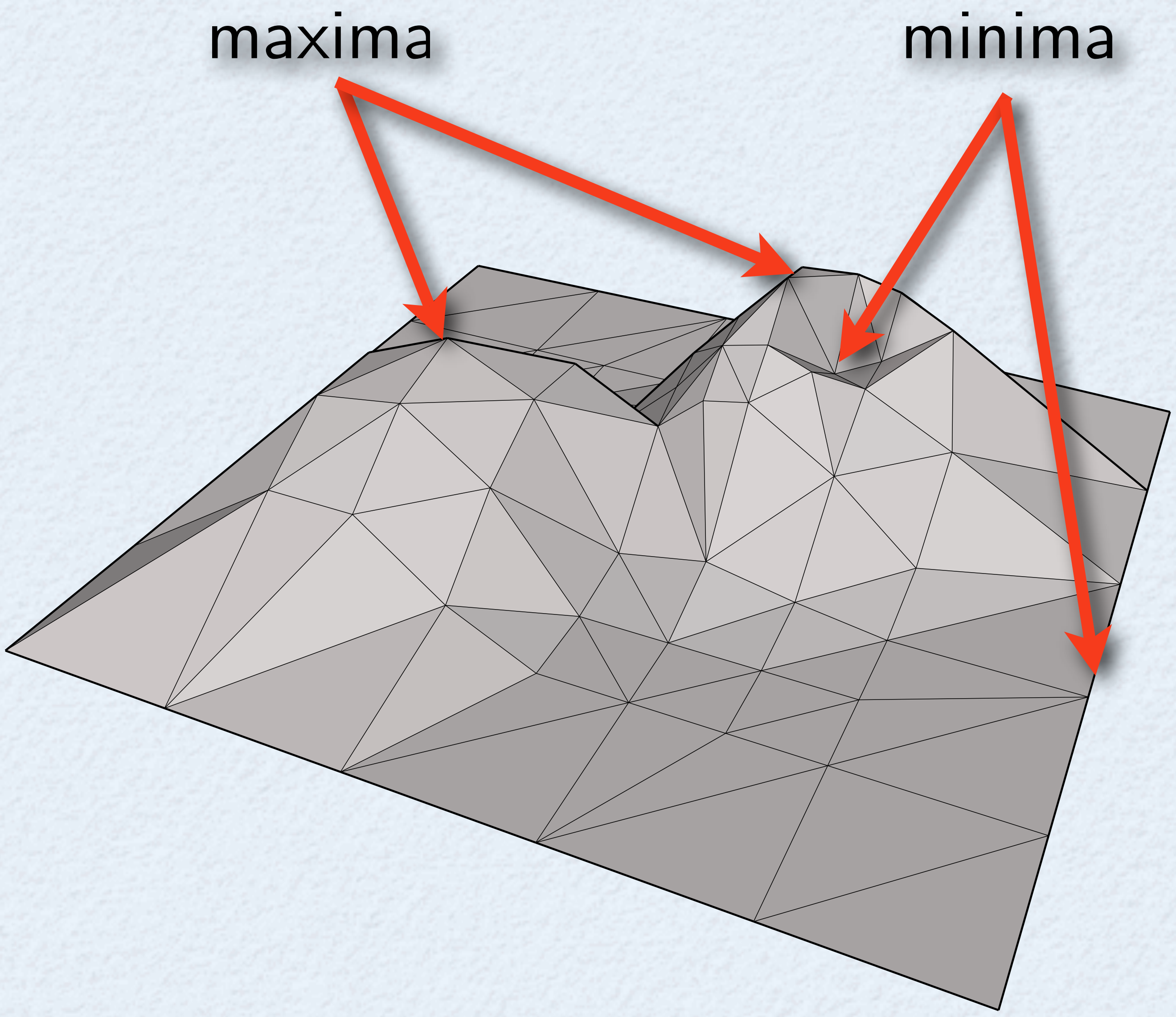
Critical Points of a Terrain



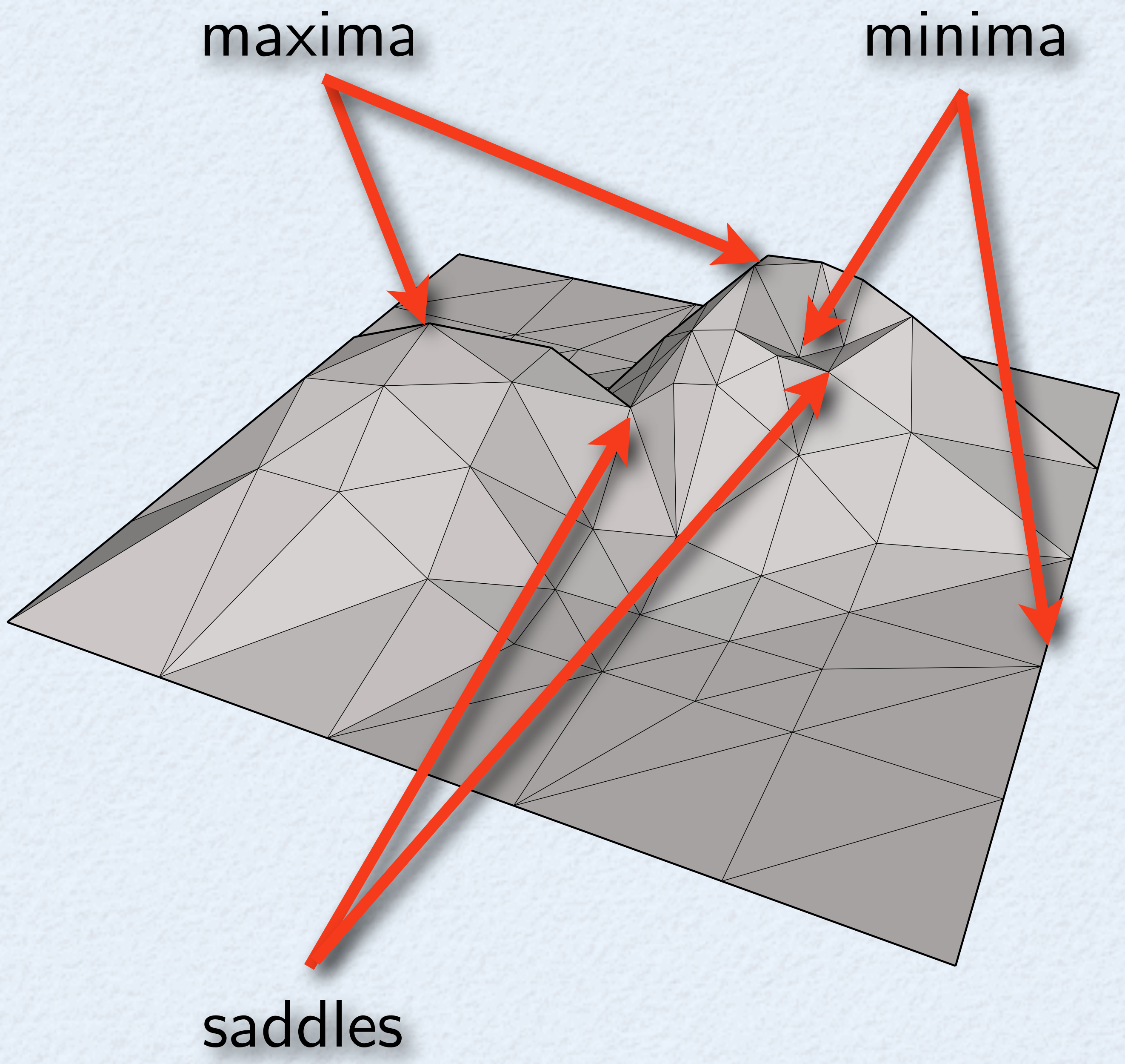
Critical Points of a Terrain



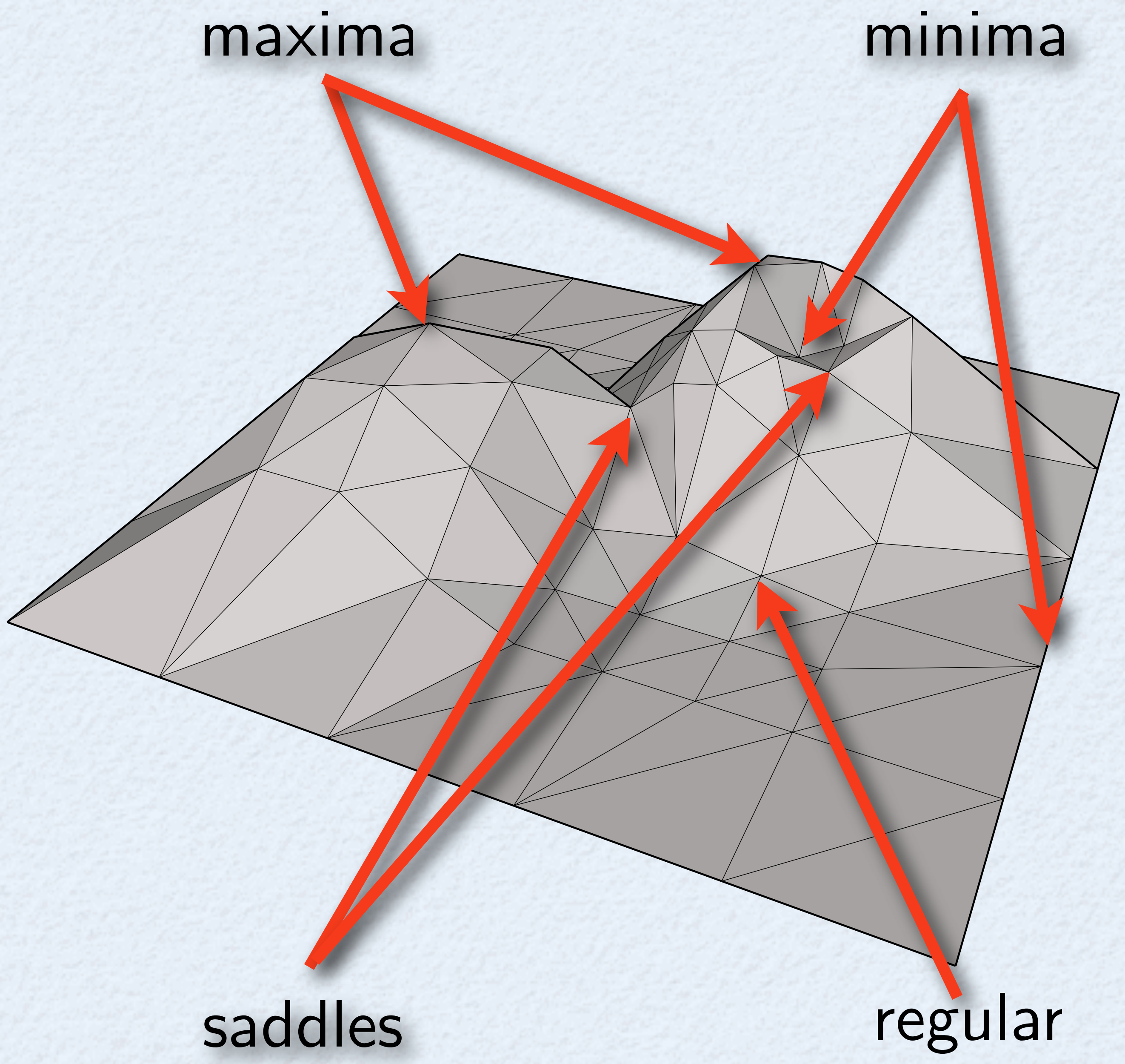
Critical Points of a Terrain



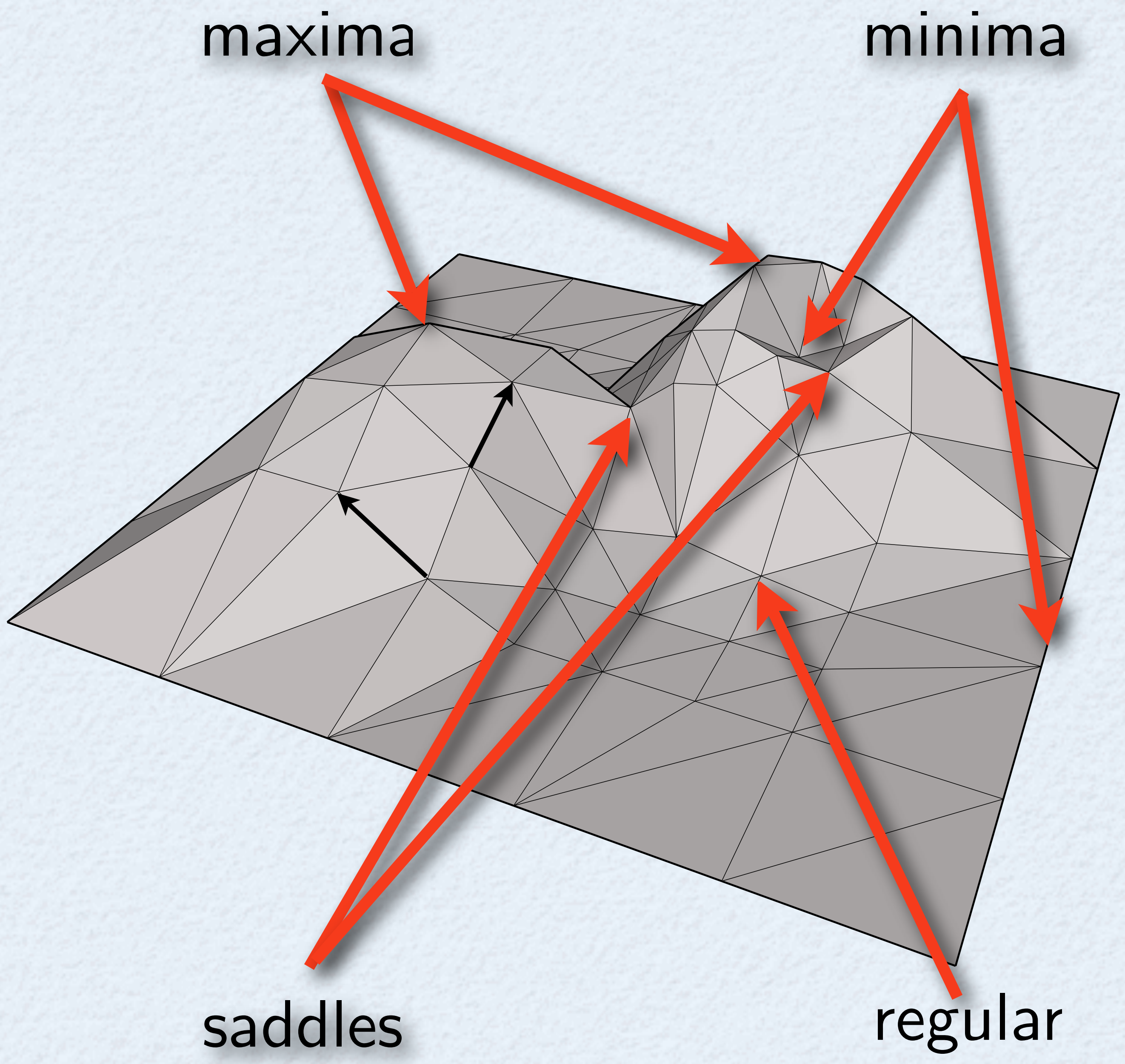
Critical Points of a Terrain



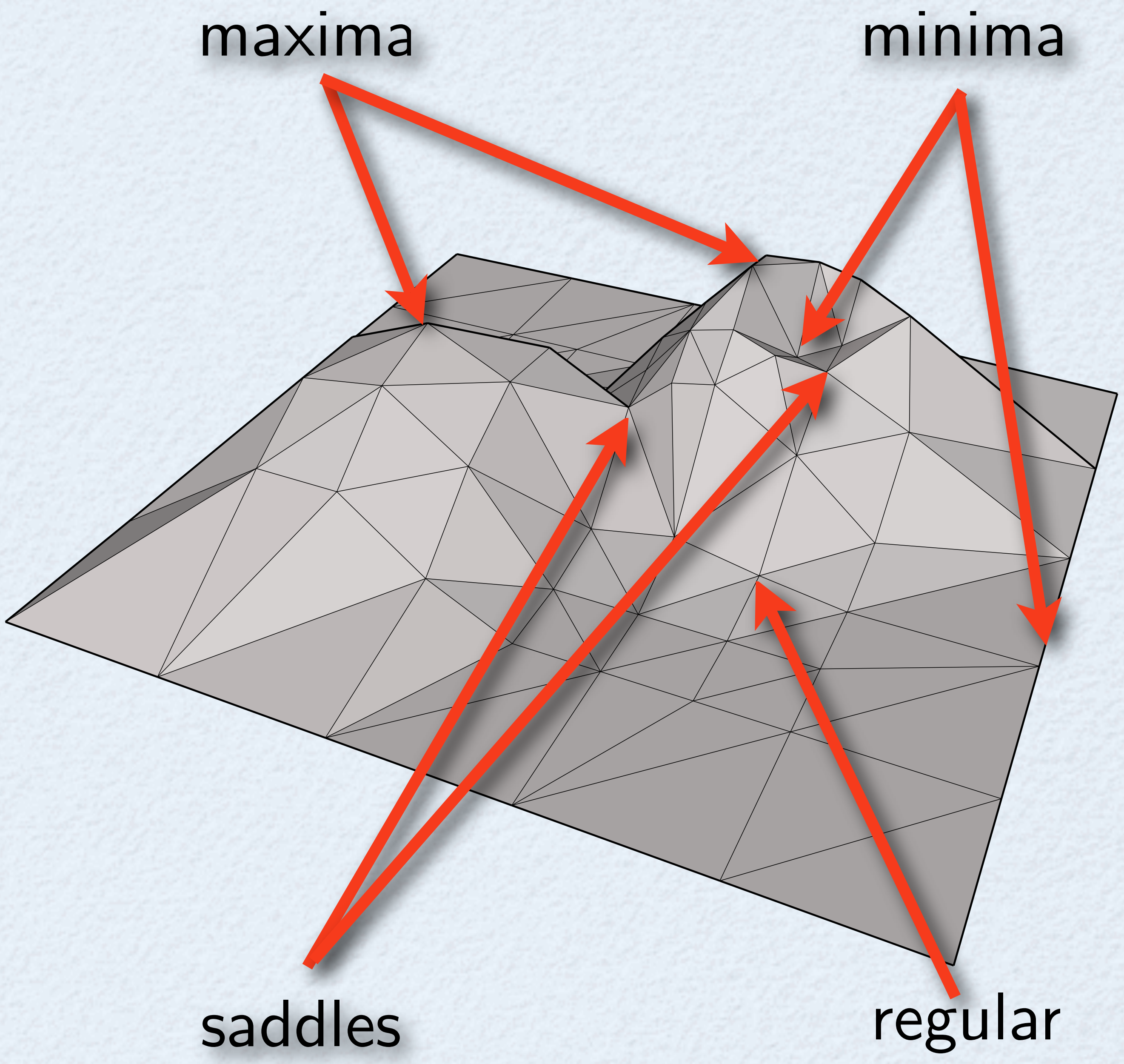
Critical Points of a Terrain



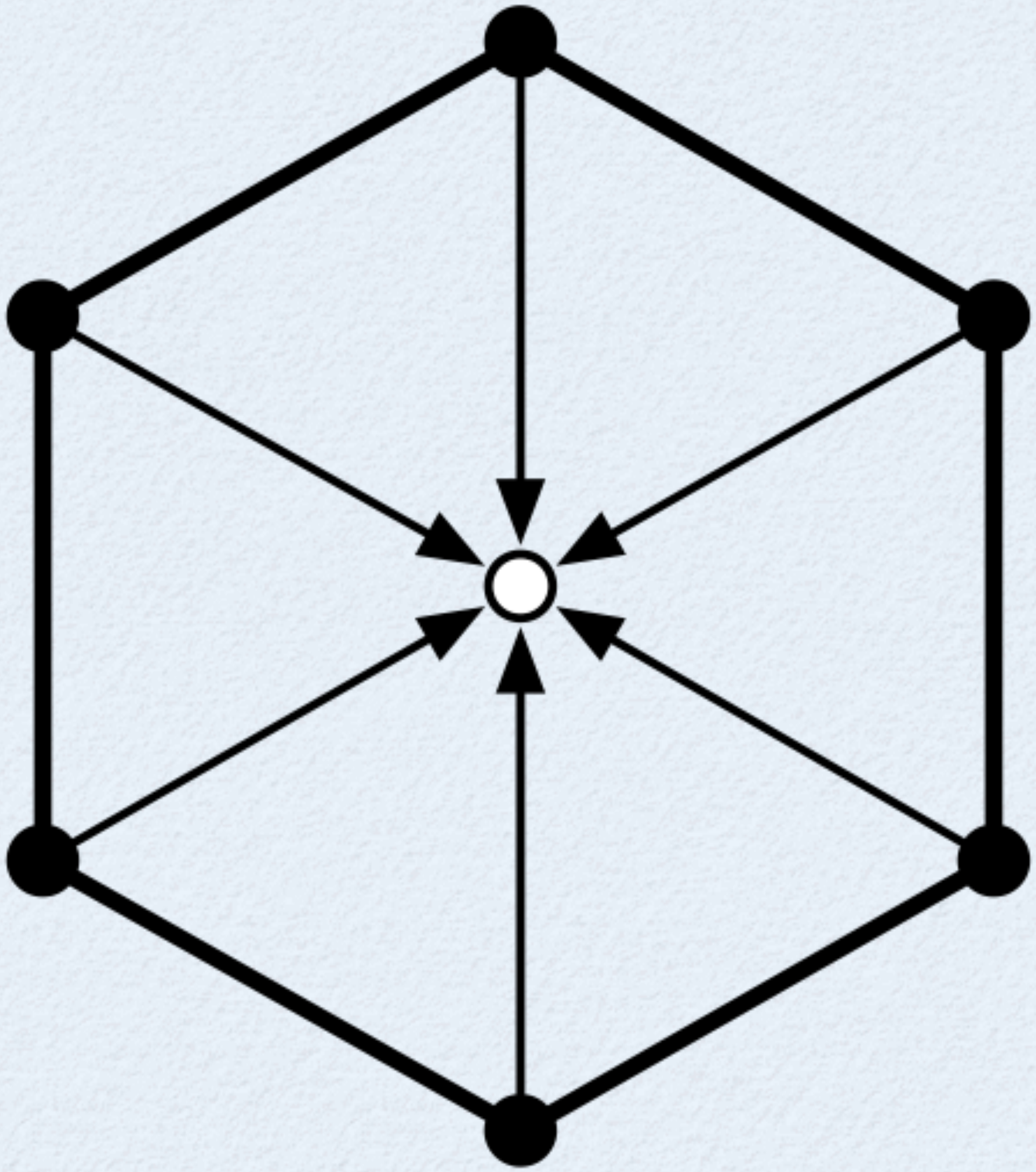
Critical Points of a Terrain

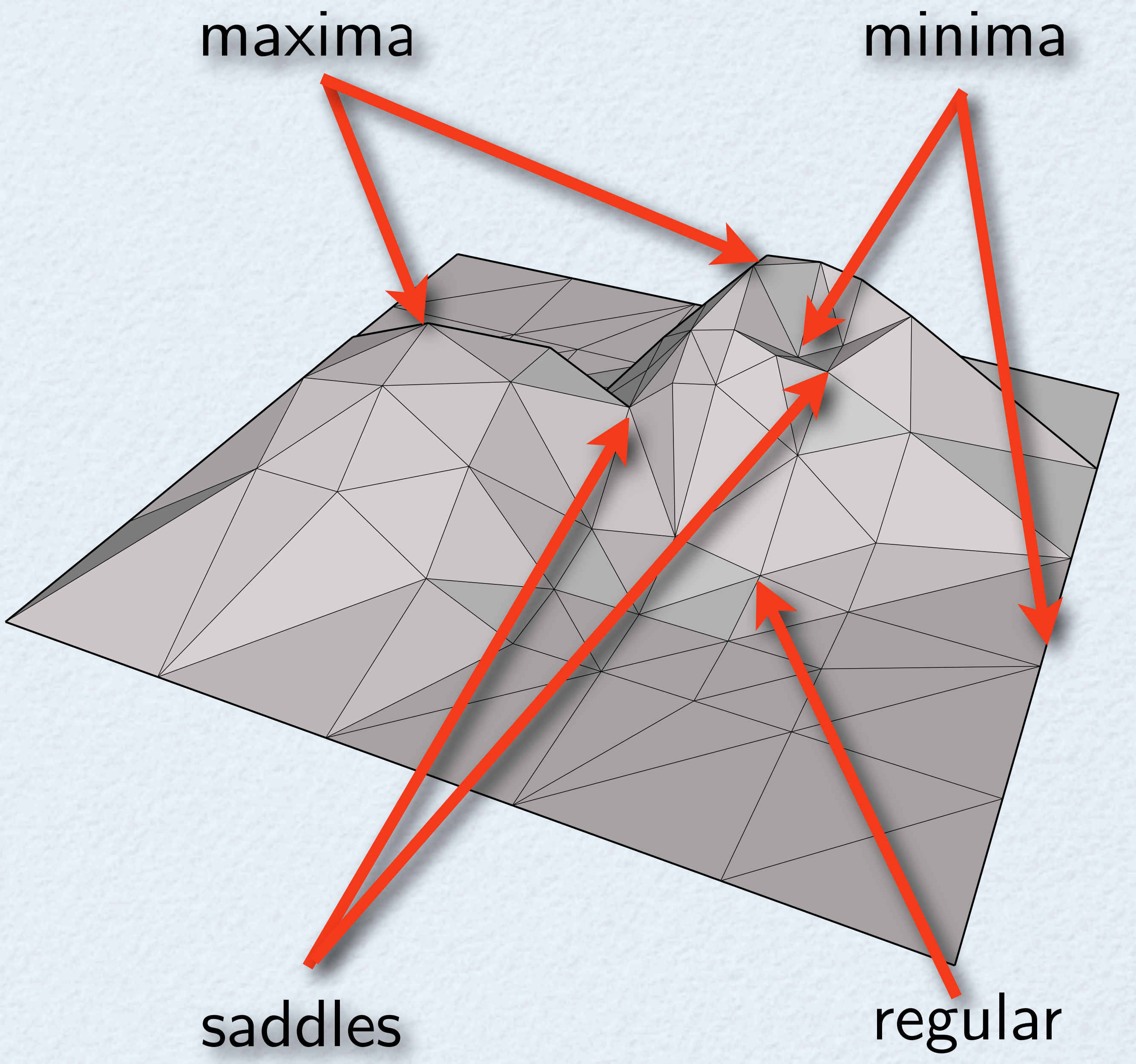


Critical Points of a Terrain

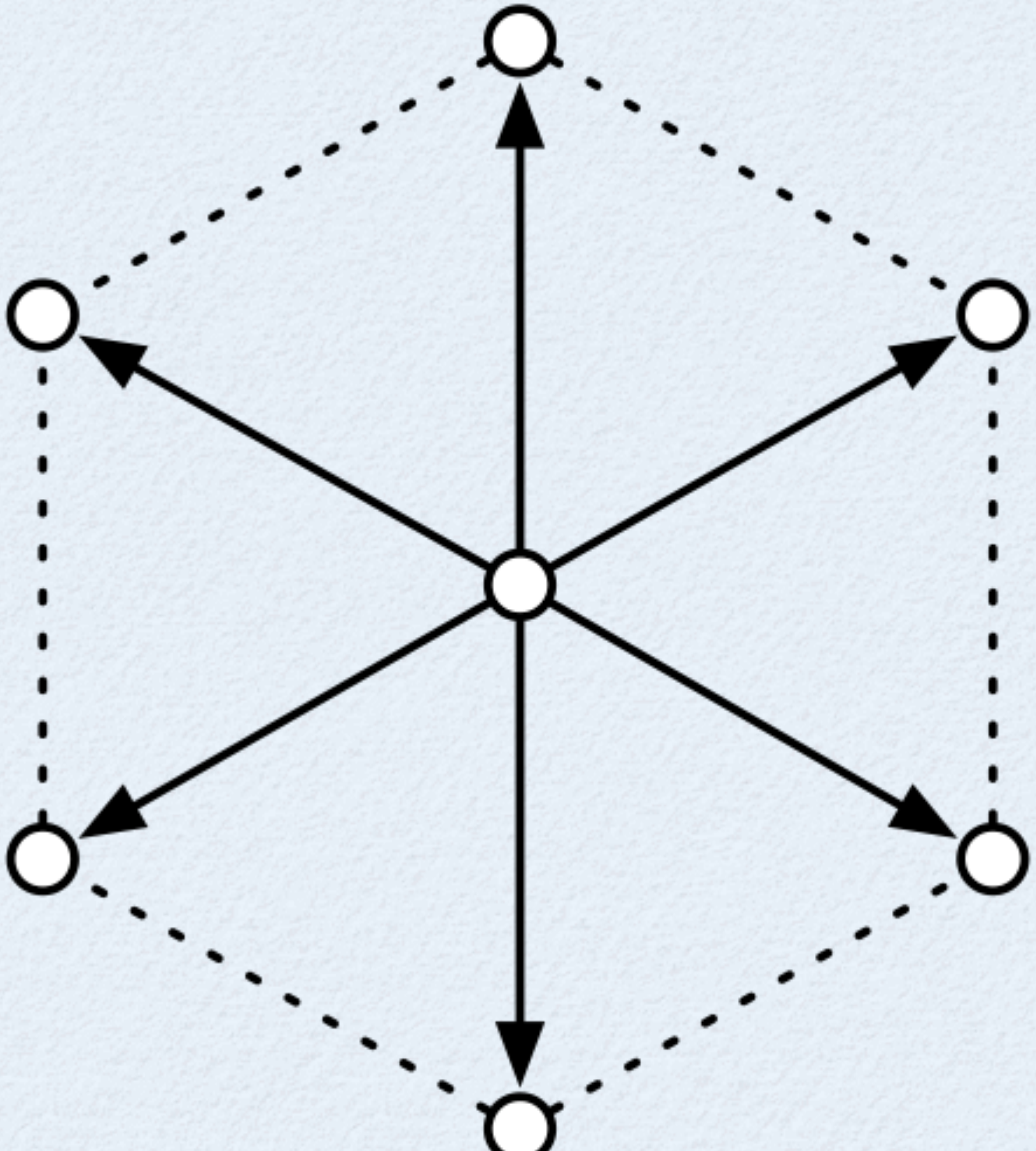


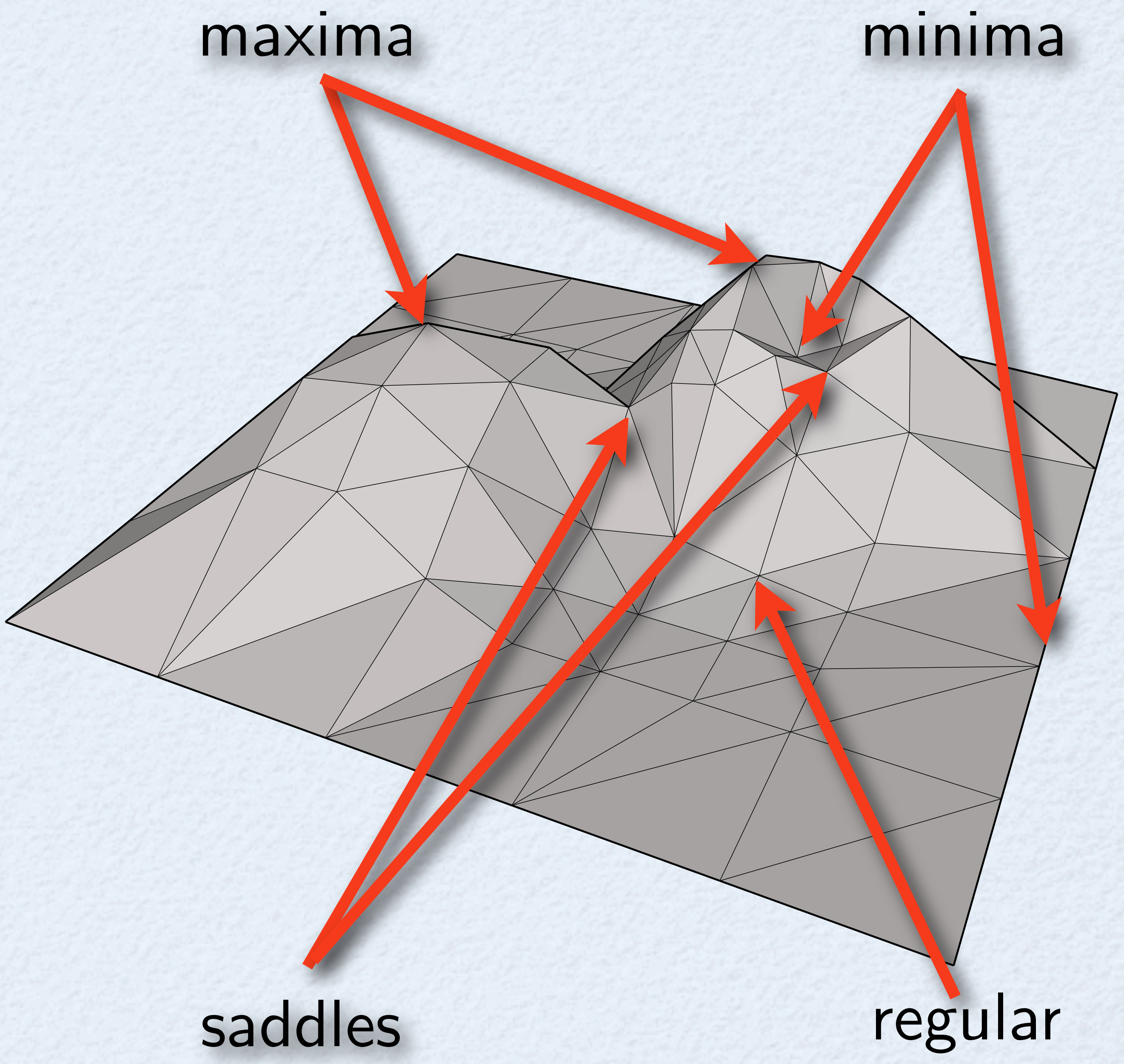
Critical Points of a Terrain



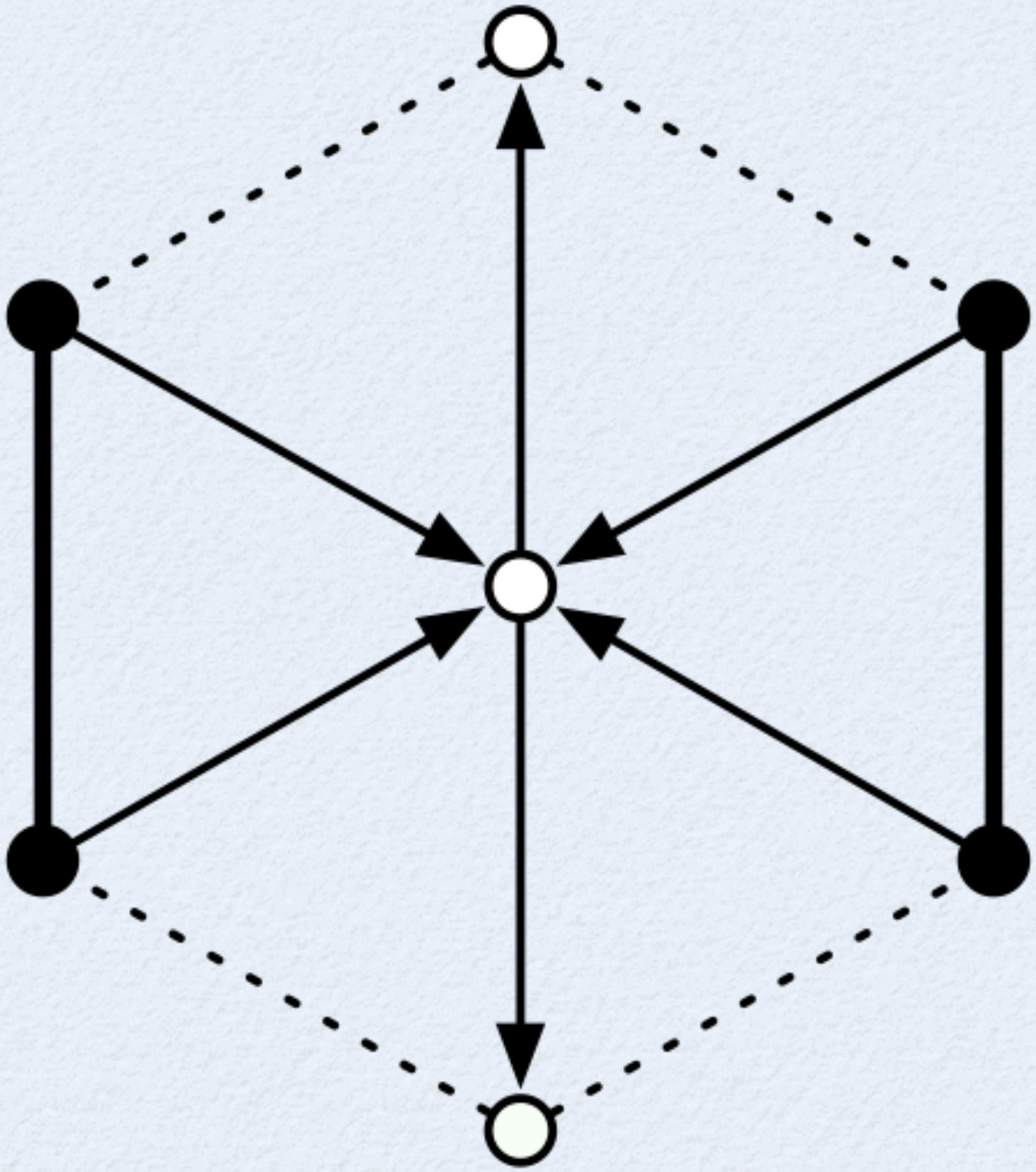


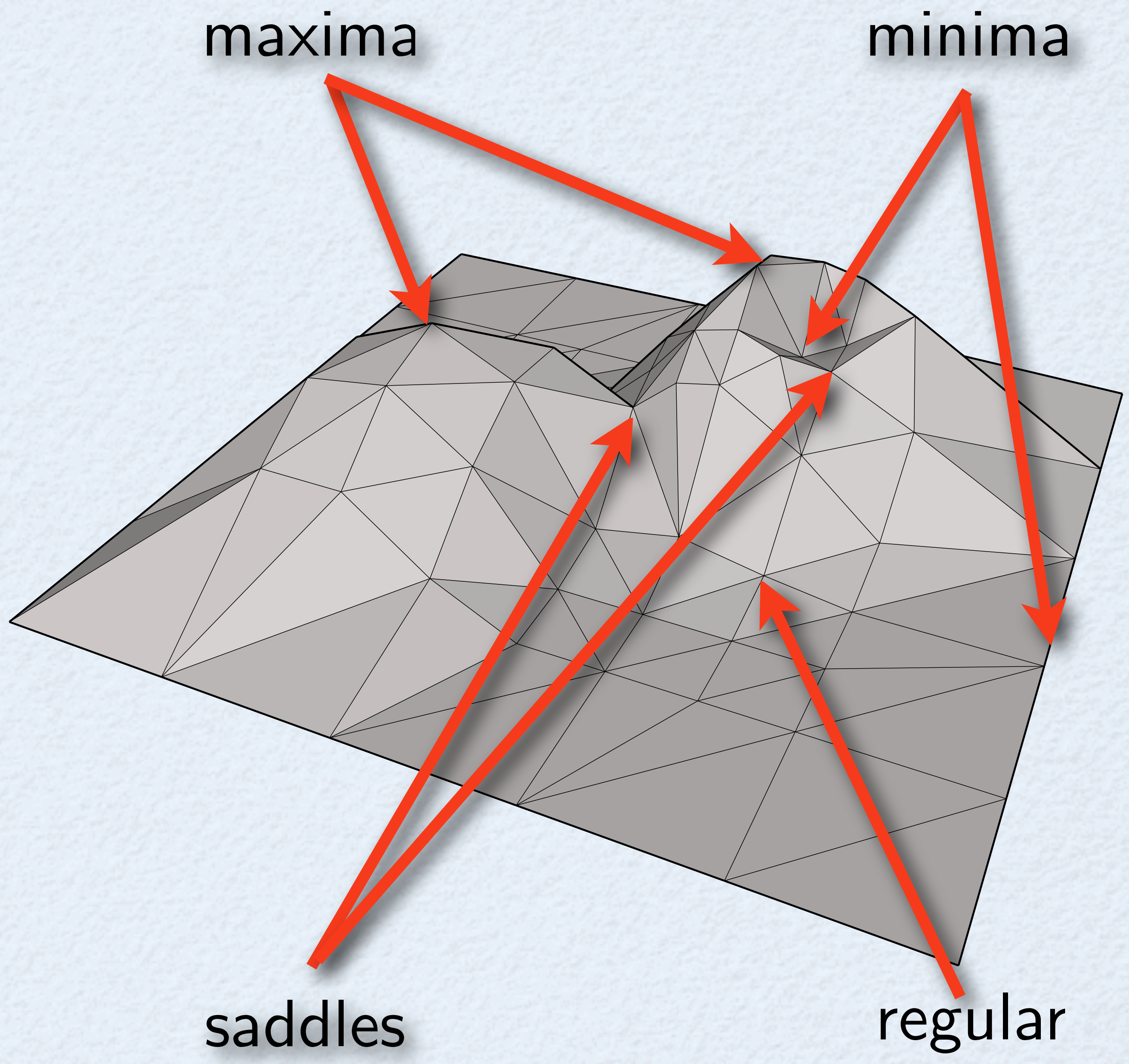
Critical Points of a Terrain



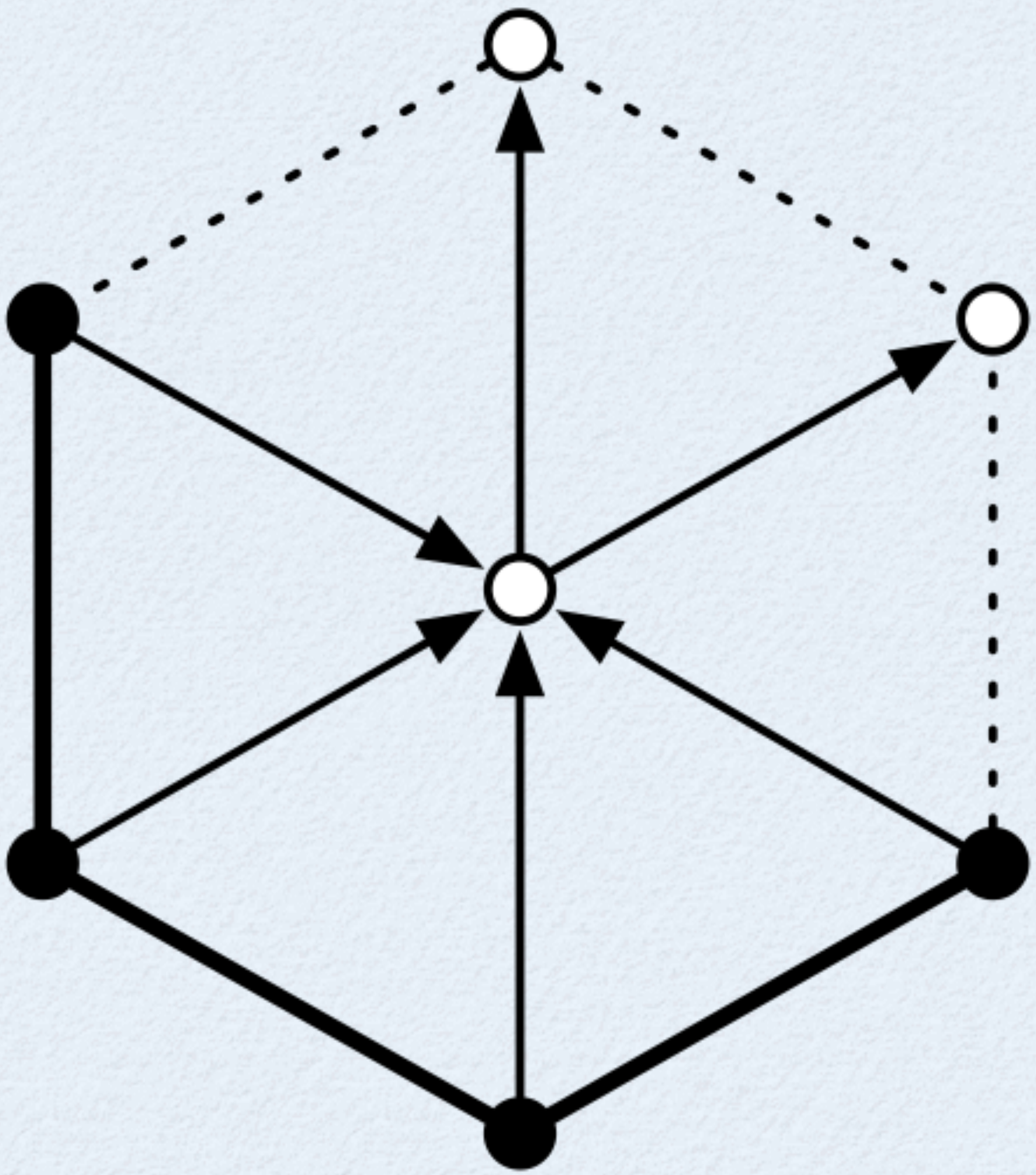


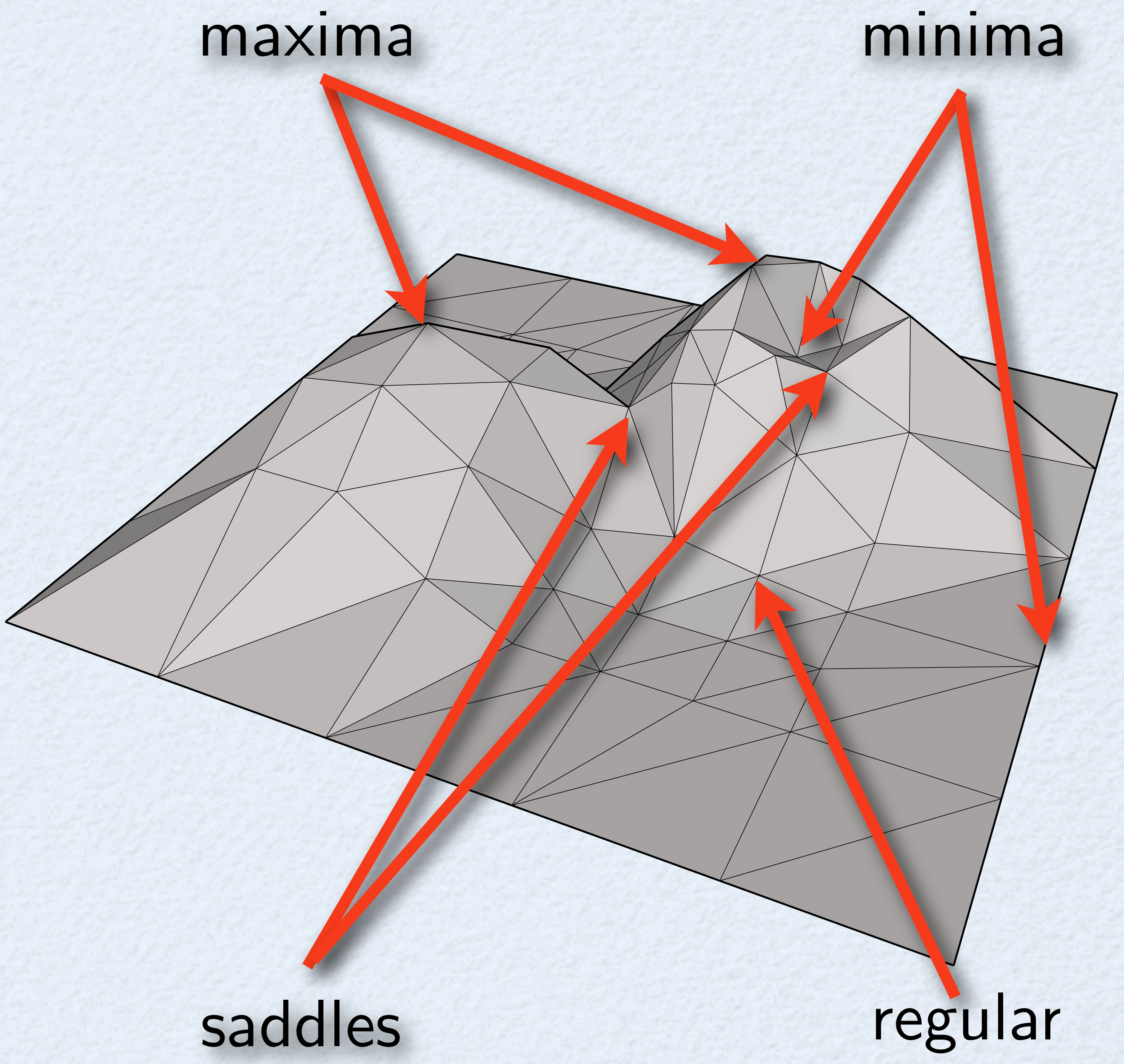
Critical Points of a Terrain





Critical Points of a Terrain



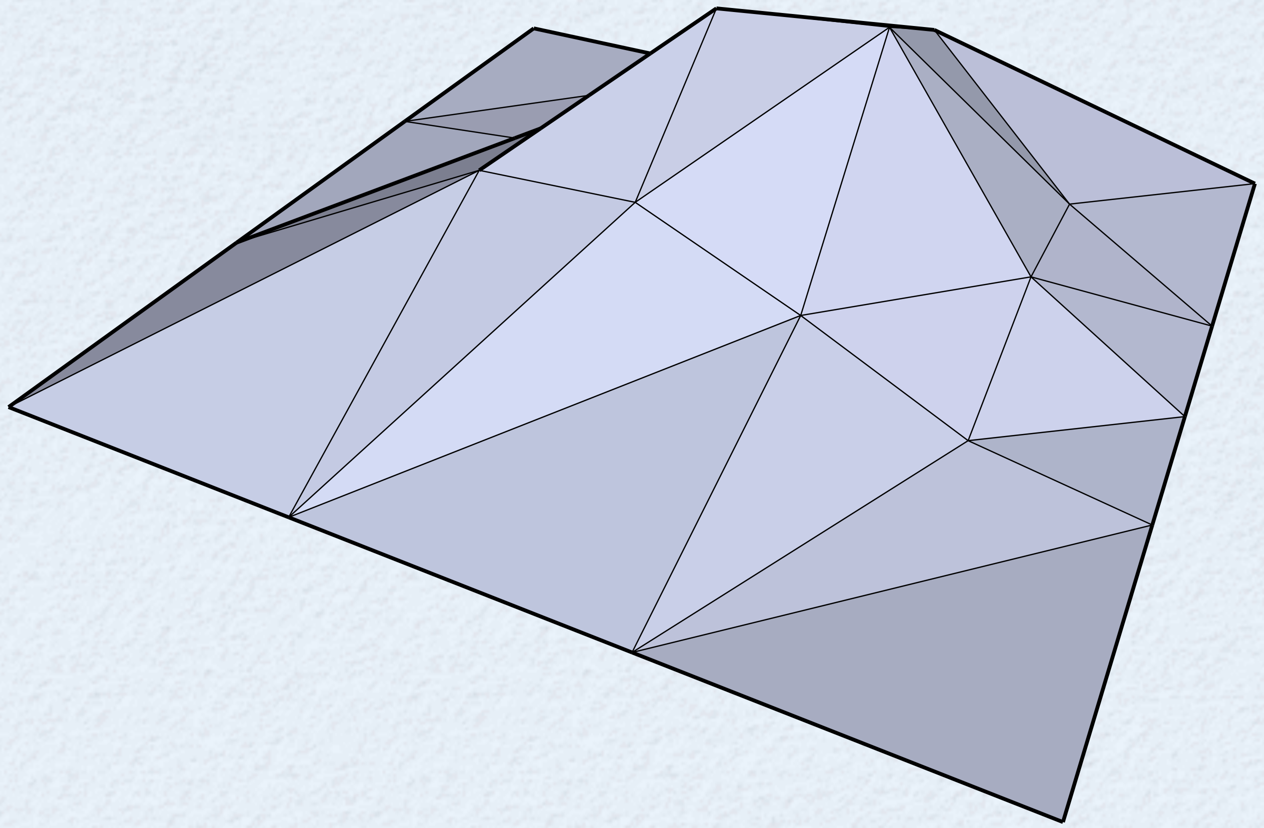


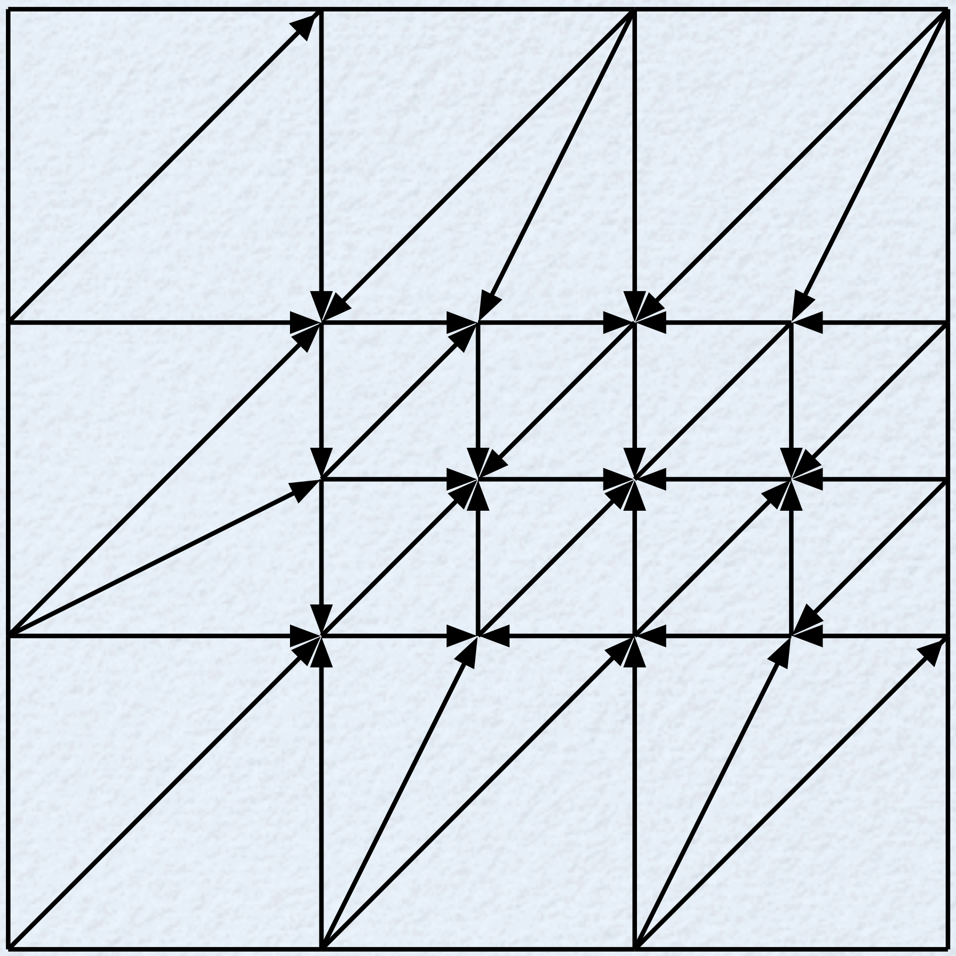
Level Ordering of Elementary Terrains

An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

Level Ordering of Elementary Terrains

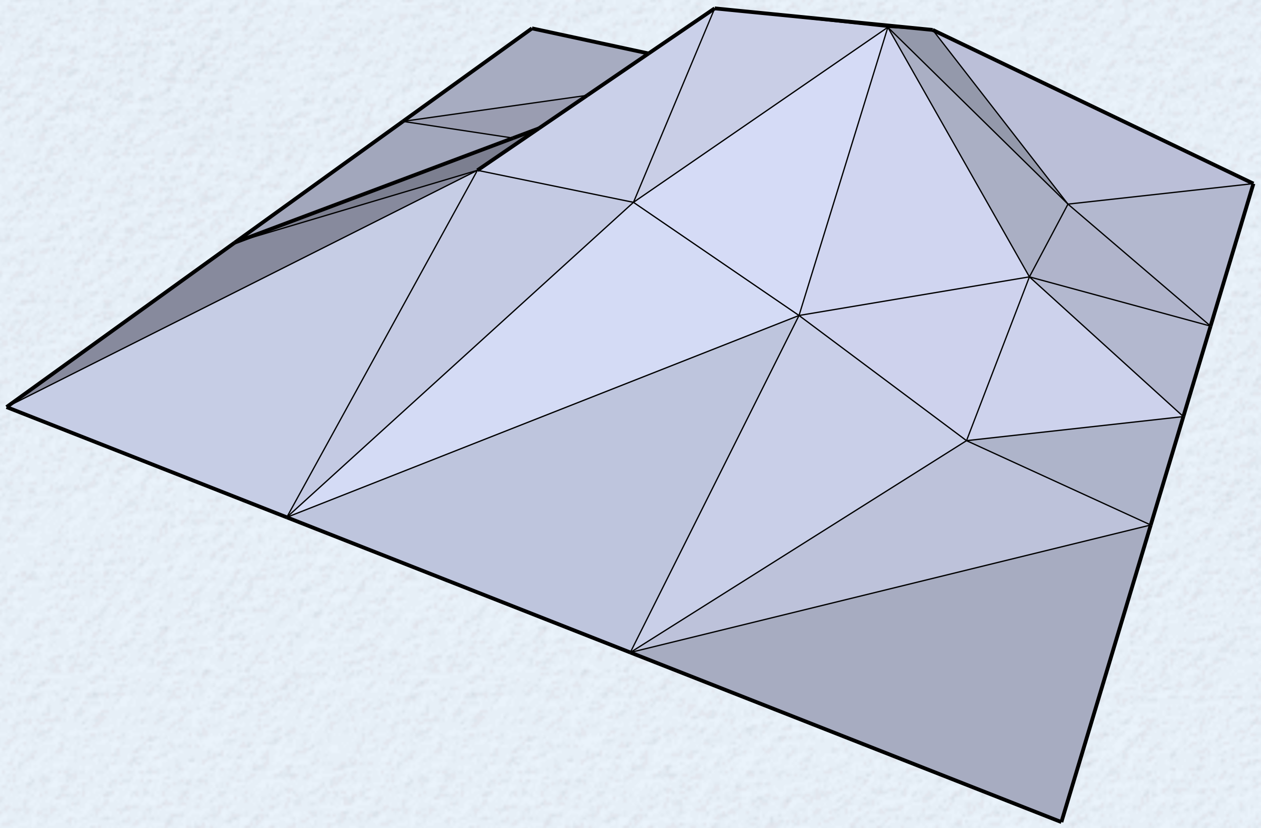
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

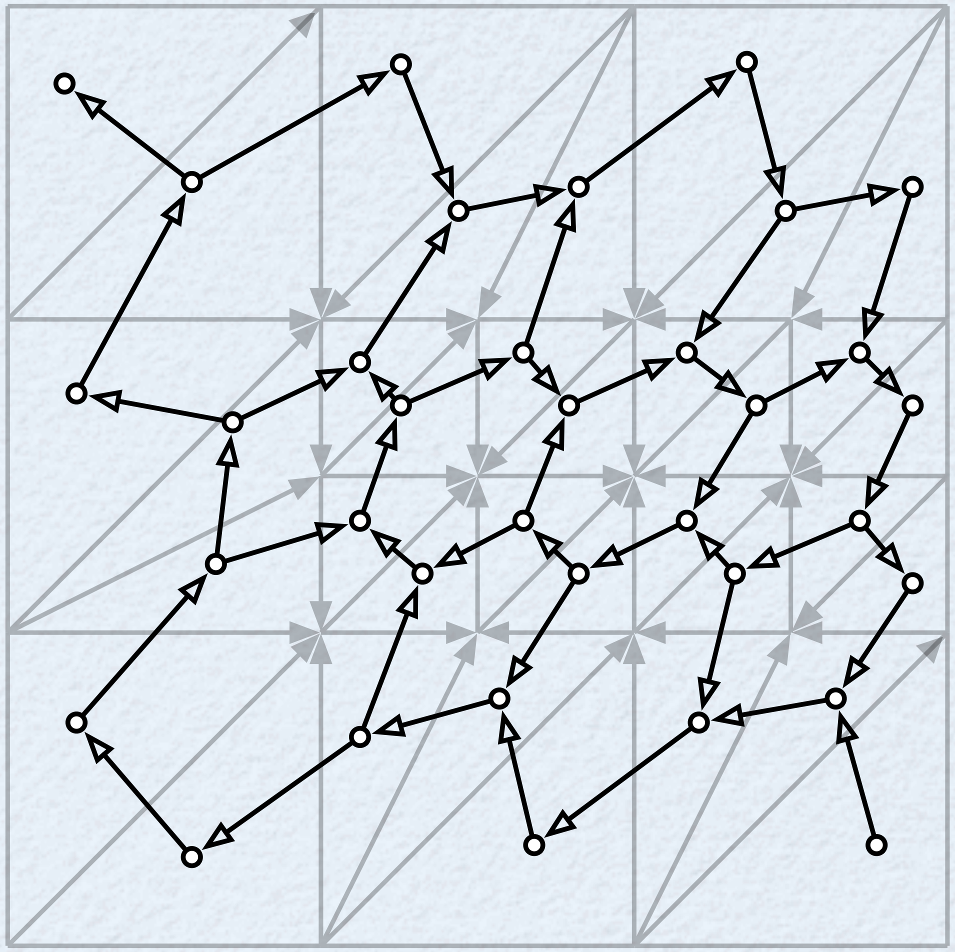




Level Ordering of Elementary Terrains

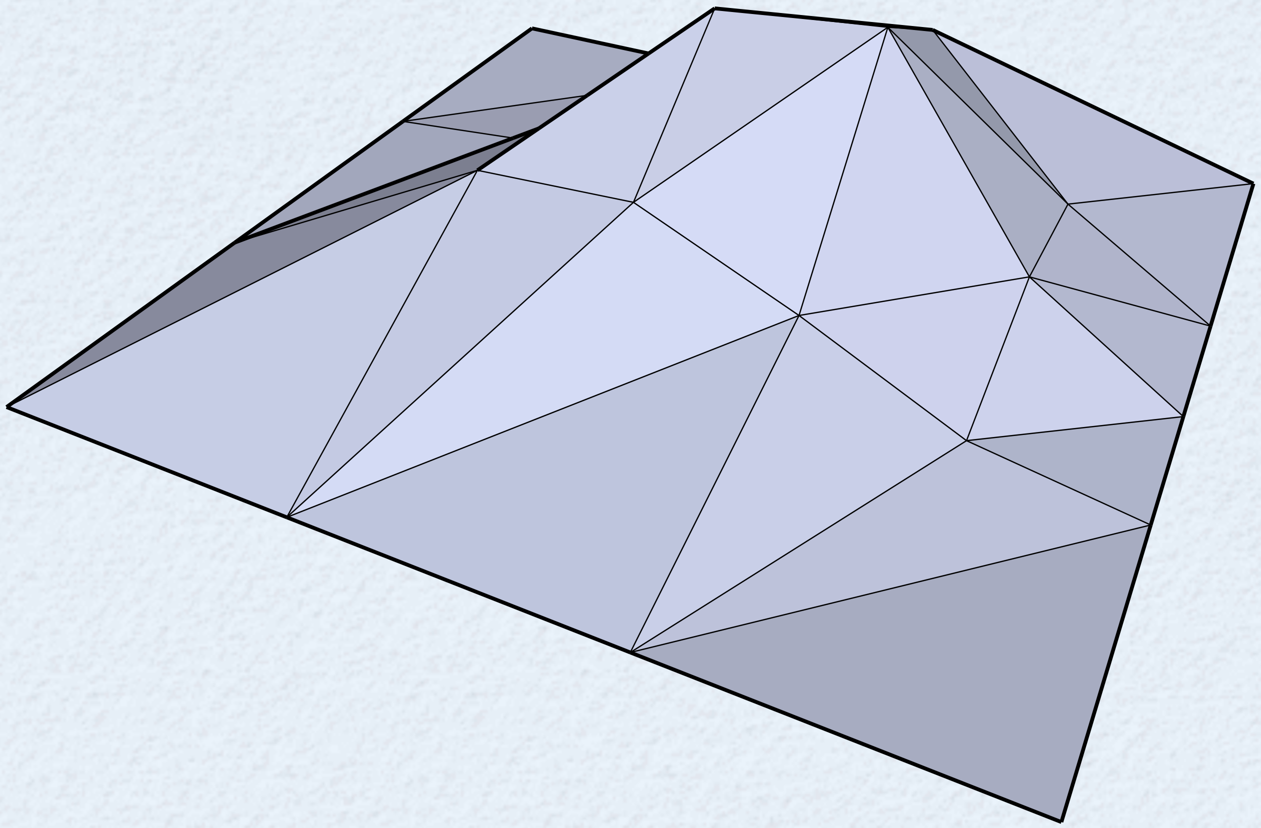
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

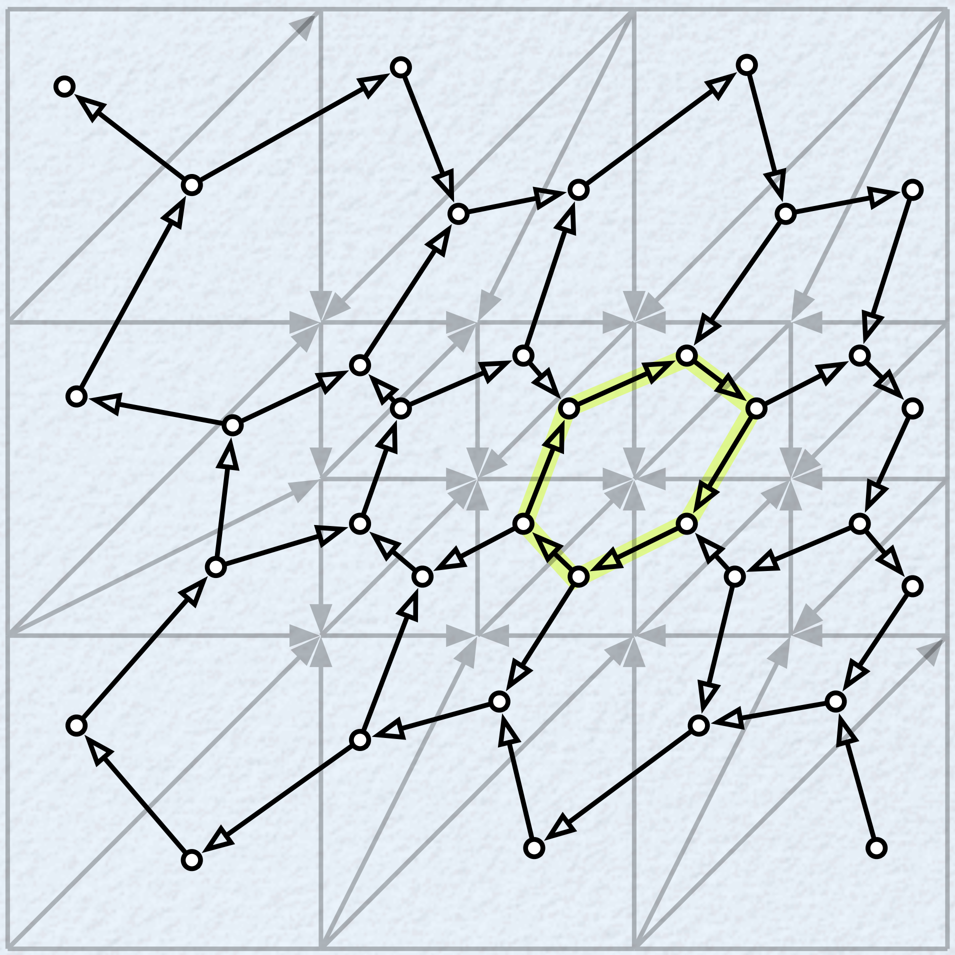




Level Ordering of Elementary Terrains

An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

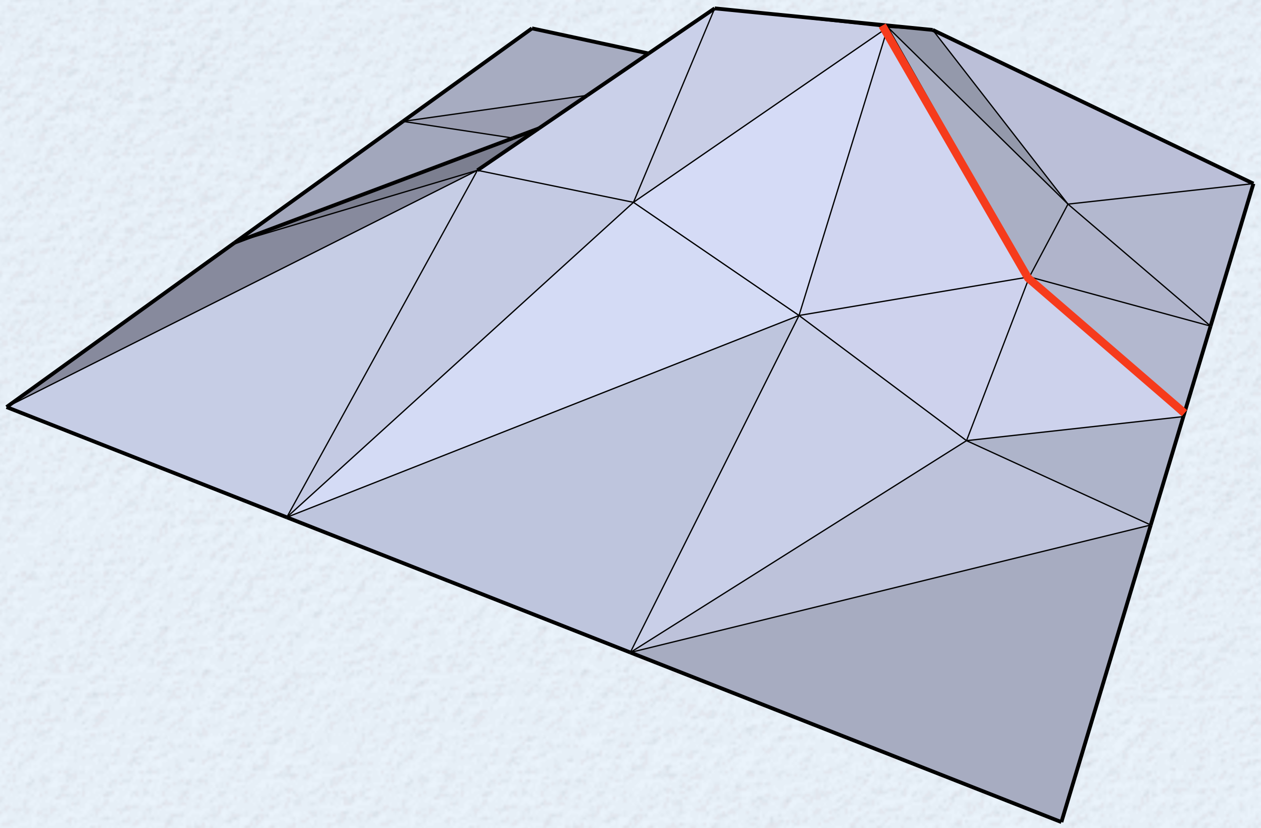


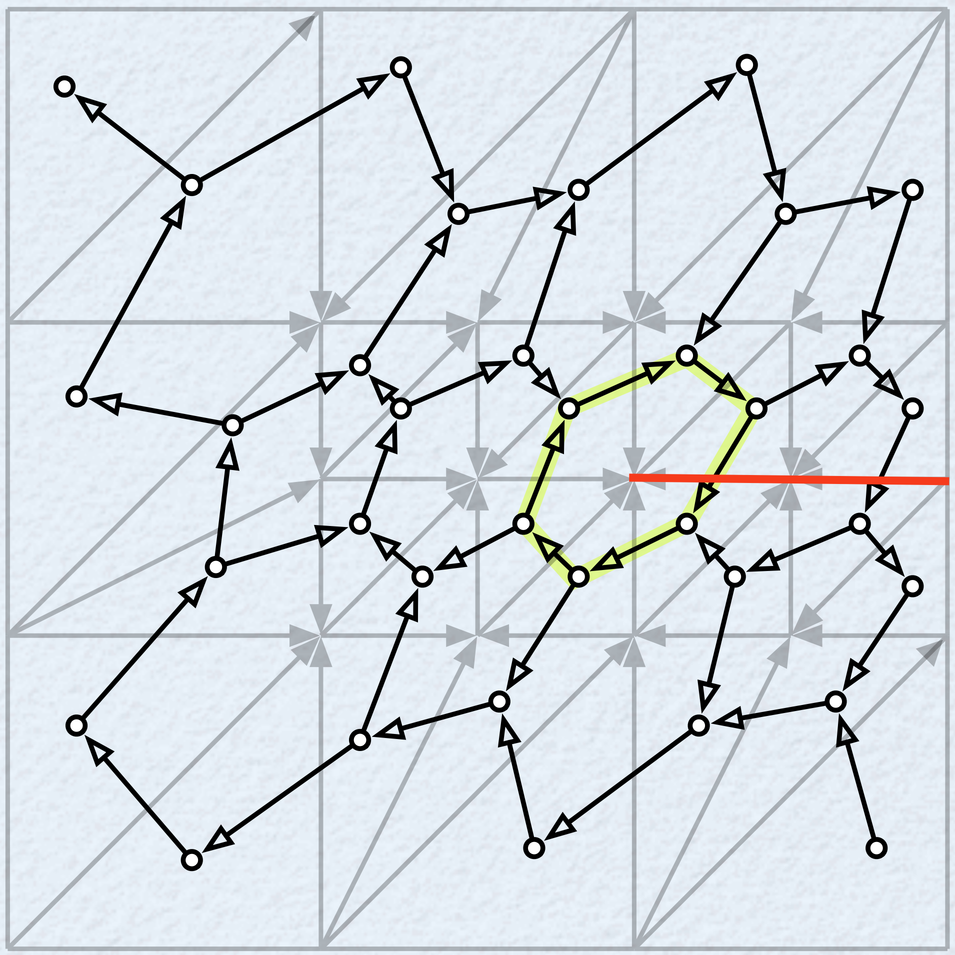


Level Ordering of Elementary Terrains

An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

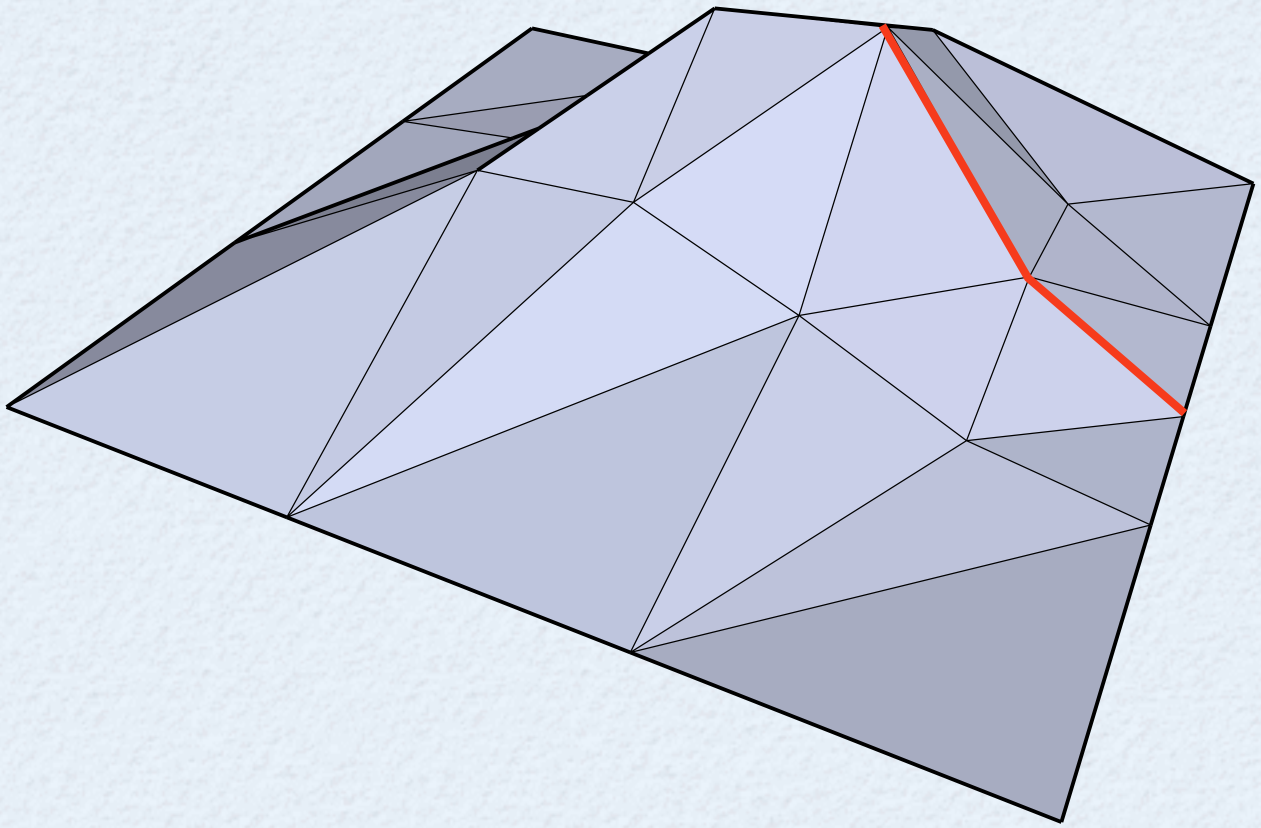
Take a monotone min (bd) to max path P and delete its dual from $\mathbb{M}^*$.

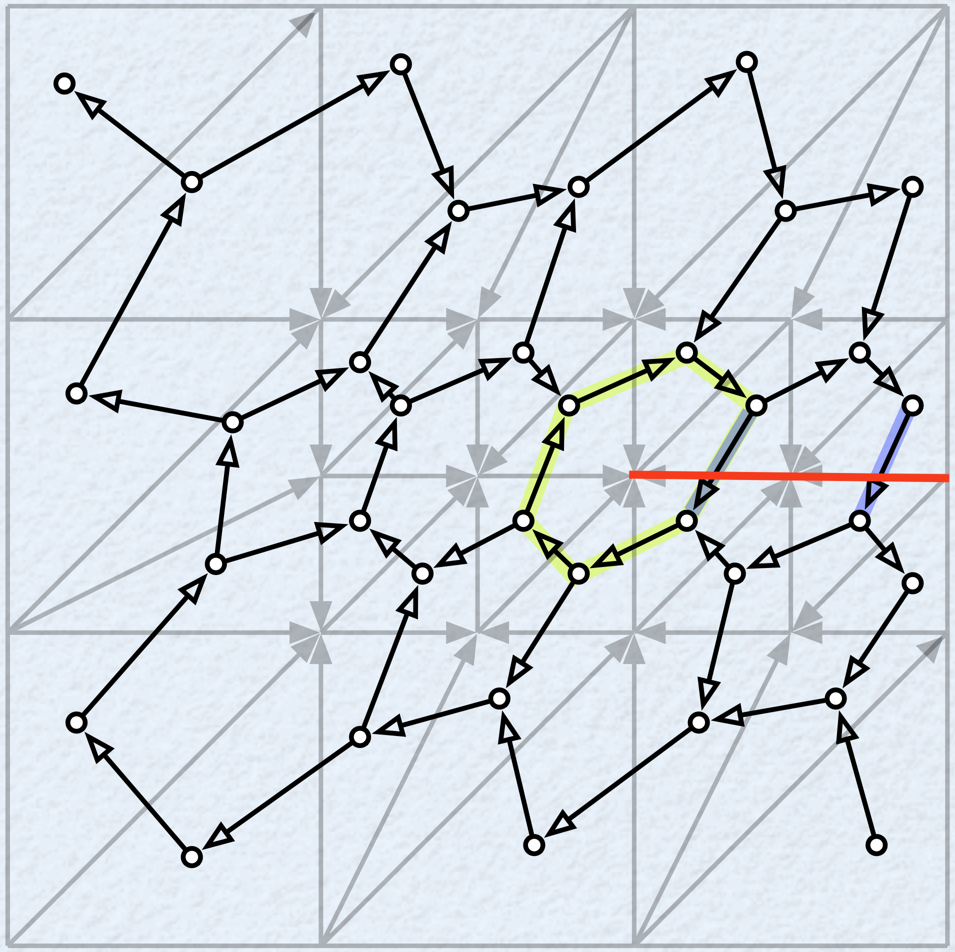




Level Ordering of Elementary Terrains

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 $\mathbb{M}^*$.

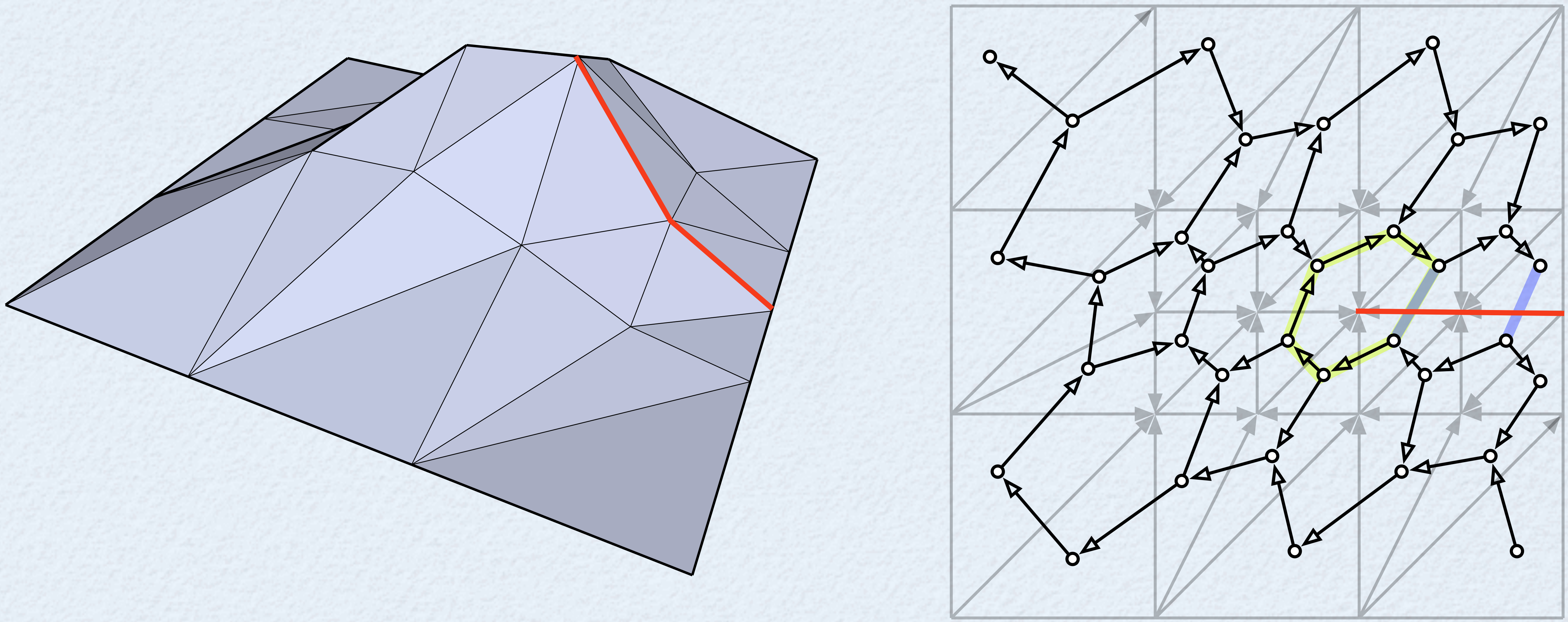




Level Ordering of Elementary Terrains

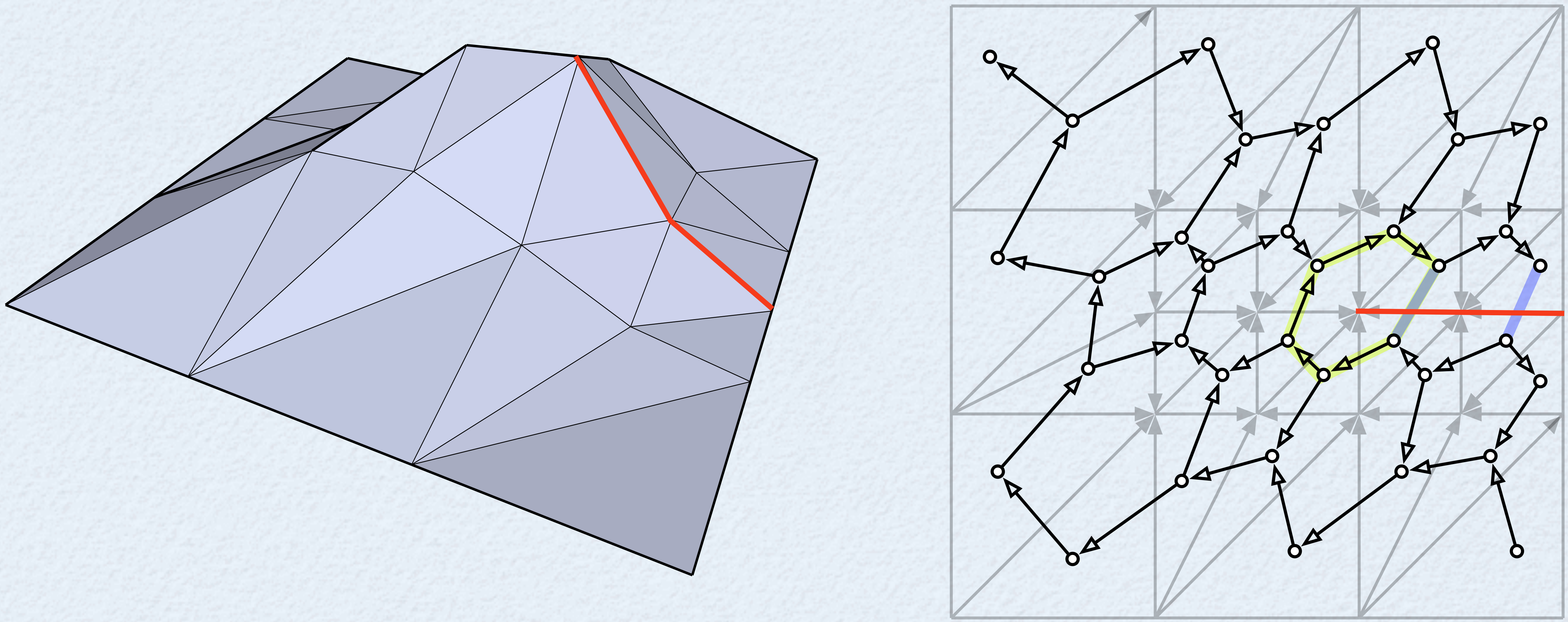
An elementary terrain has no saddles; thus 1 max and 1 min (boundary).

Take a monotone min (bd) to max path P and delete its dual from $\mathbb{M}^*$.



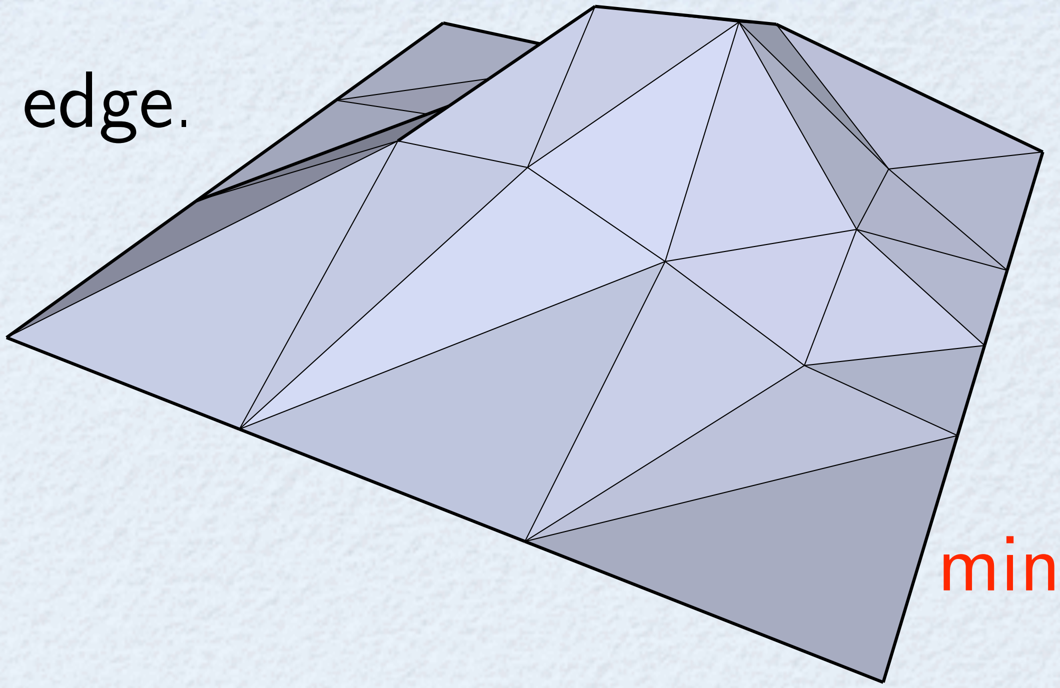
Level Ordering of Elementary Terrains

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 $\mathbb{M}^*$.



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

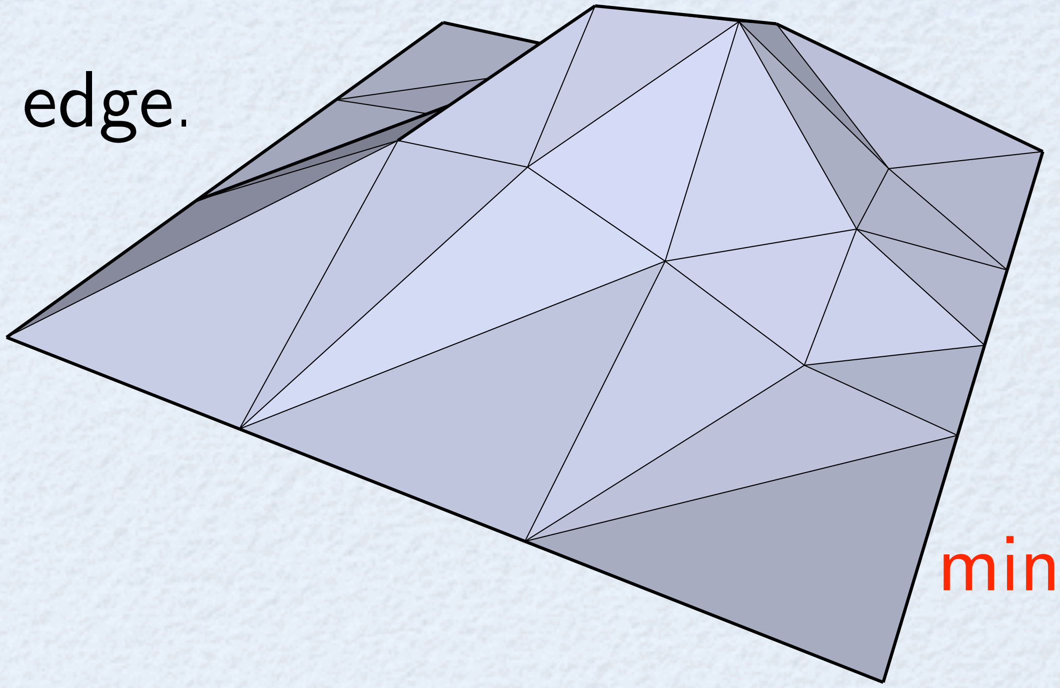
Partial Order on Triangles



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

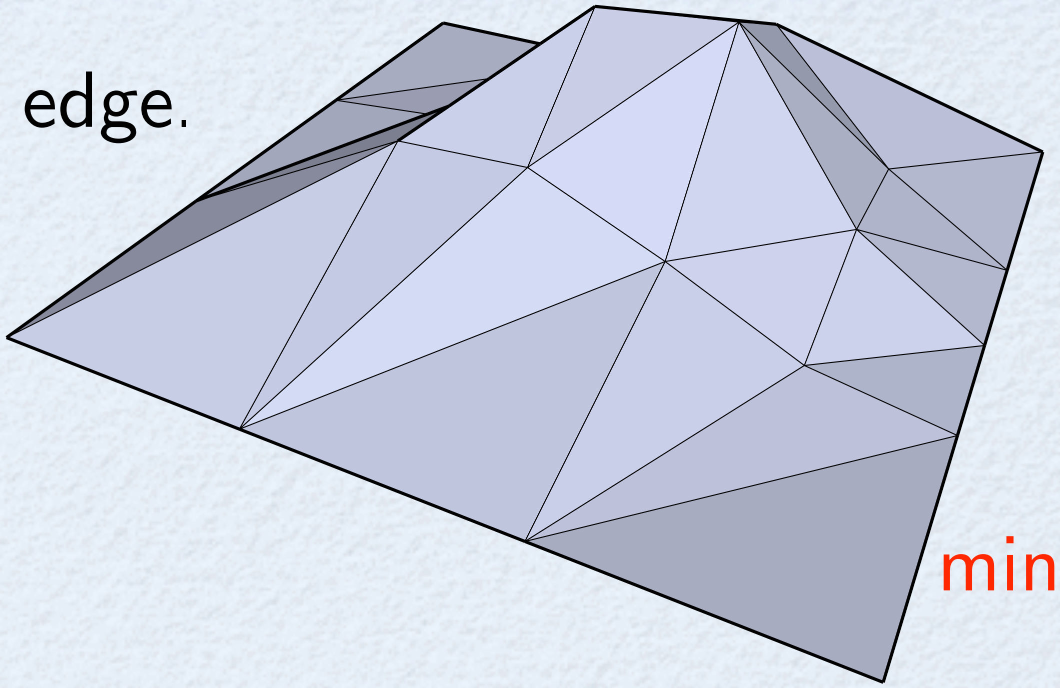
Partial Order on Triangles

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



1. max vertex is reachable from every vertex.

Partial Order on Triangles



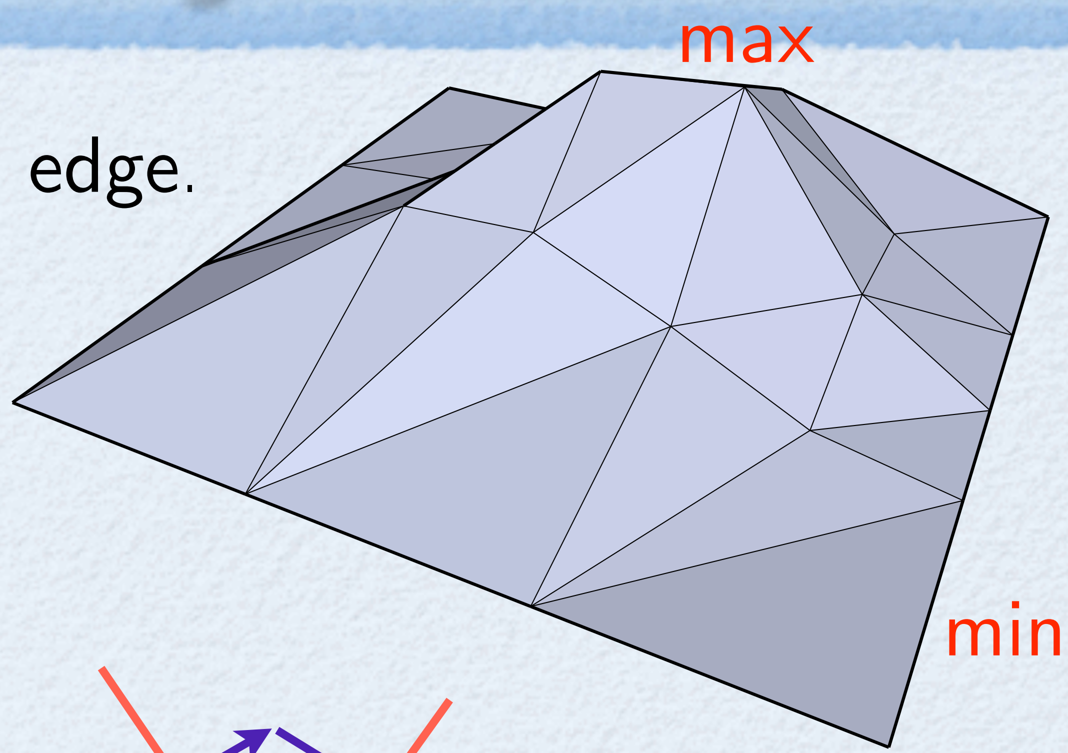
Lemma. Every cycle of $\mathbb{M}^*$ loses precisely one edge.

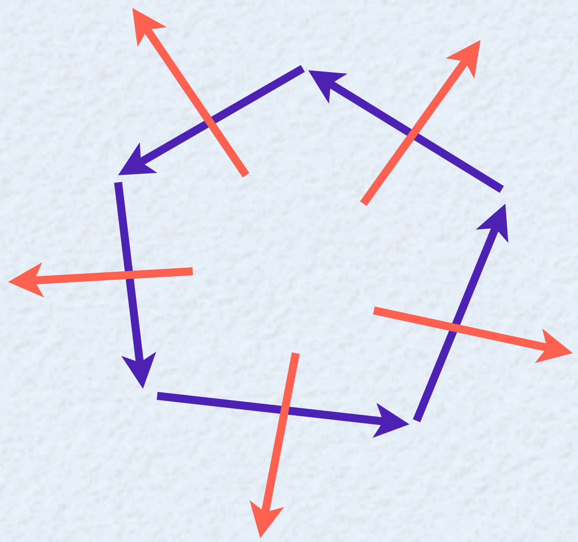
1. max vertex is reachable from every vertex.
2. Every vertex is reachable from min (bd).

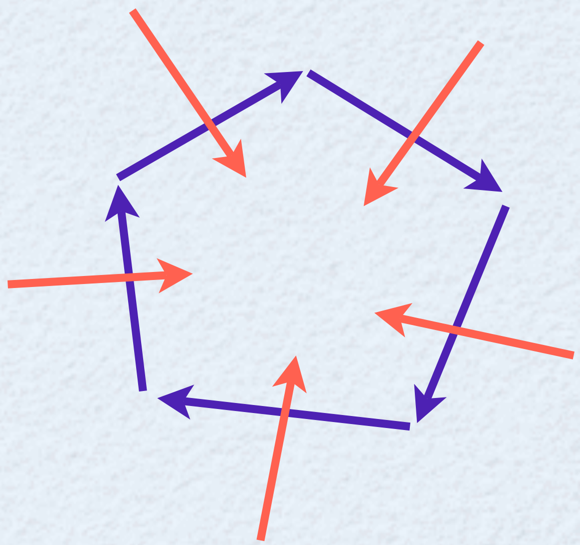
Partial Order on Triangles

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

1. max vertex is reachable from every vertex.
2. Every vertex is reachable from min (bd).

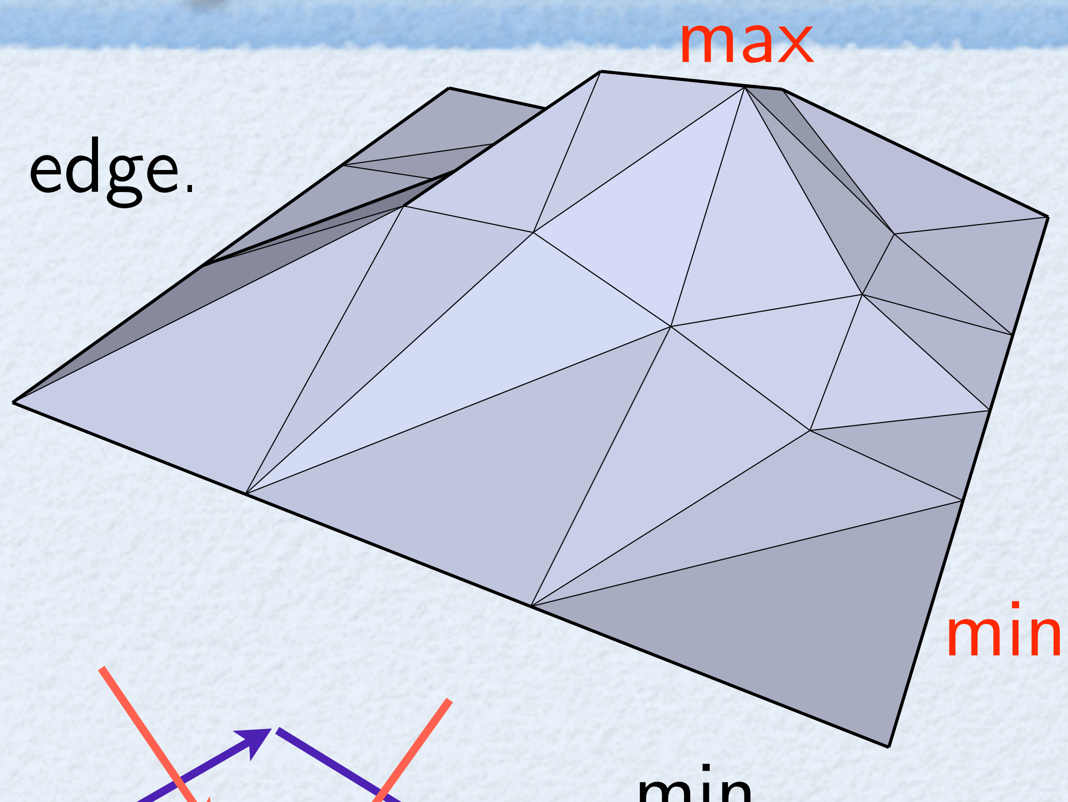




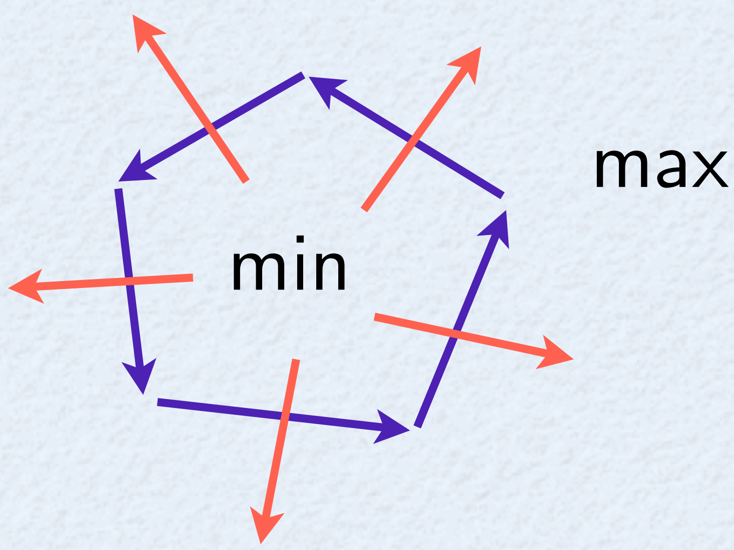


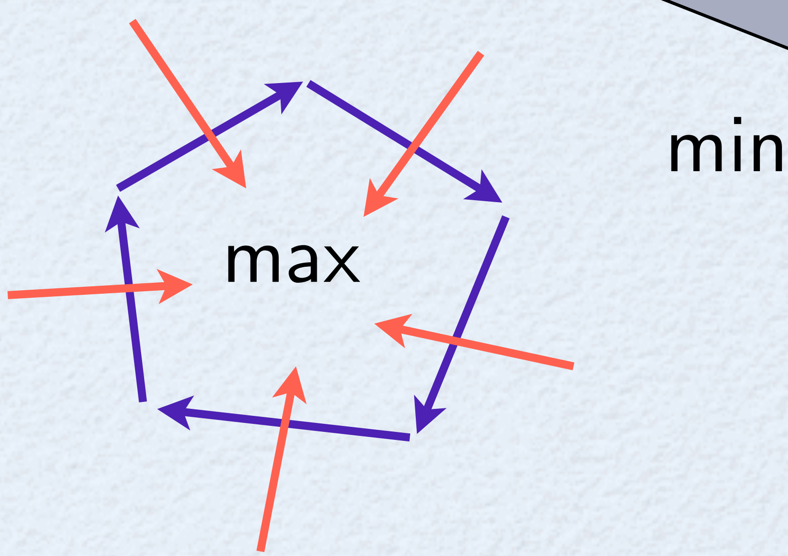
Partial Order on Triangles

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

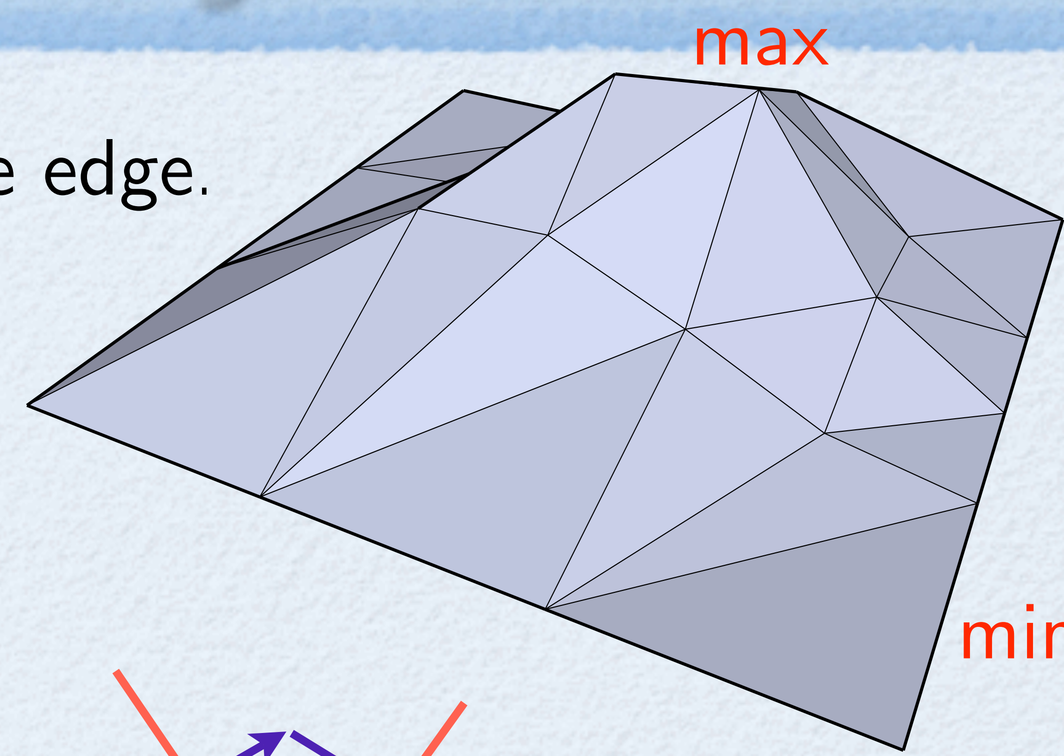


1. max vertex is reachable from every vertex.
2. Every vertex is reachable from min (bd).



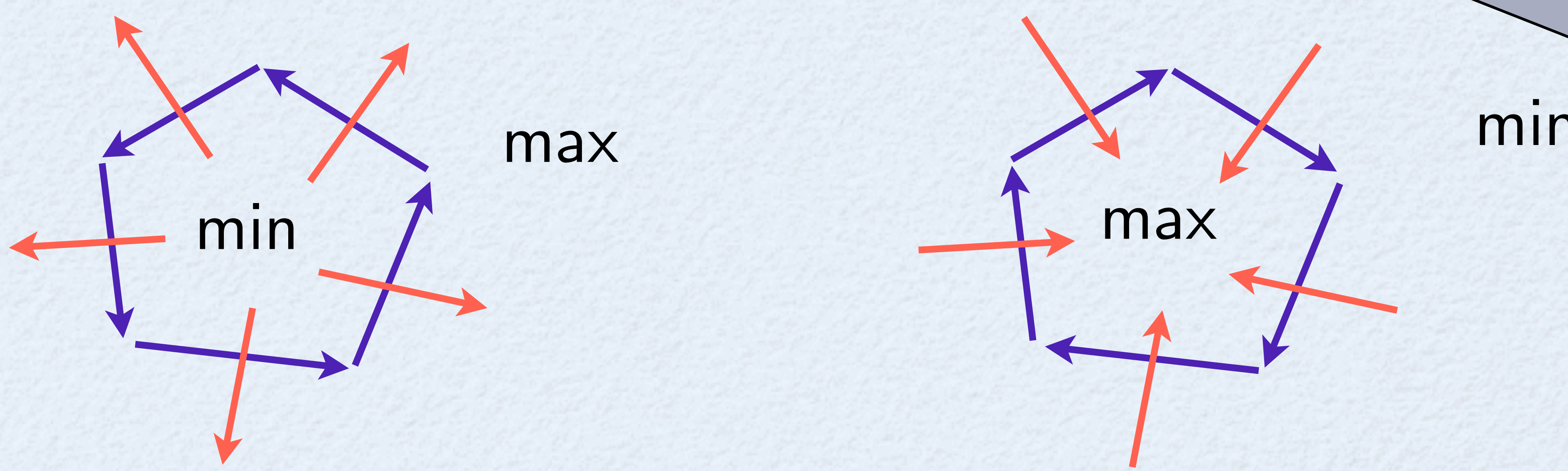


Partial Order on Triangles



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

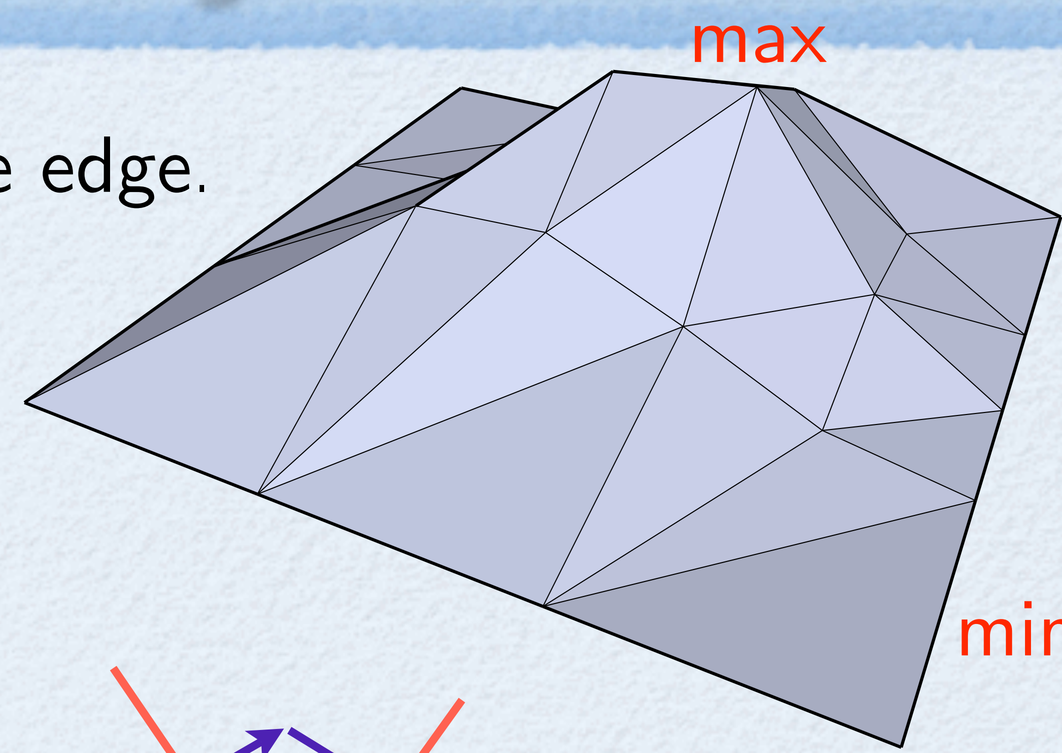
1. max vertex is reachable from every vertex.
2. Every vertex is reachable from min (bd).



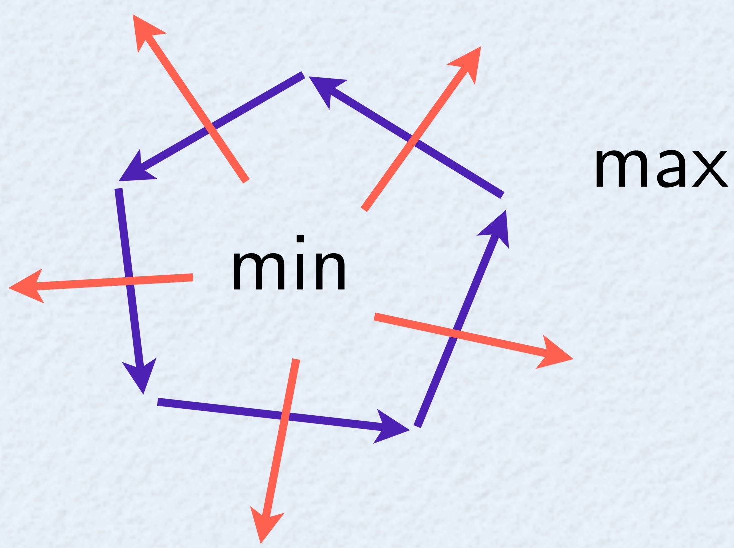
3. The min-max path can only cross the cycle once.

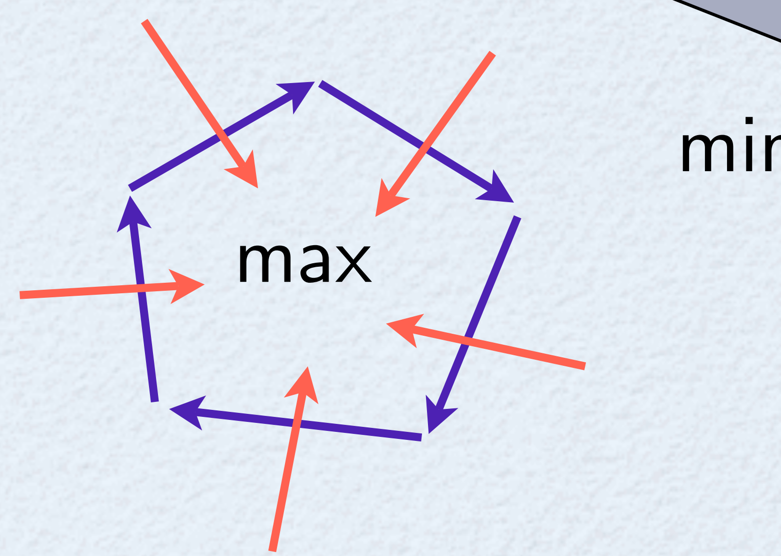
Partial Order on Triangles

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



1. max vertex is reachable from every vertex.
2. Every vertex is reachable from min (bd).



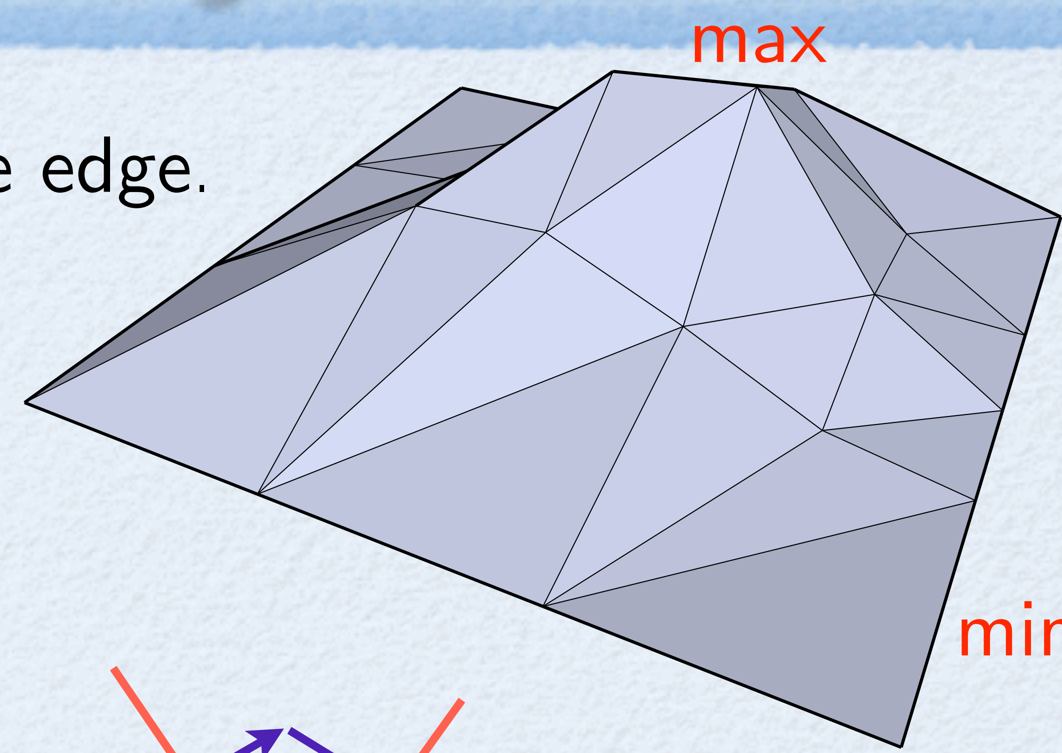


3. The min-max path can only cross the cycle once.

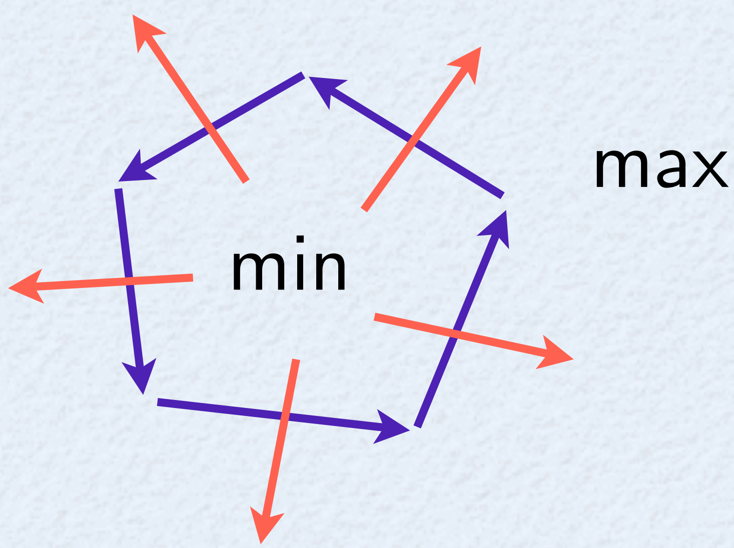
Lemma. The resulting dual graph is acyclic and the induced relation "$\prec$" a partial order. Thus we can topologically sort it into a total order.

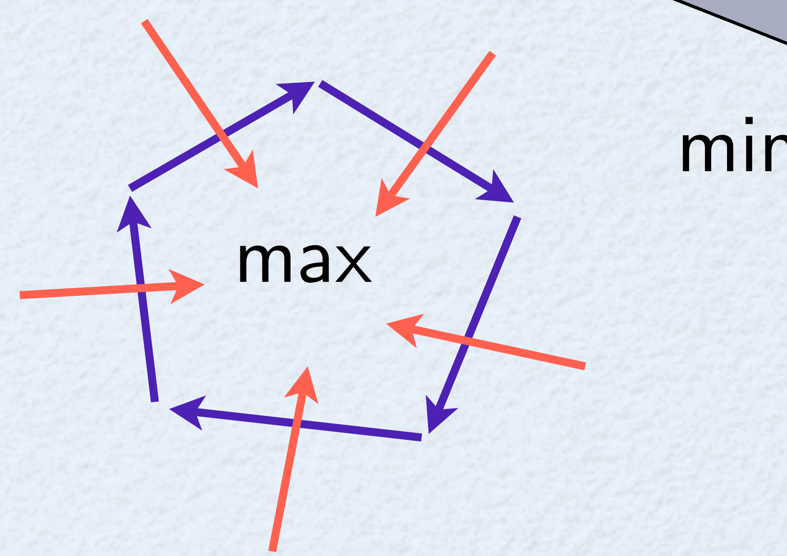
Partial Order on Triangles

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



1. max vertex is reachable from every vertex.
2. Every vertex is reachable from min (bd).





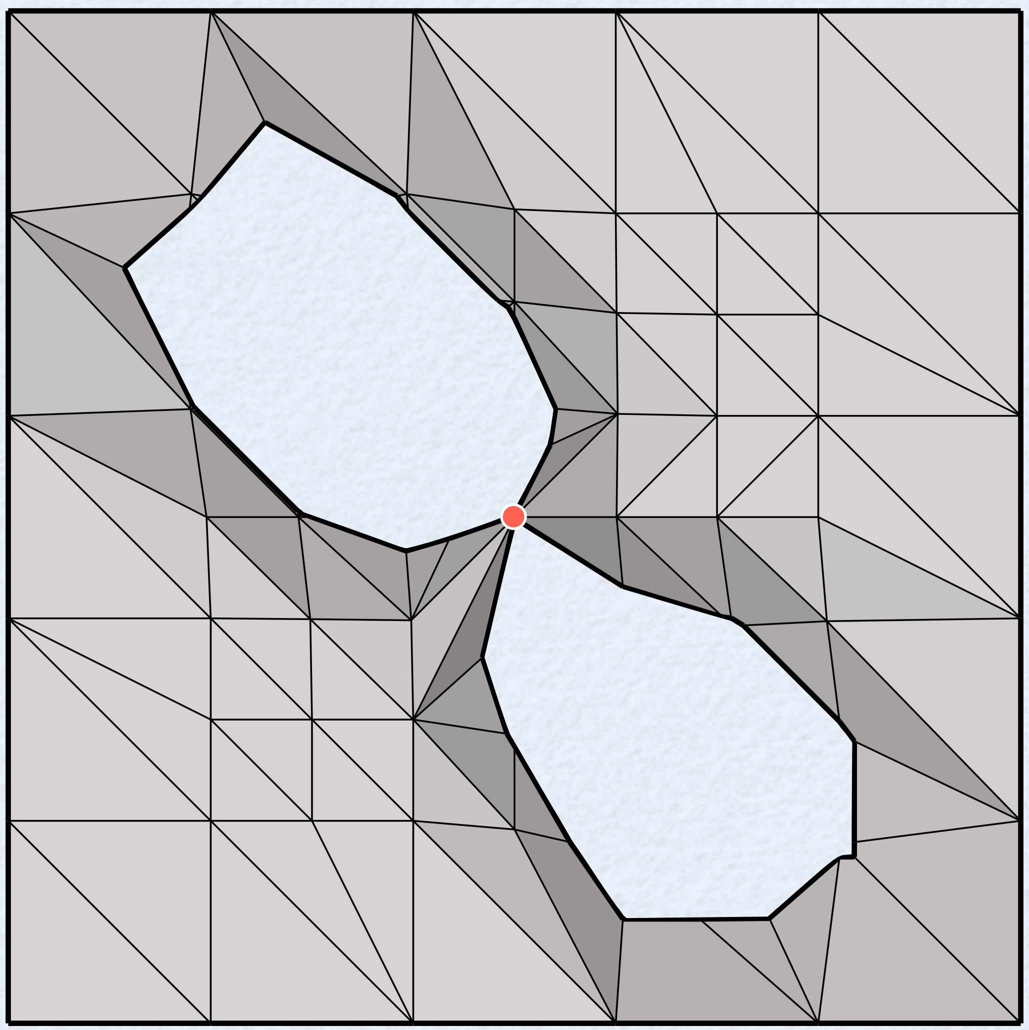
3. The min-max path can only cross the cycle once.

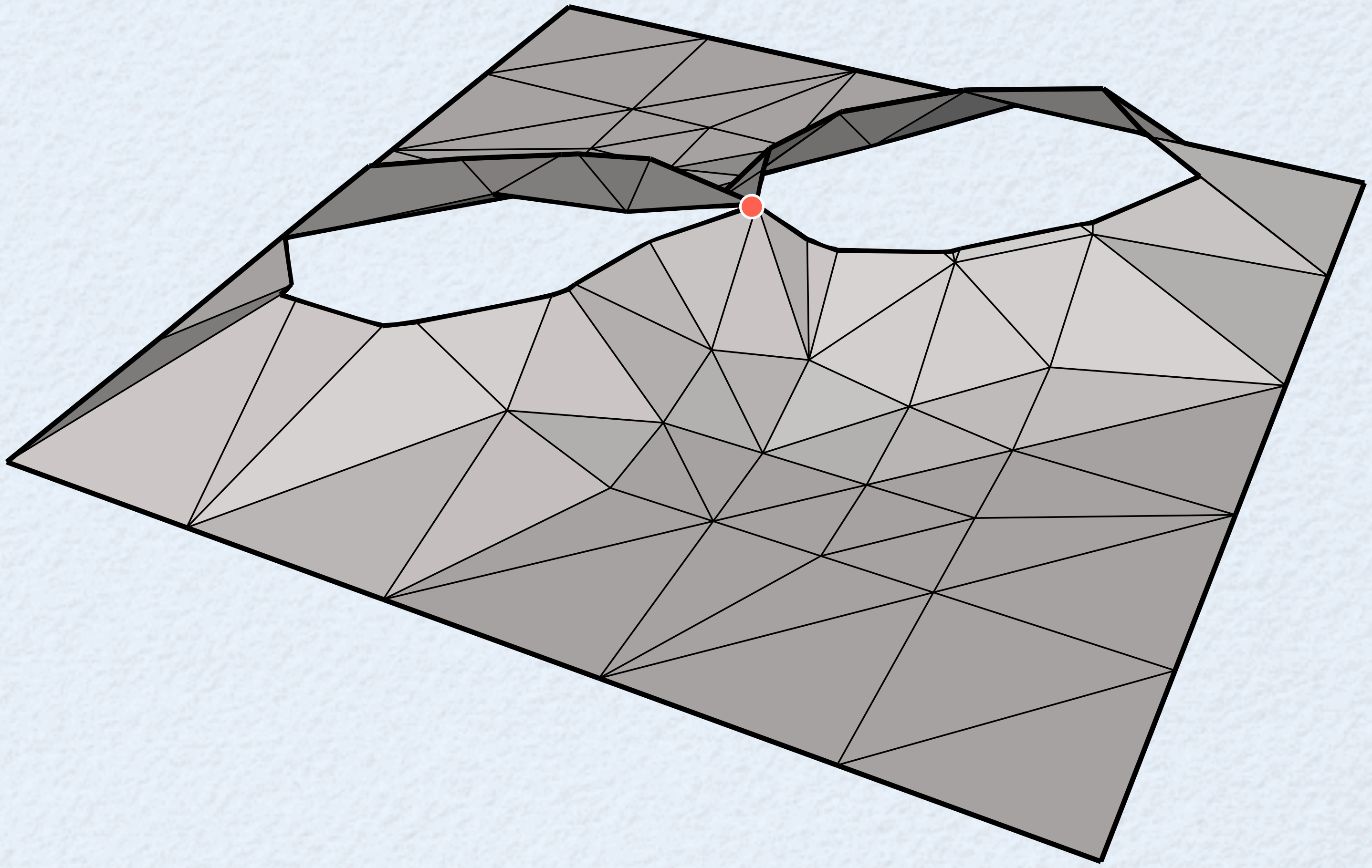
Lemma. The resulting dual graph is acyclic and the induced relation "$\prec$" a partial order. Thus we can topologically sort it into a total order.

[Arge, Toma, Zeh'03]

What about terrains with saddles?

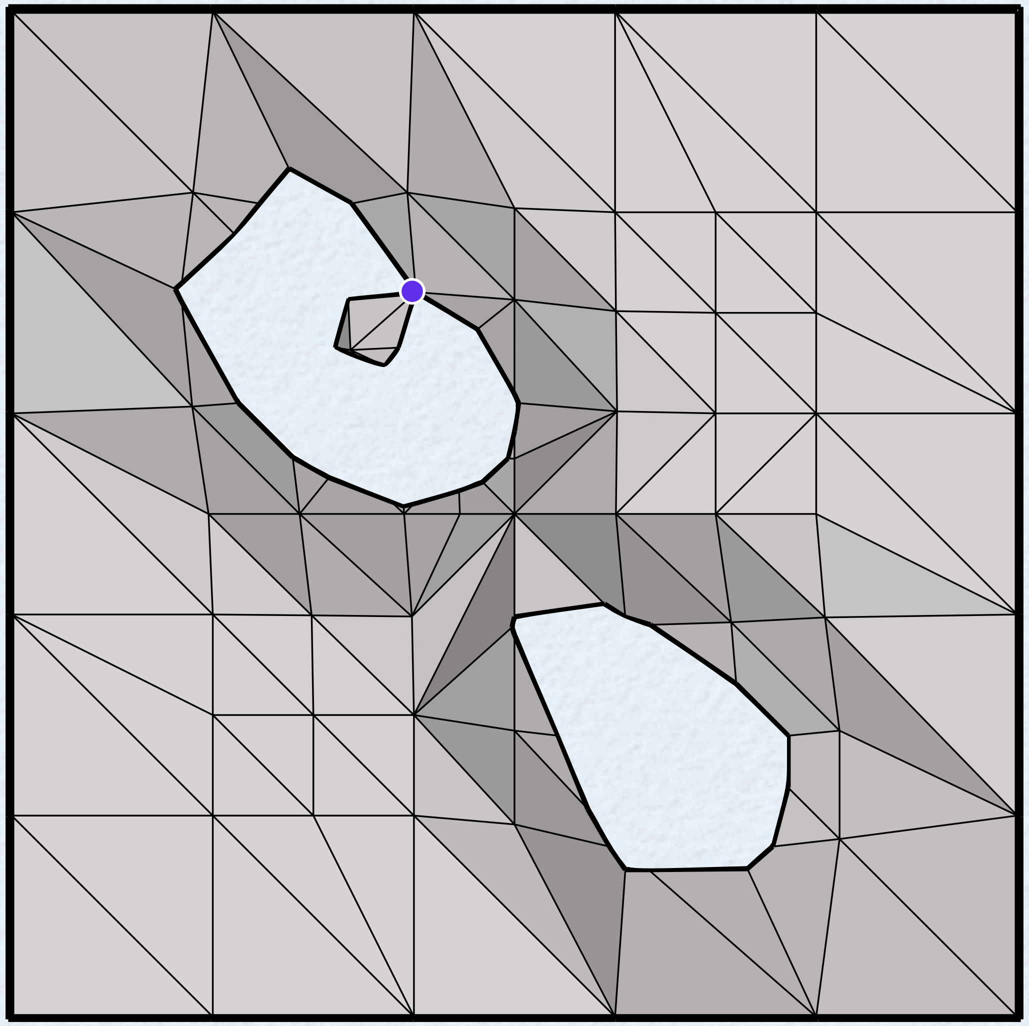
A saddle is negative if it joins two disjoint connected components of its sublevel-set and positive otherwise.

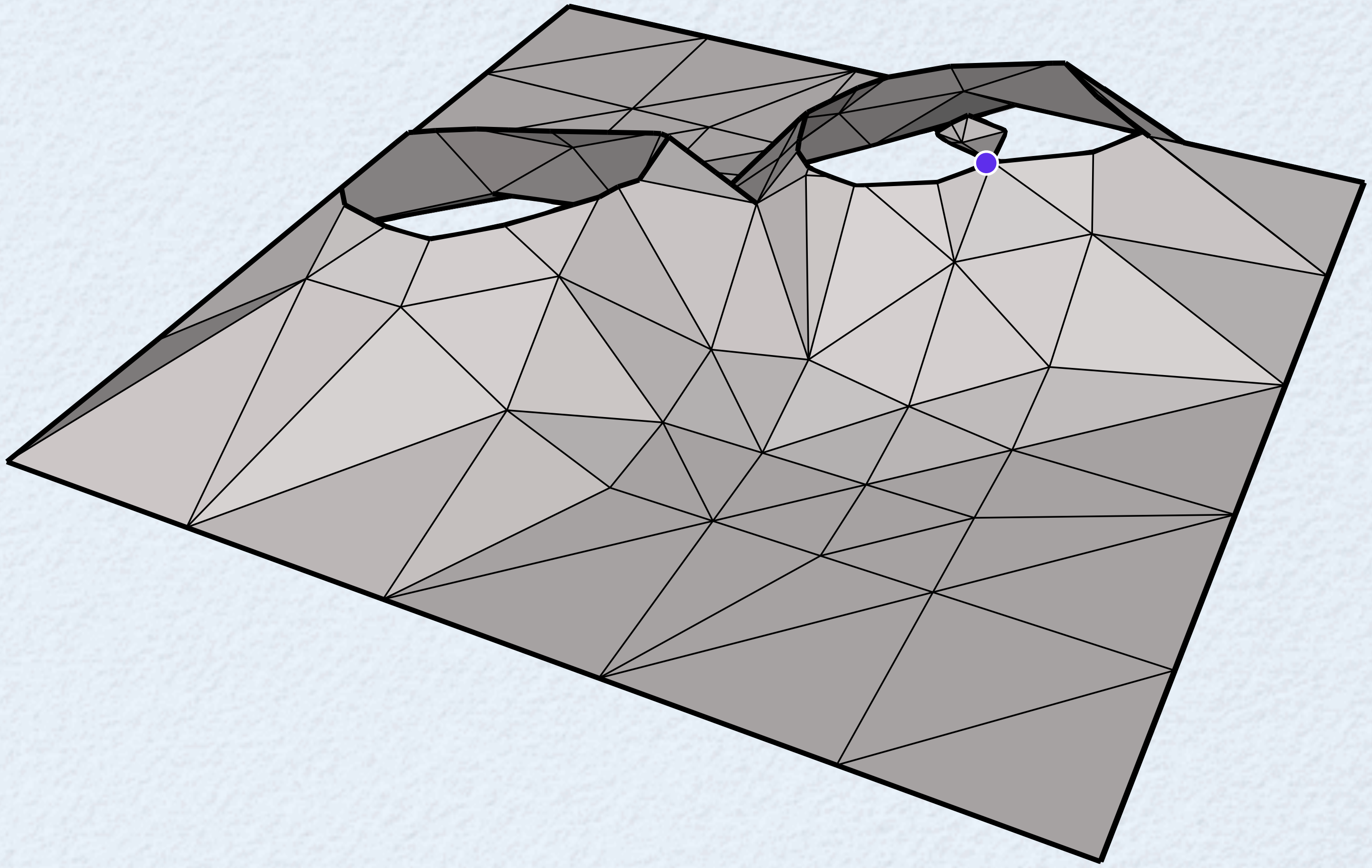




What about terrains with saddles?

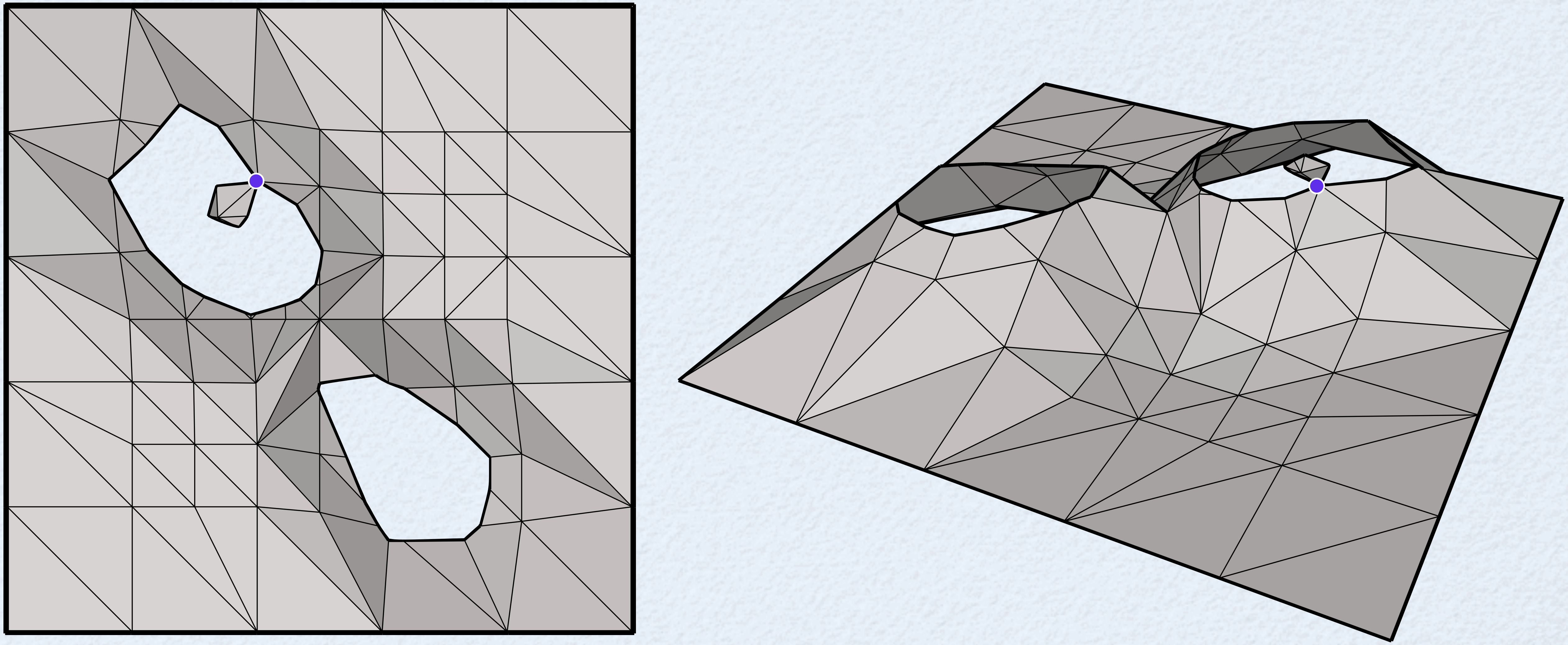
A saddle is negative if it joins two disjoint connected components of its sublevel-set and positive otherwise.





What about terrains with saddles?

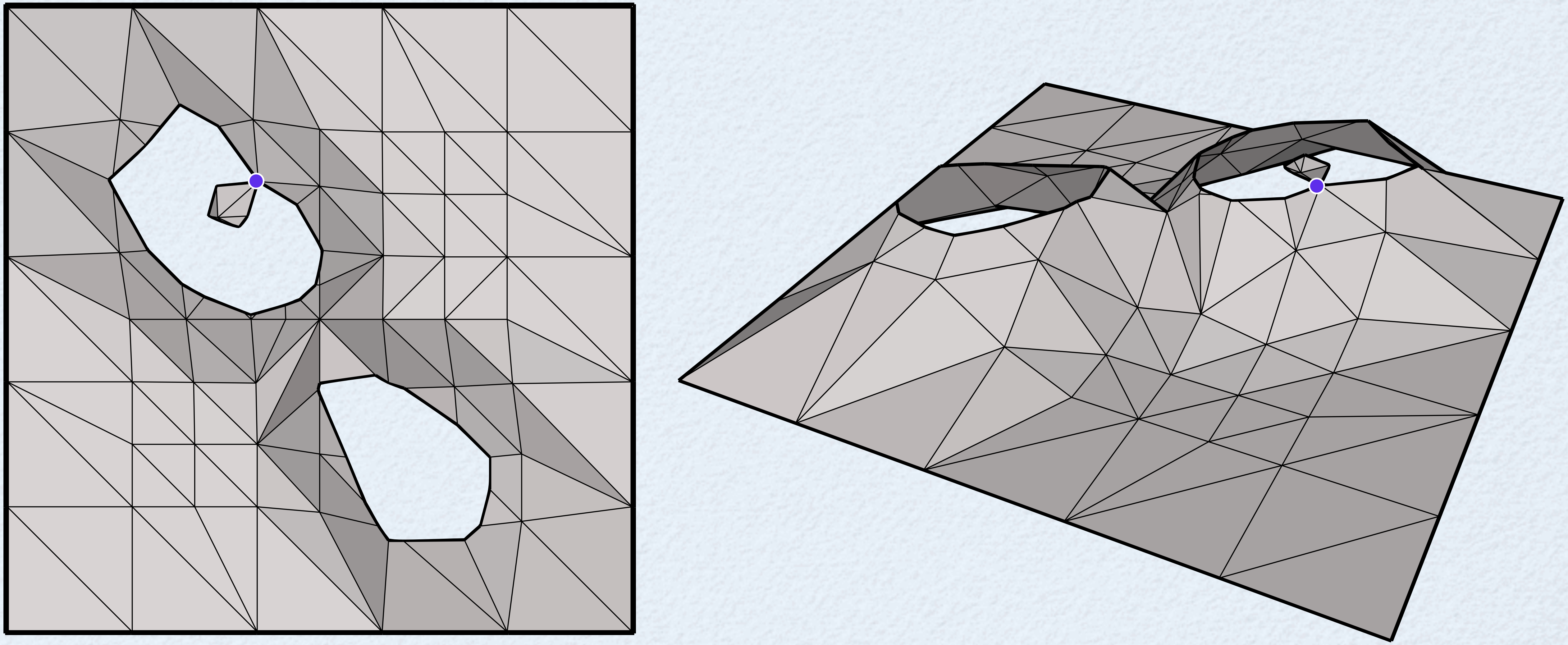
A saddle is negative if it joins two disjoint connected components of its sublevel-set and positive otherwise.



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

What about terrains with saddles?

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

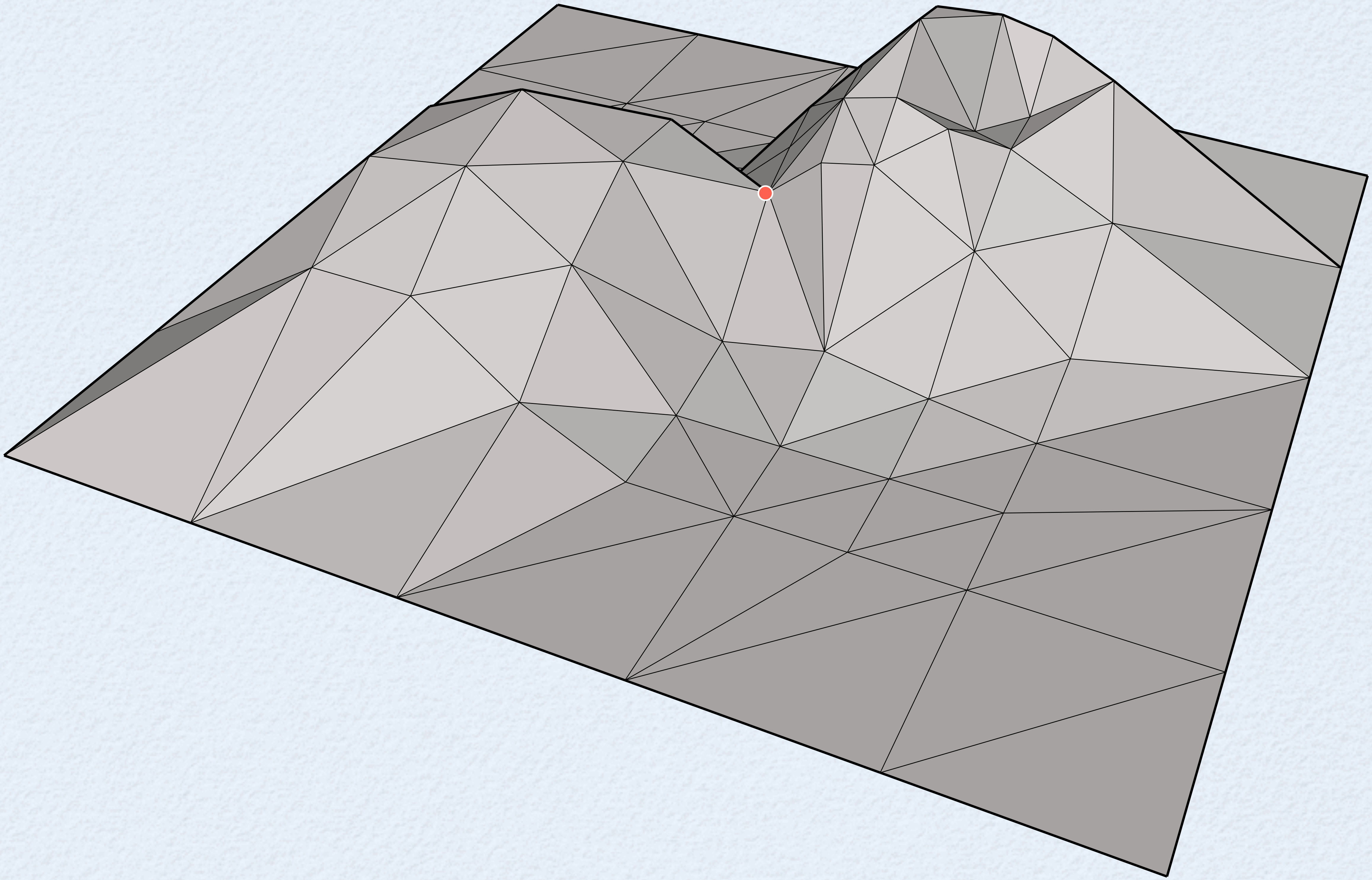


If we replace h with $-h$, the two types switch roles.

[Agarwal, Arge, Yi '06] Positive and negative saddle points can be found in $O(\text{Sort}(N))$ I/Os.

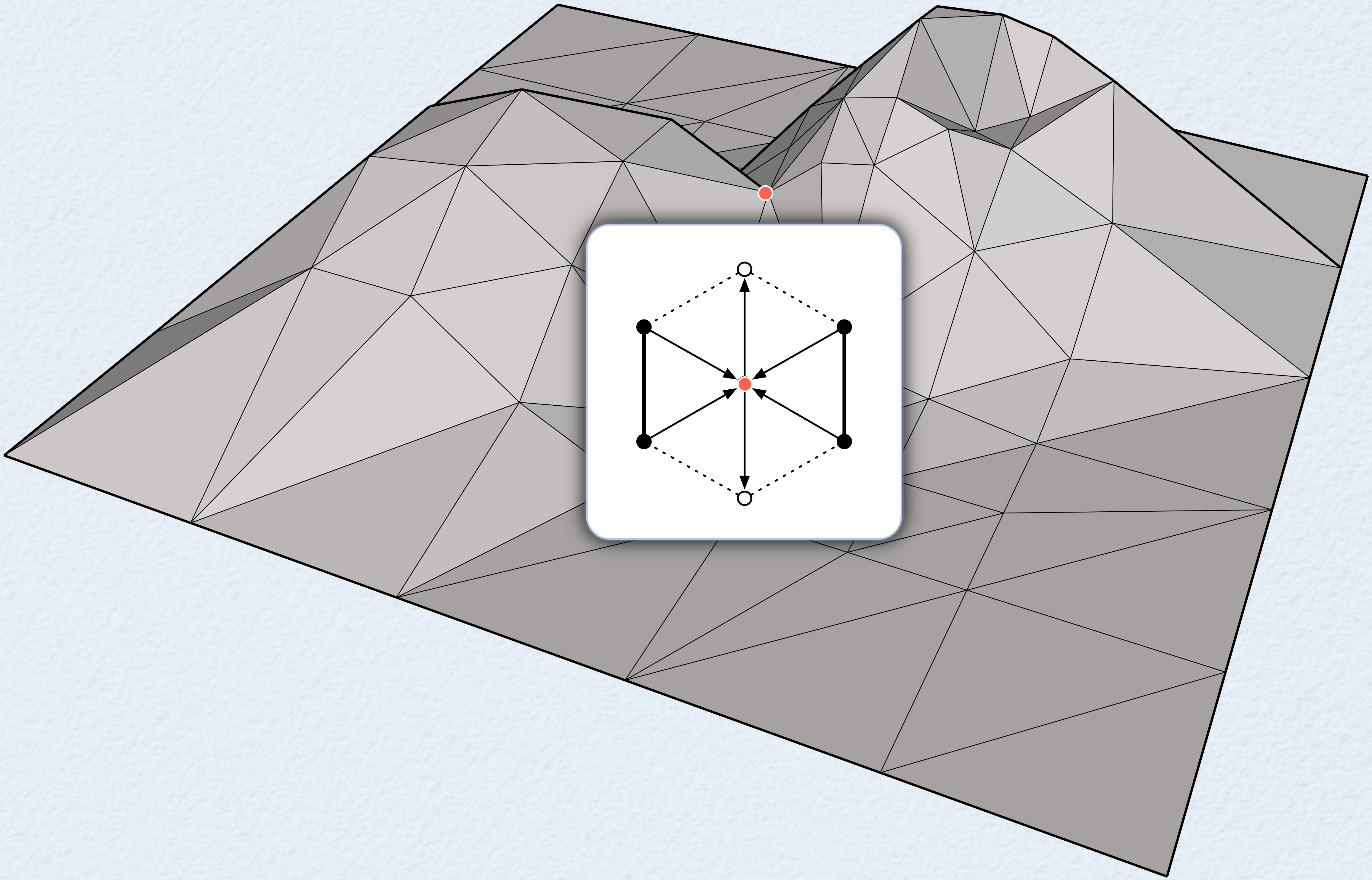
Positive and Negative Cut-Trees

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



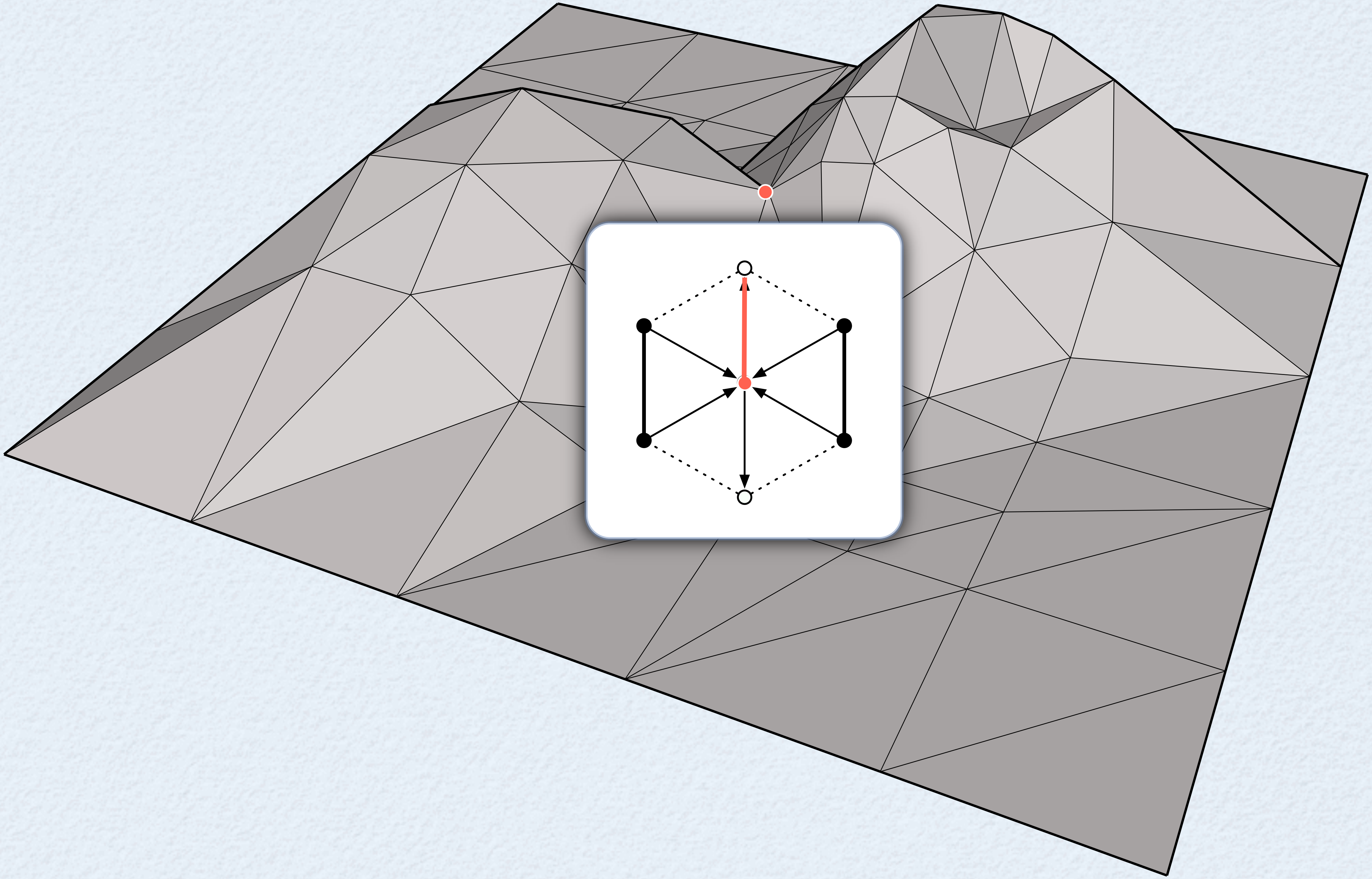
Positive and Negative Cut-Trees

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



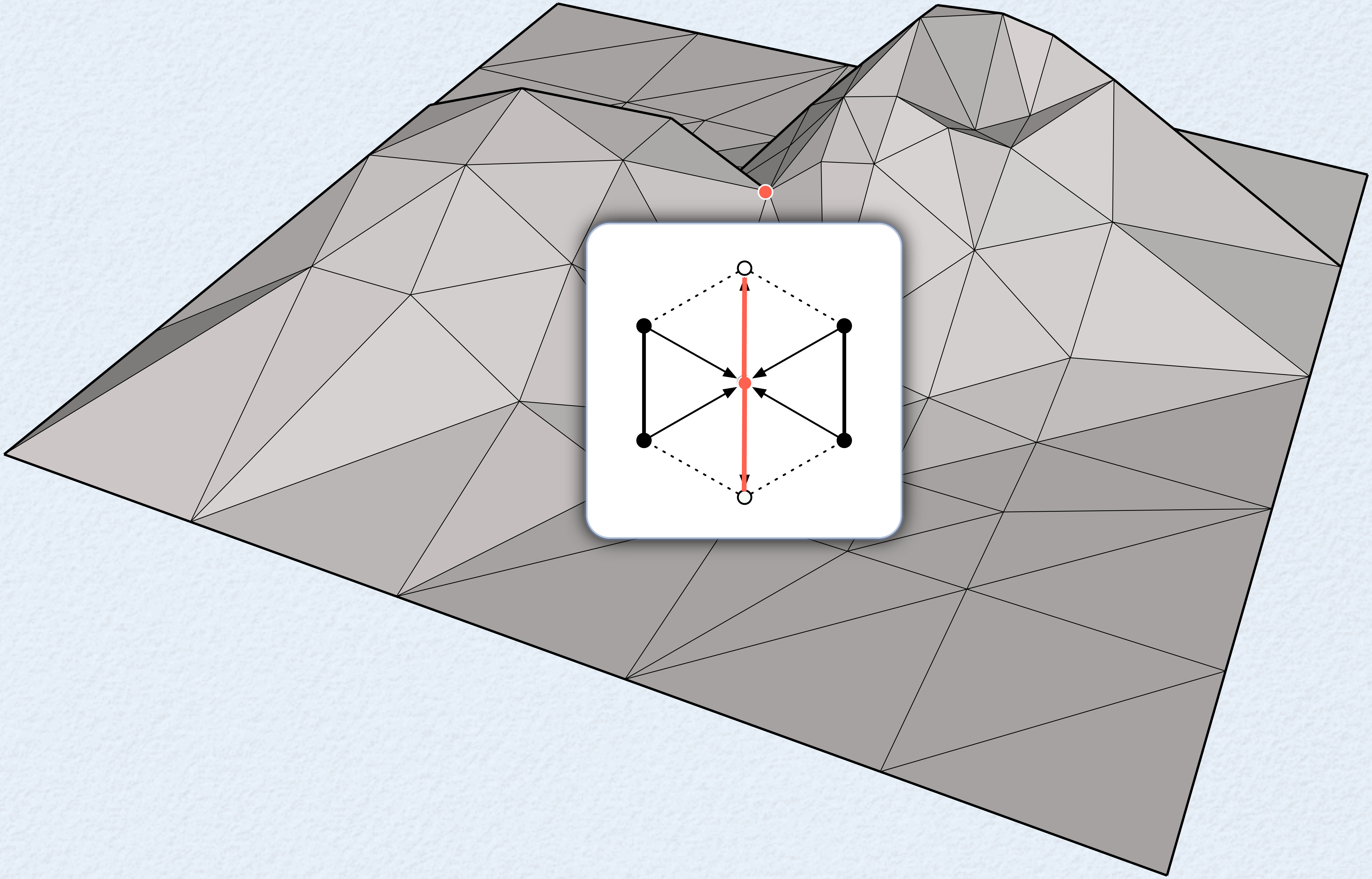
Positive and Negative Cut-Trees

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



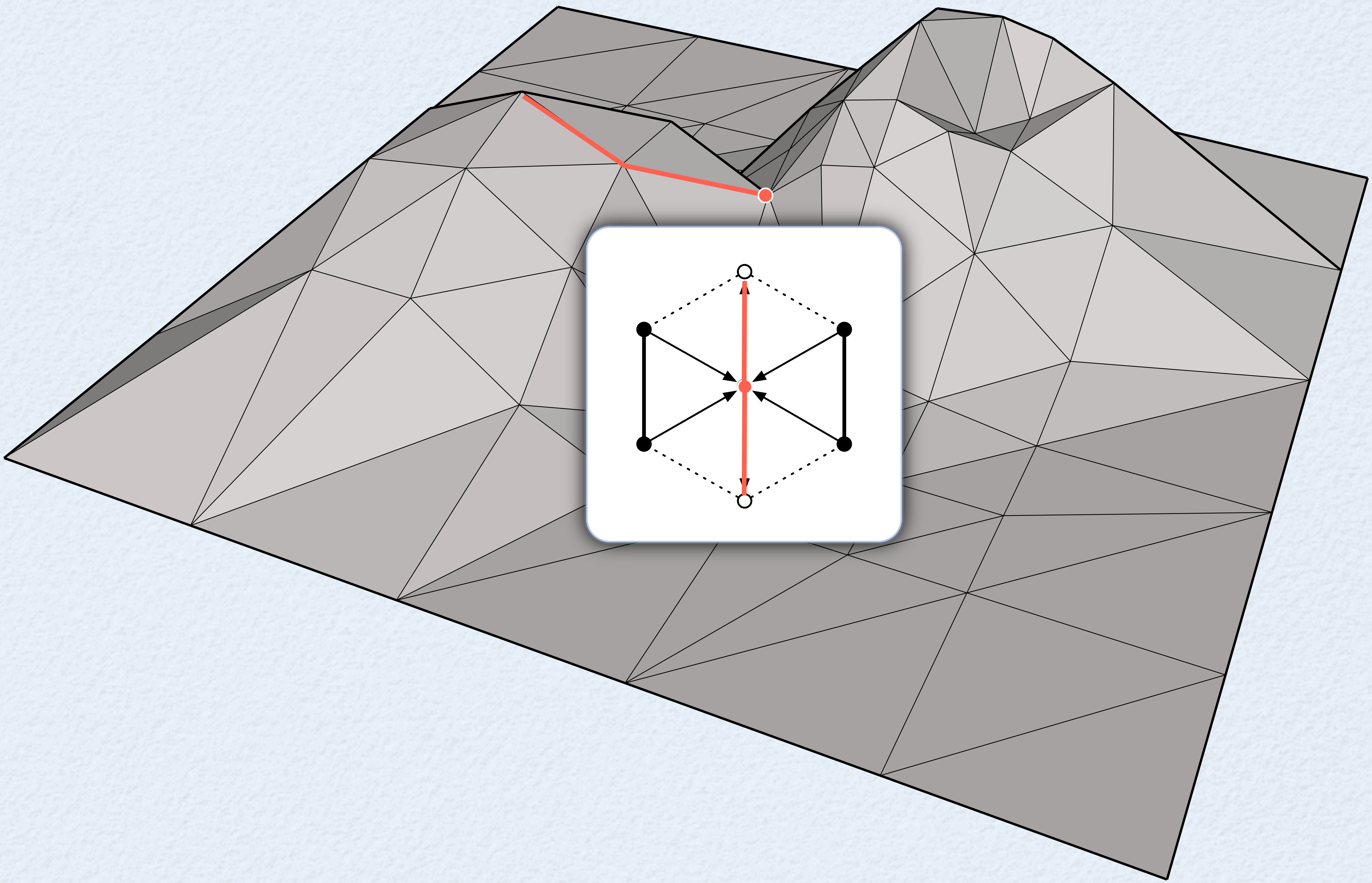
Positive and Negative Cut-Trees

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



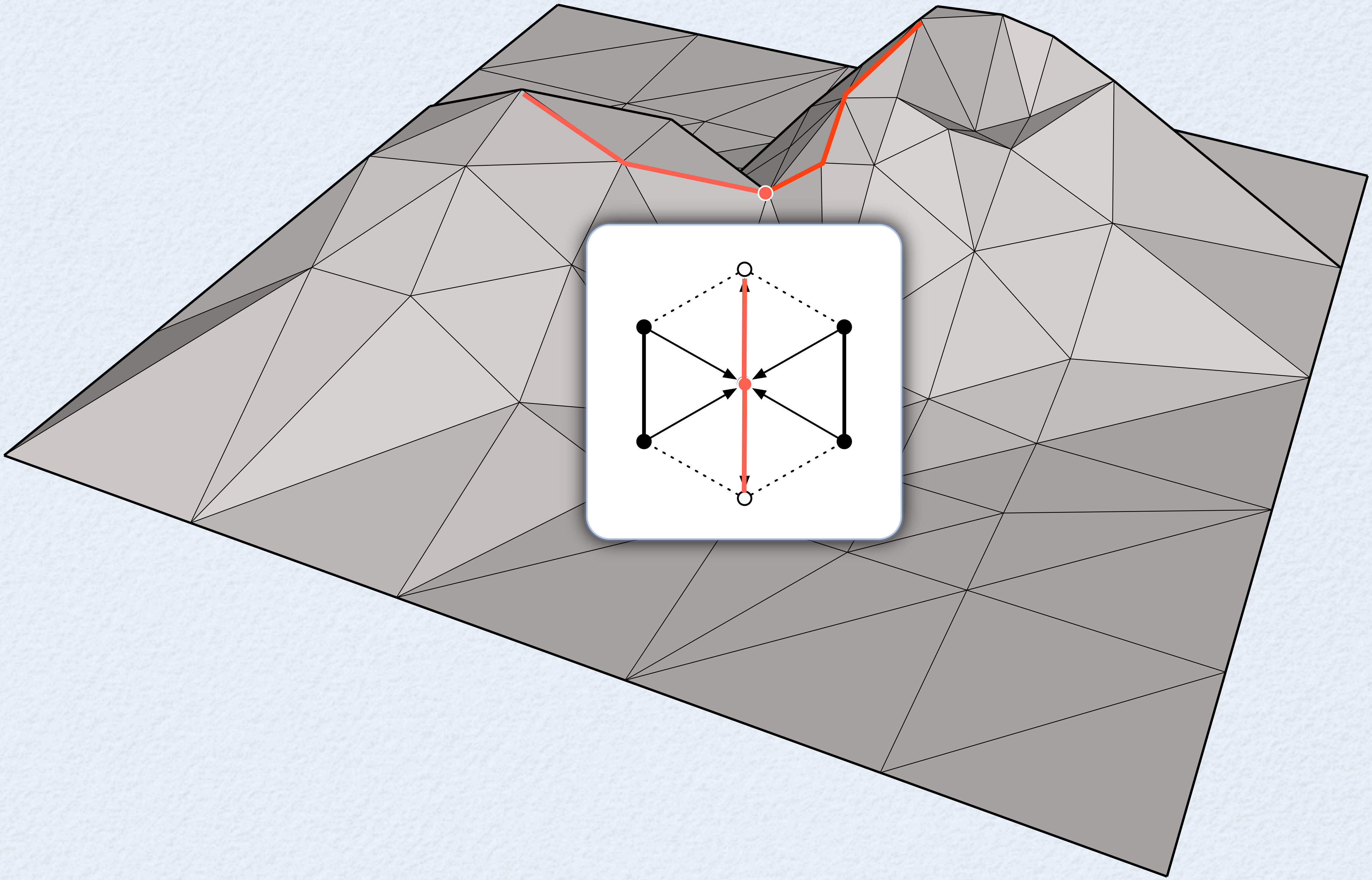
Positive and Negative Cut-Trees

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



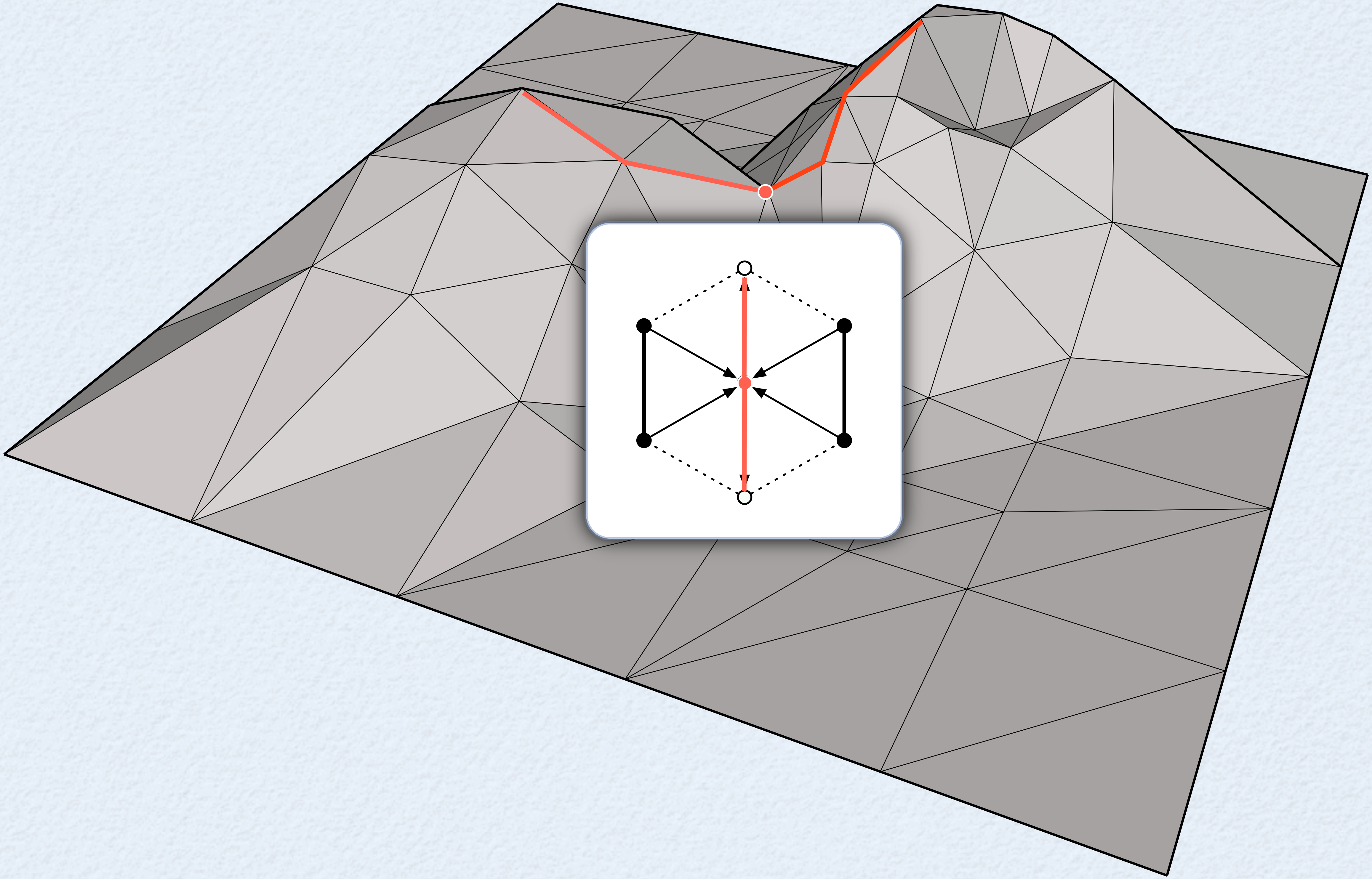
Positive and Negative Cut-Trees

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 and Negative Cut-Trees

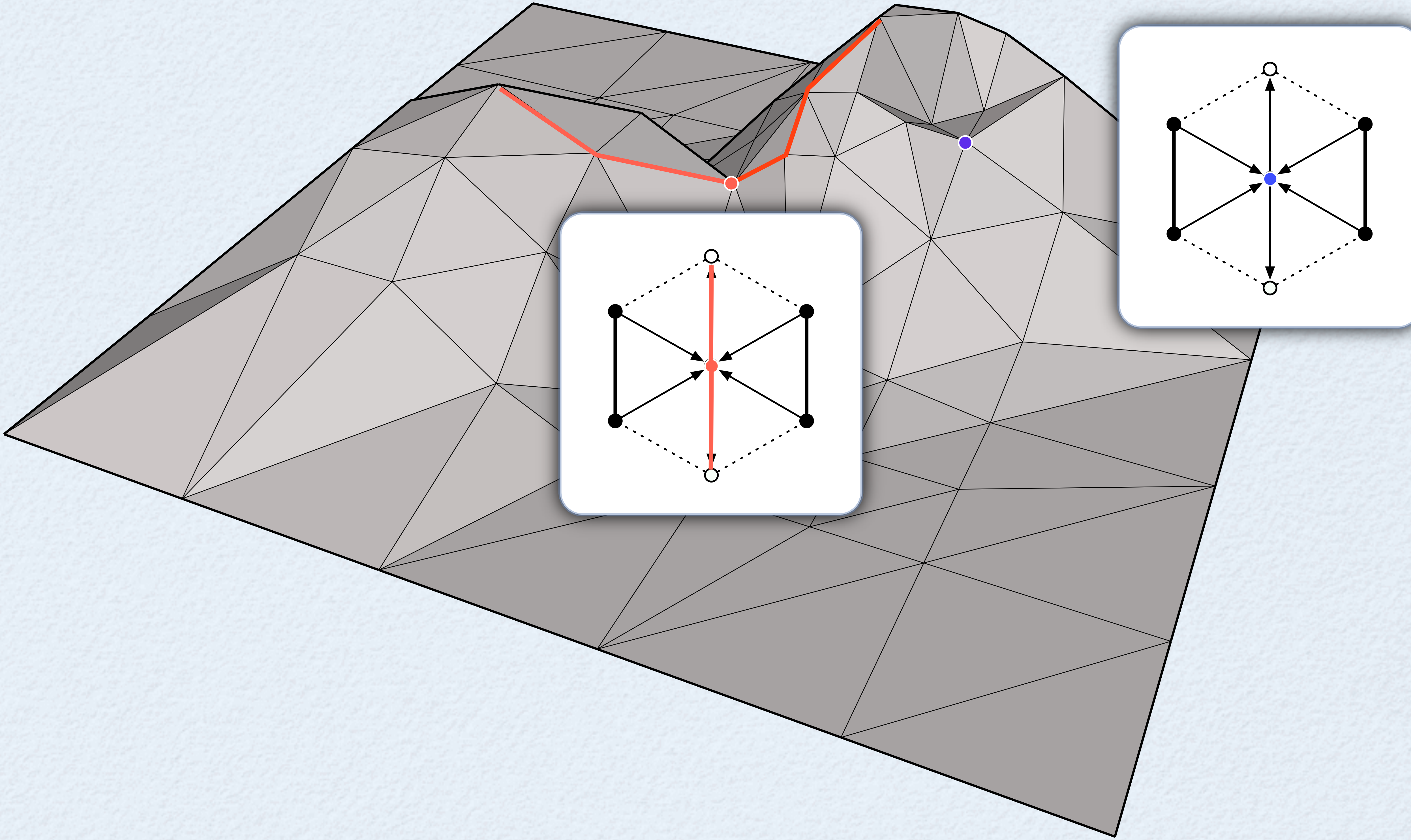
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.



Lemma. The result is a tree (not just forest) that reaches every maximum.

Positive and Negative Cut-Trees

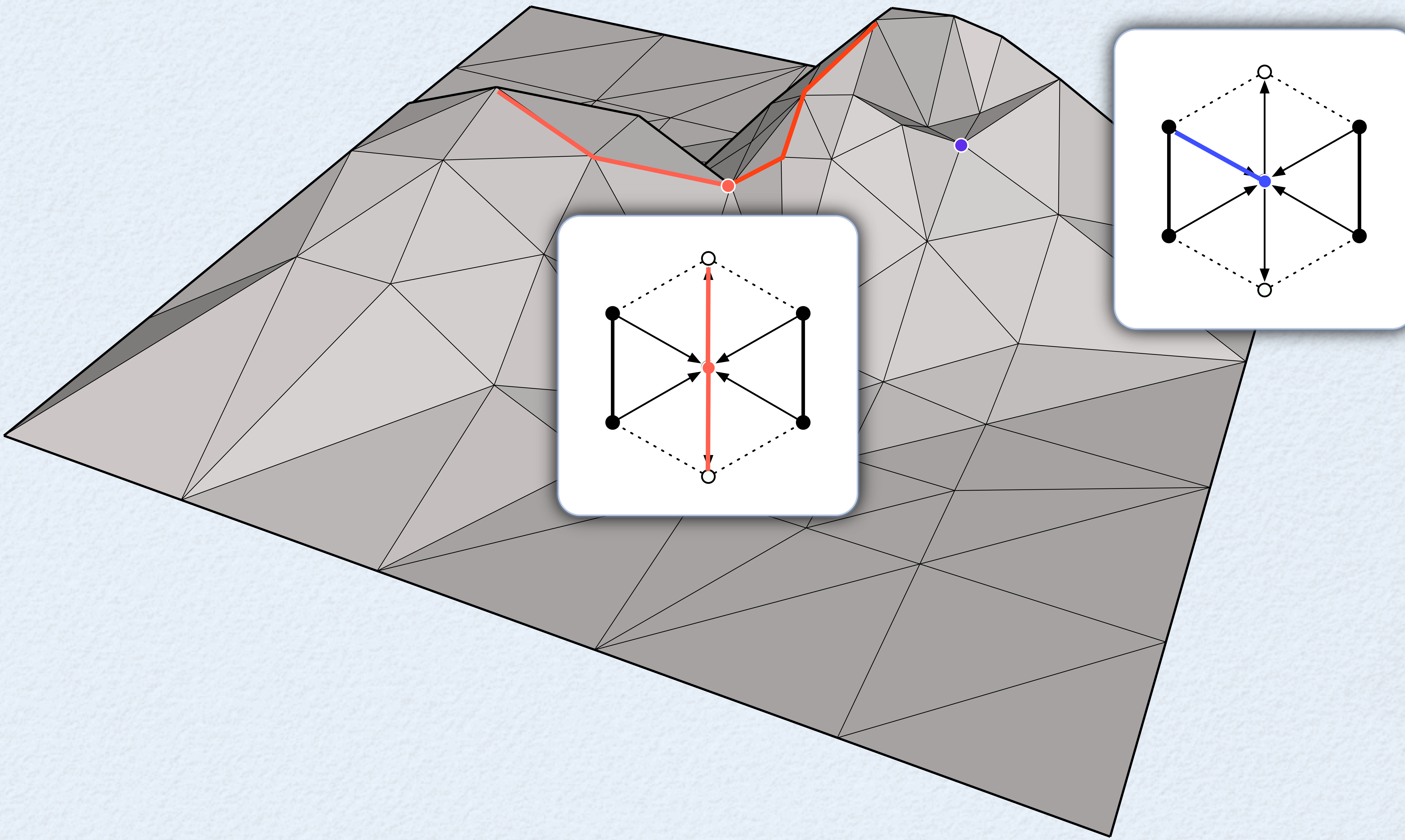
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.



Lemma. The result is a tree (not just forest) that reaches every maximum.

Positive and Negative Cut-Trees

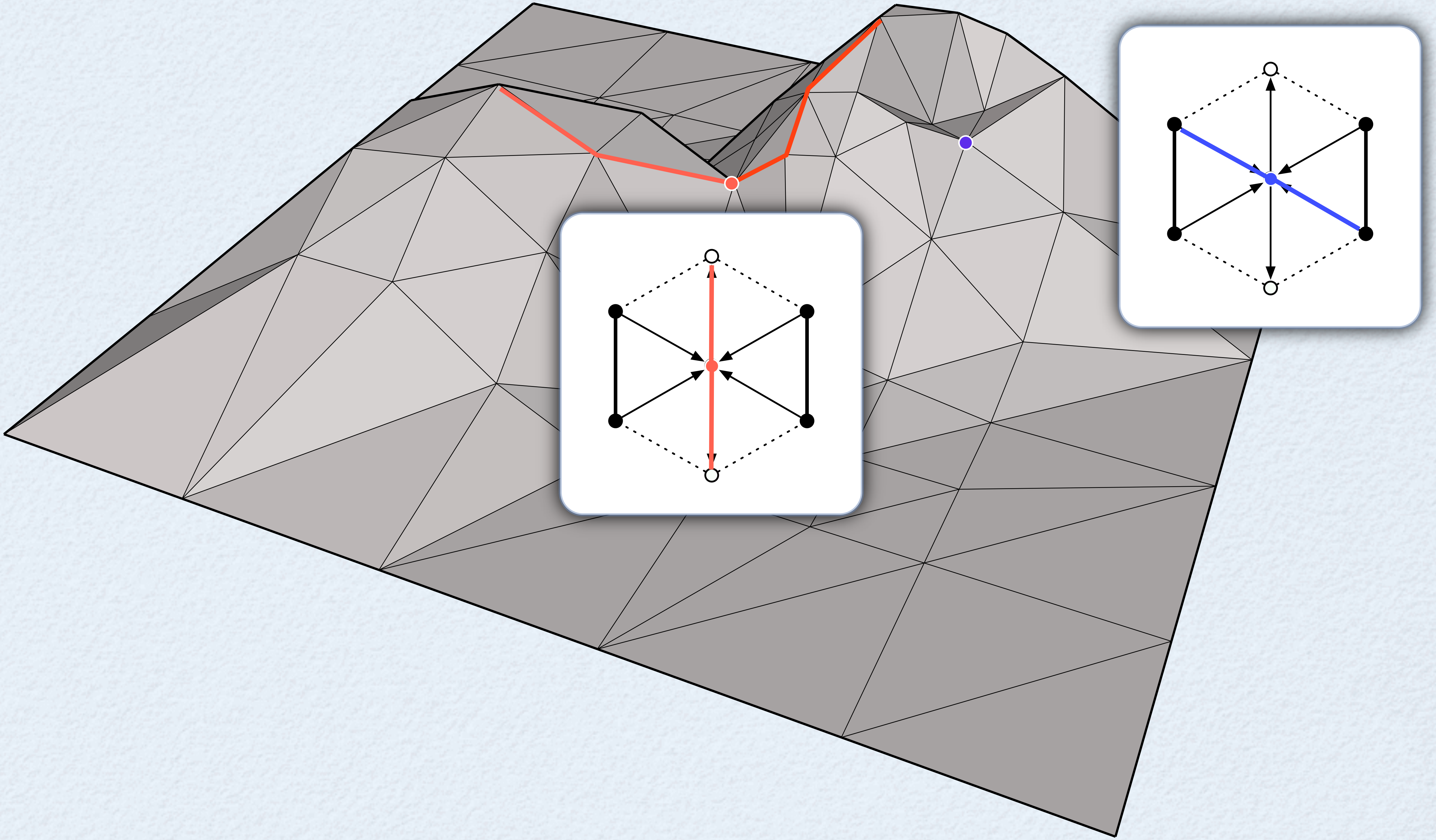
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.



Lemma. The result is a tree (not just forest) that reaches every maximum.

Positive and Negative Cut-Trees

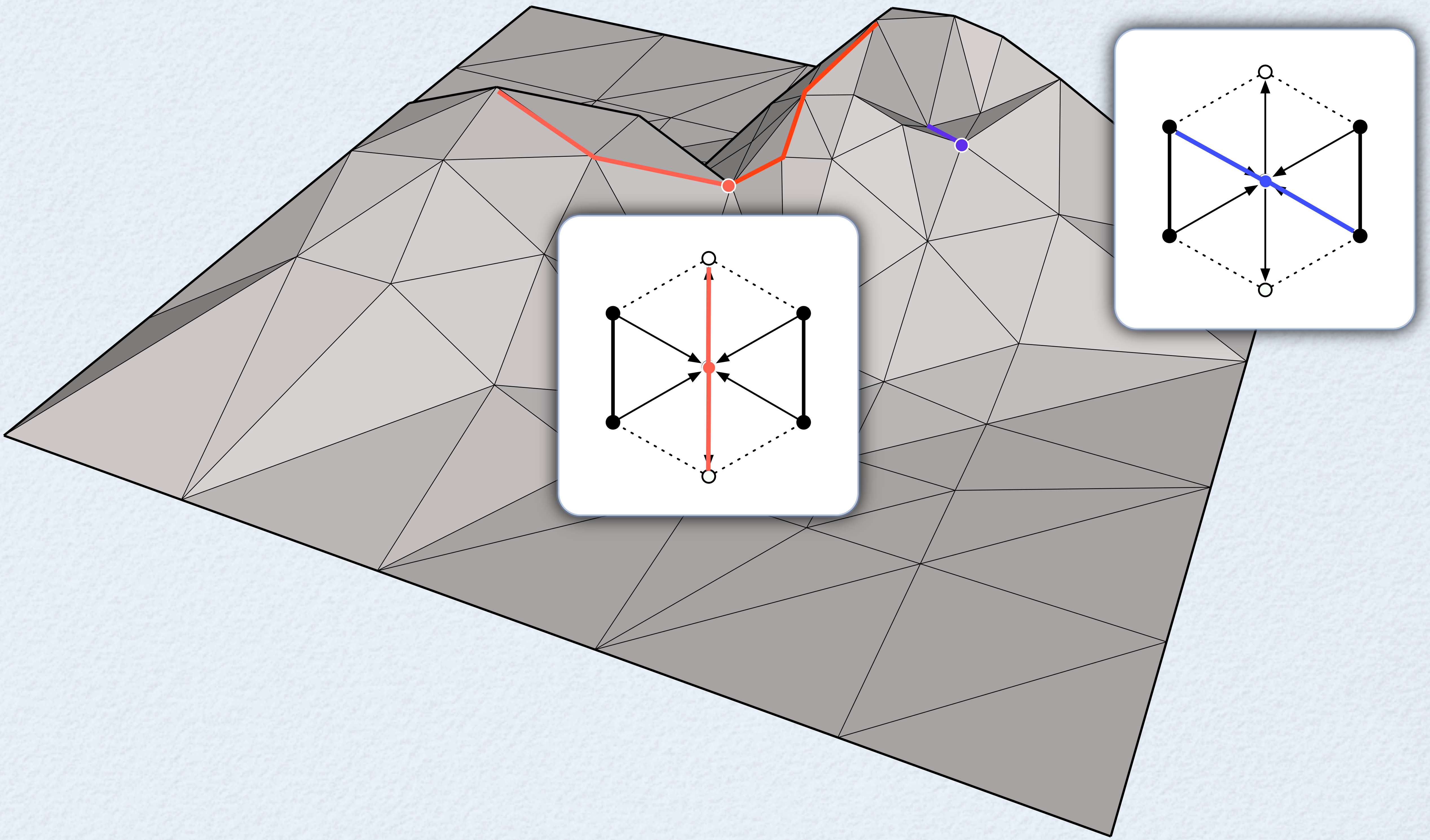
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.



Lemma. The result is a tree (not just forest) that reaches every maximum.

Positive and Negative Cut-Trees

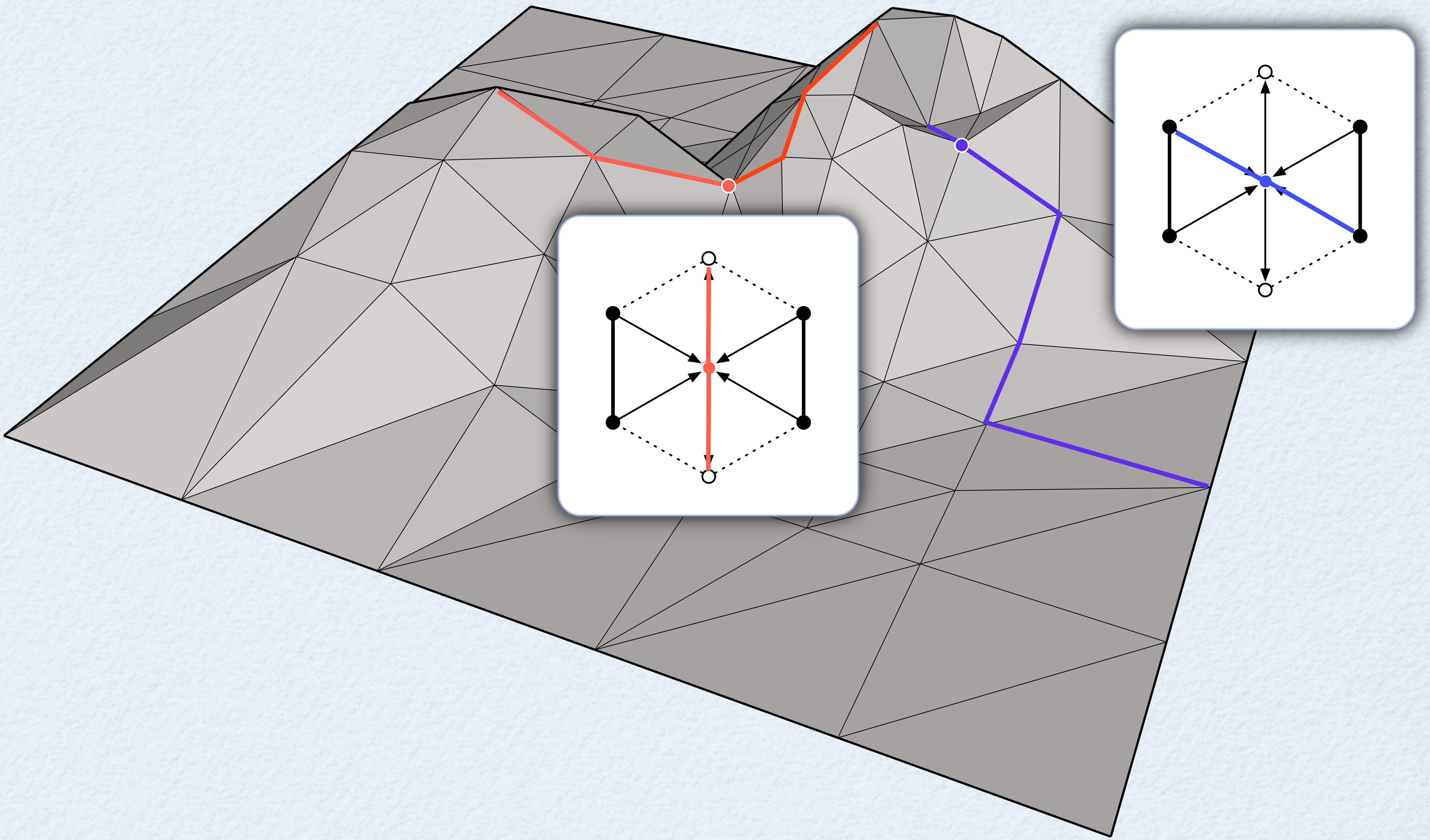
Positive Cut-Tree: follow an ascending path in every connected component of the upper link of every positive saddle, joining paths when they collide.



Lemma. The result is a tree (not just forest) that reaches every maximum.

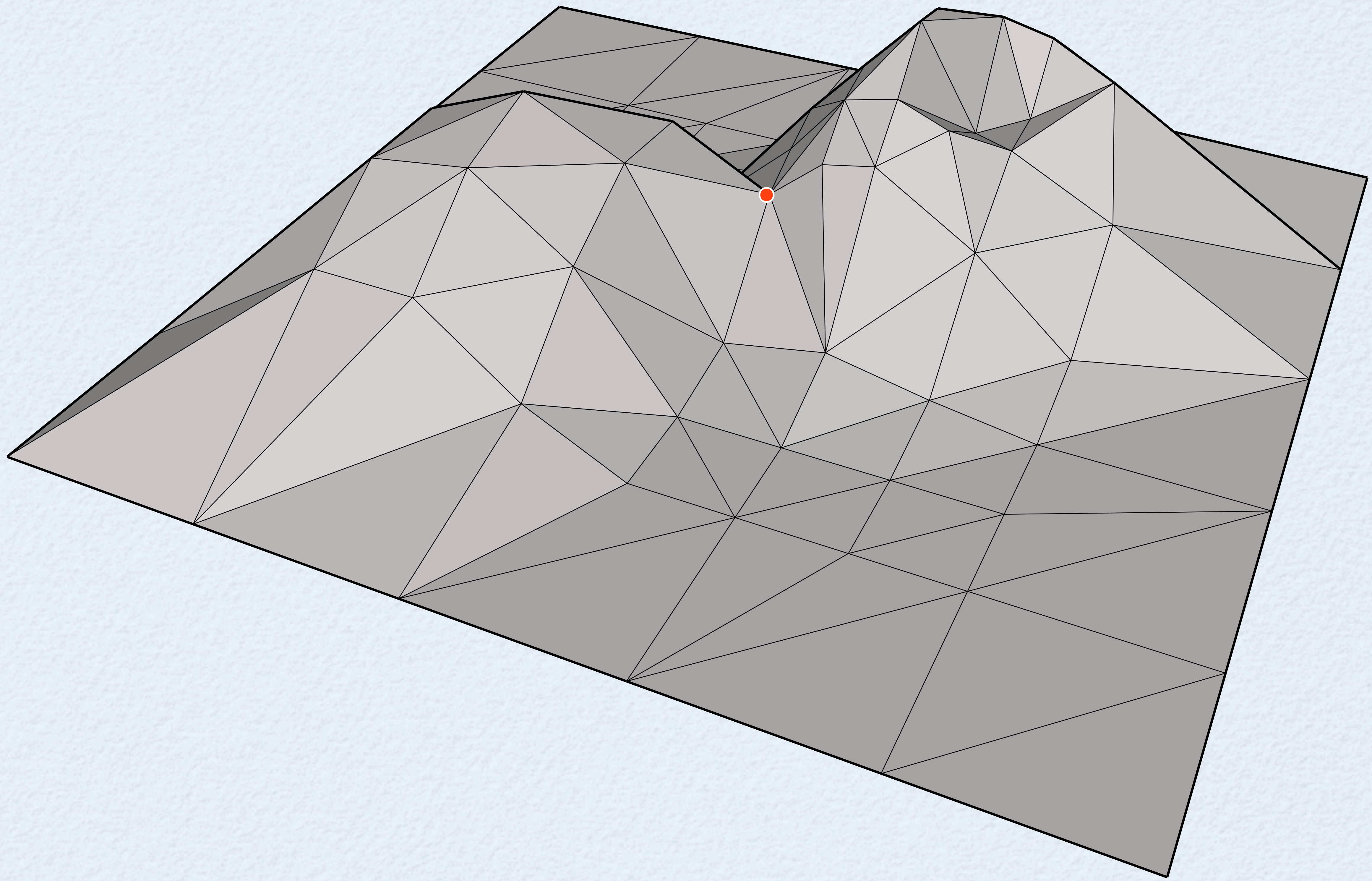
Positive and Negative Cut-Trees

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

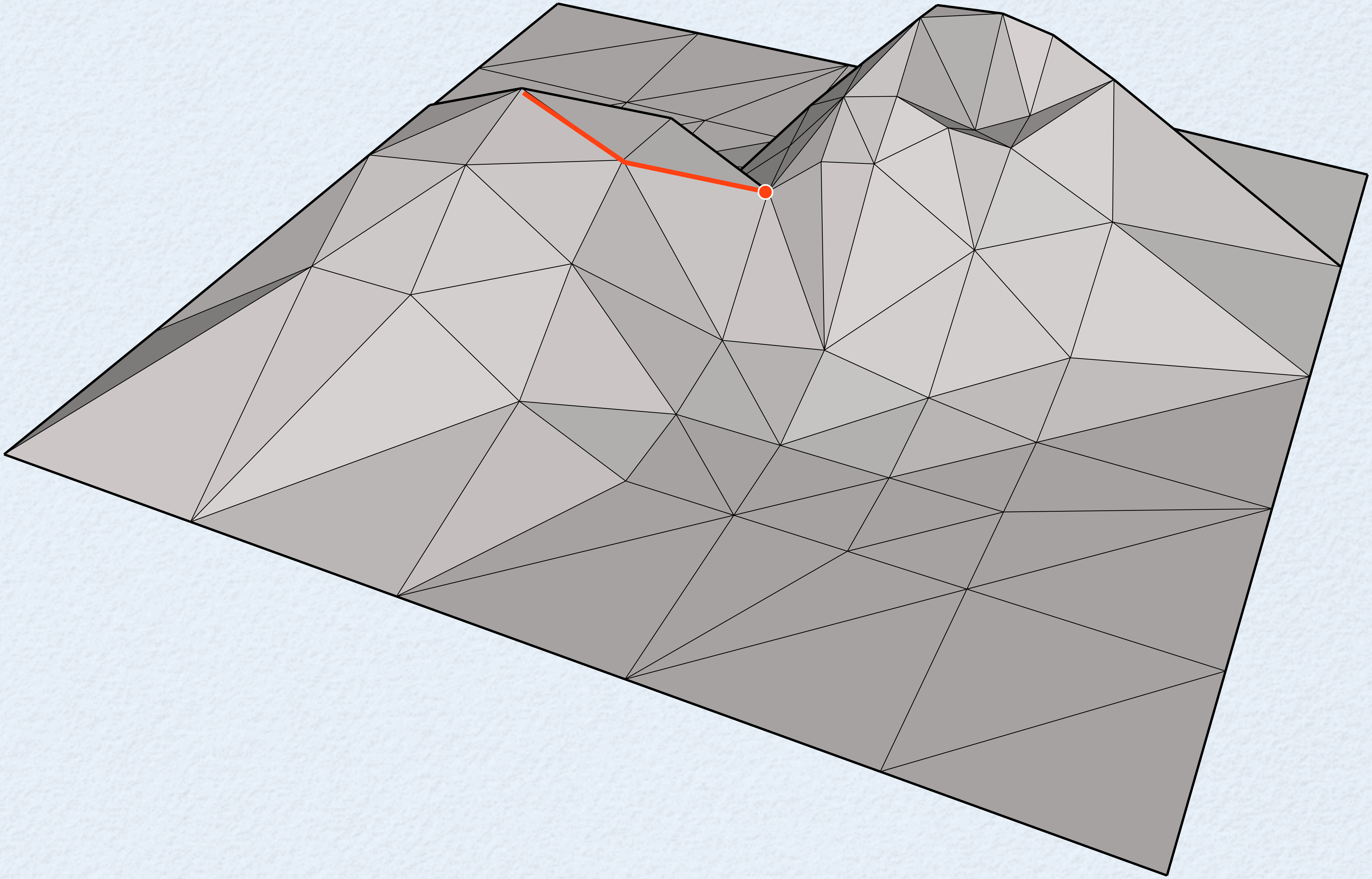


Lemma. The result is a tree (not just forest) that reaches every maximum.

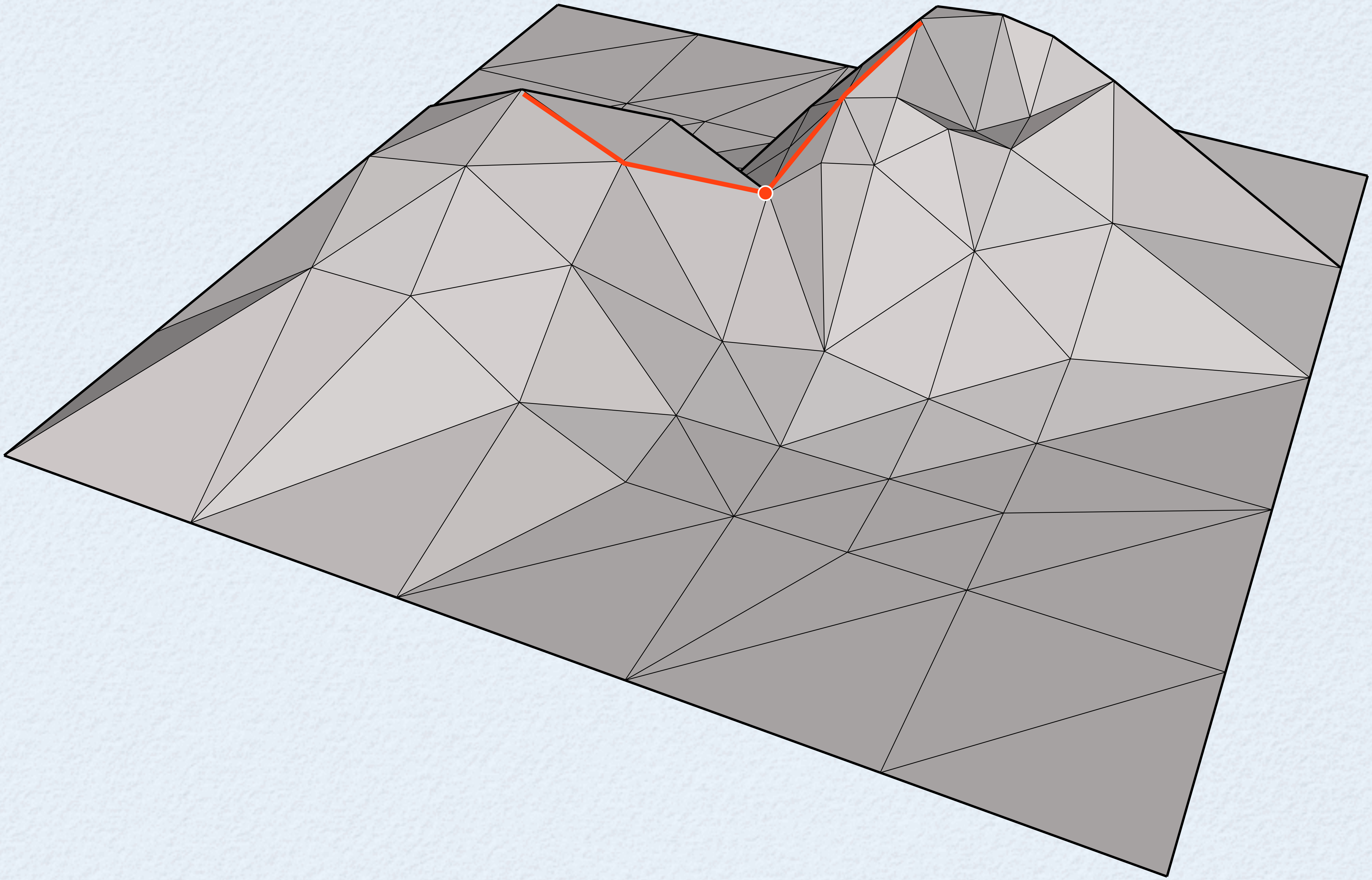
Turning a terrain into an elementary one by surgery



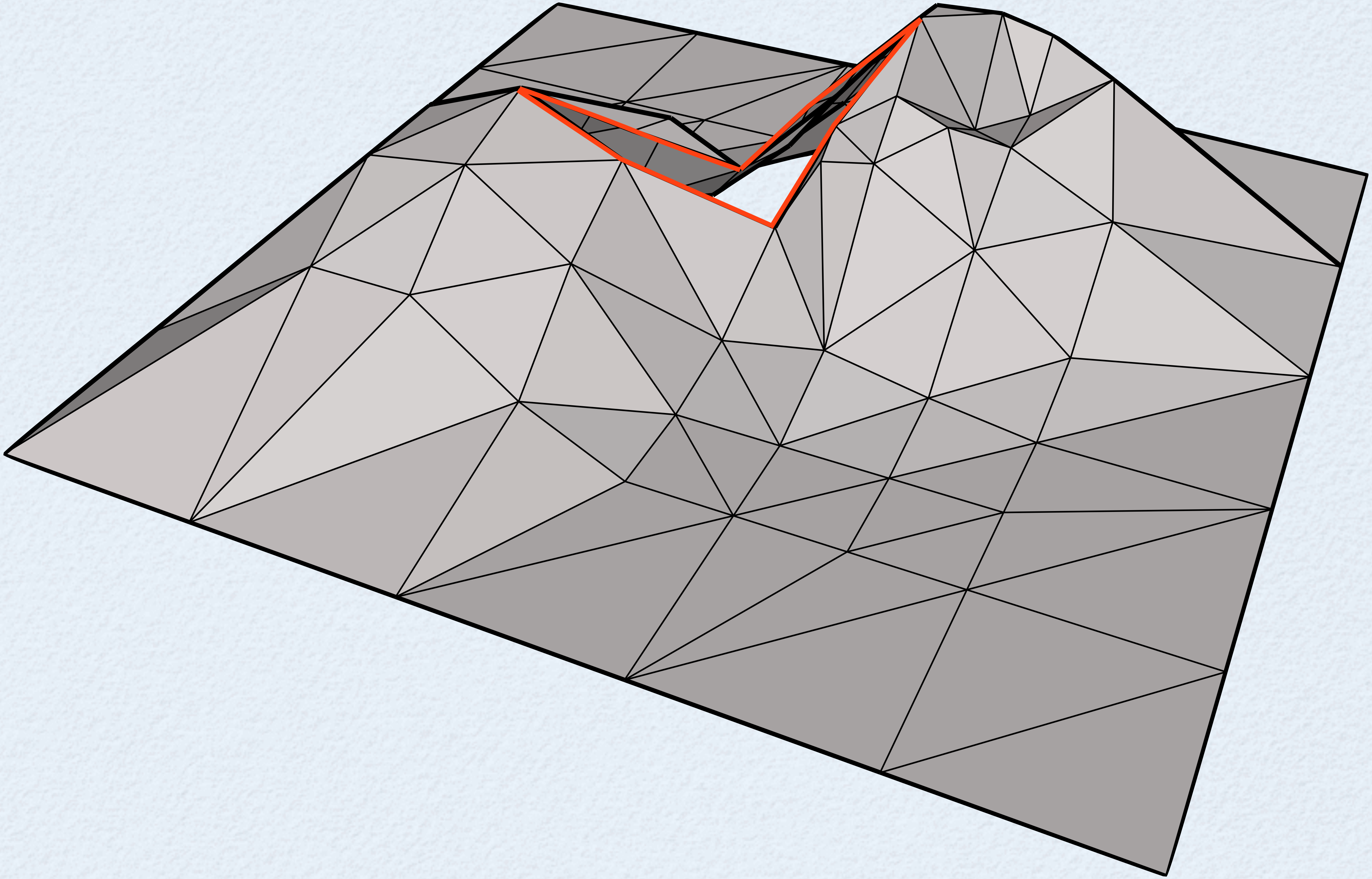
Turning a terrain into an elementary one by surgery



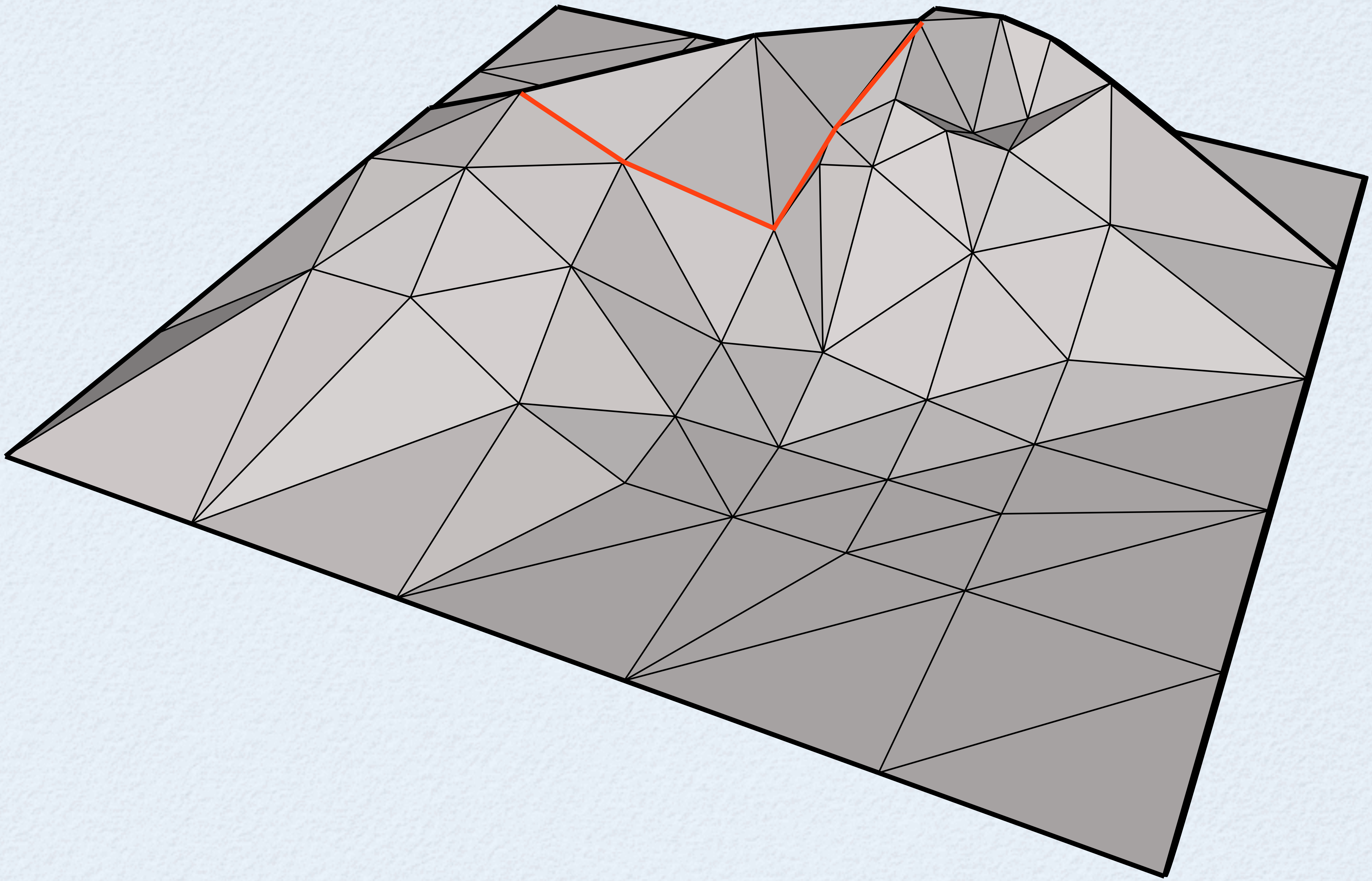
Turning a terrain into an elementary one by surgery



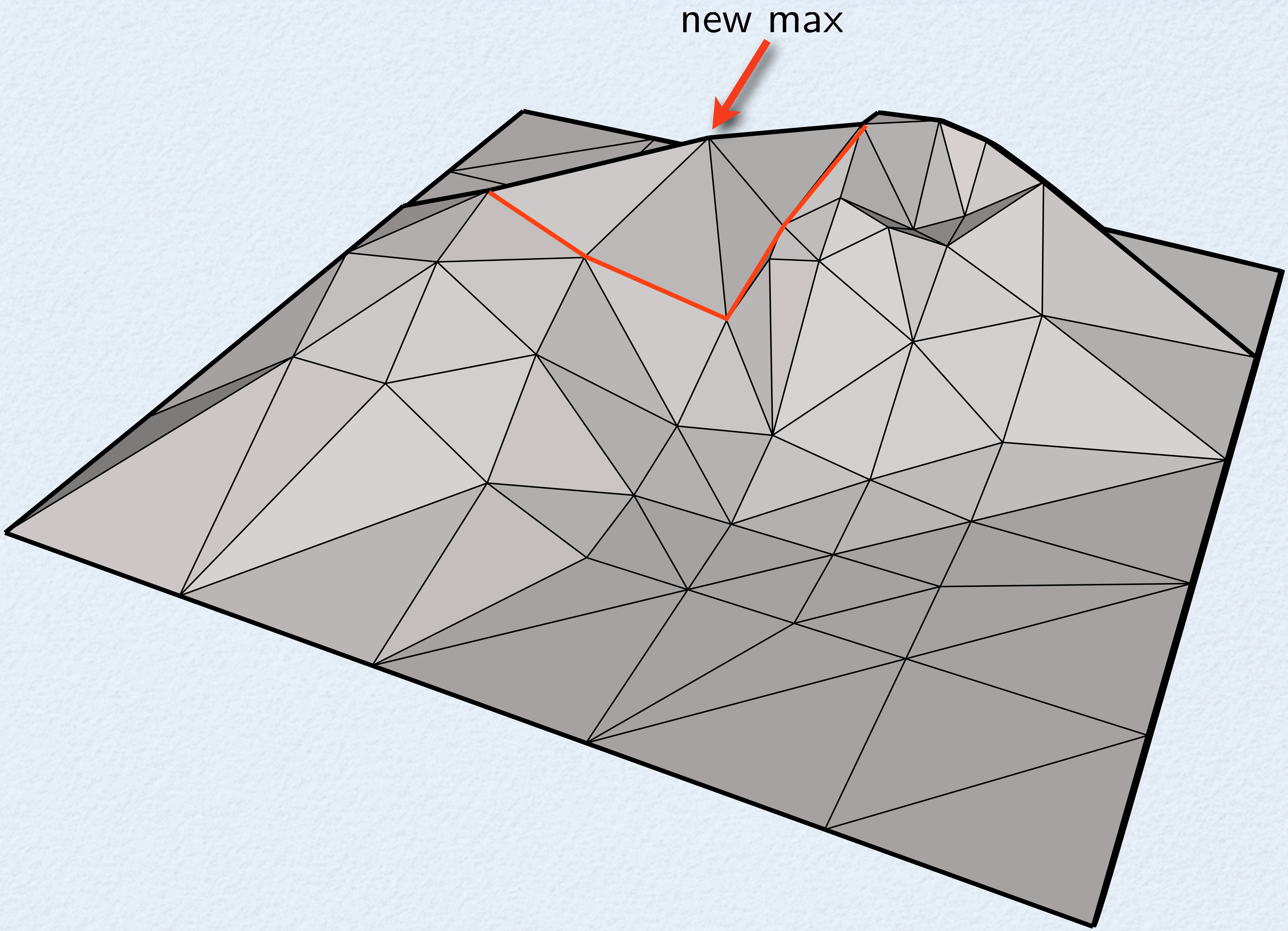
Turning a terrain into an elementary one by surgery



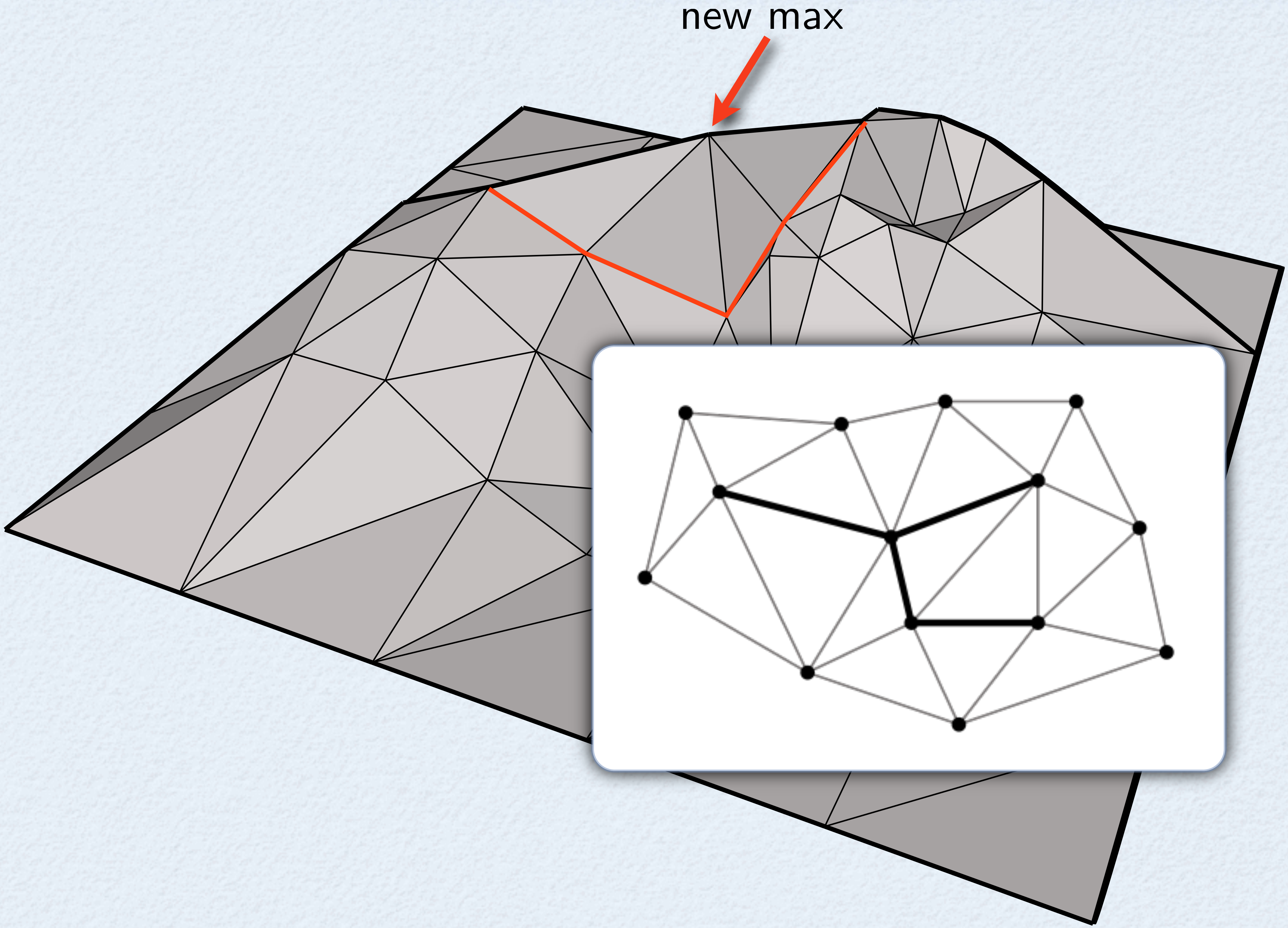
Turning a terrain into an elementary one by surgery



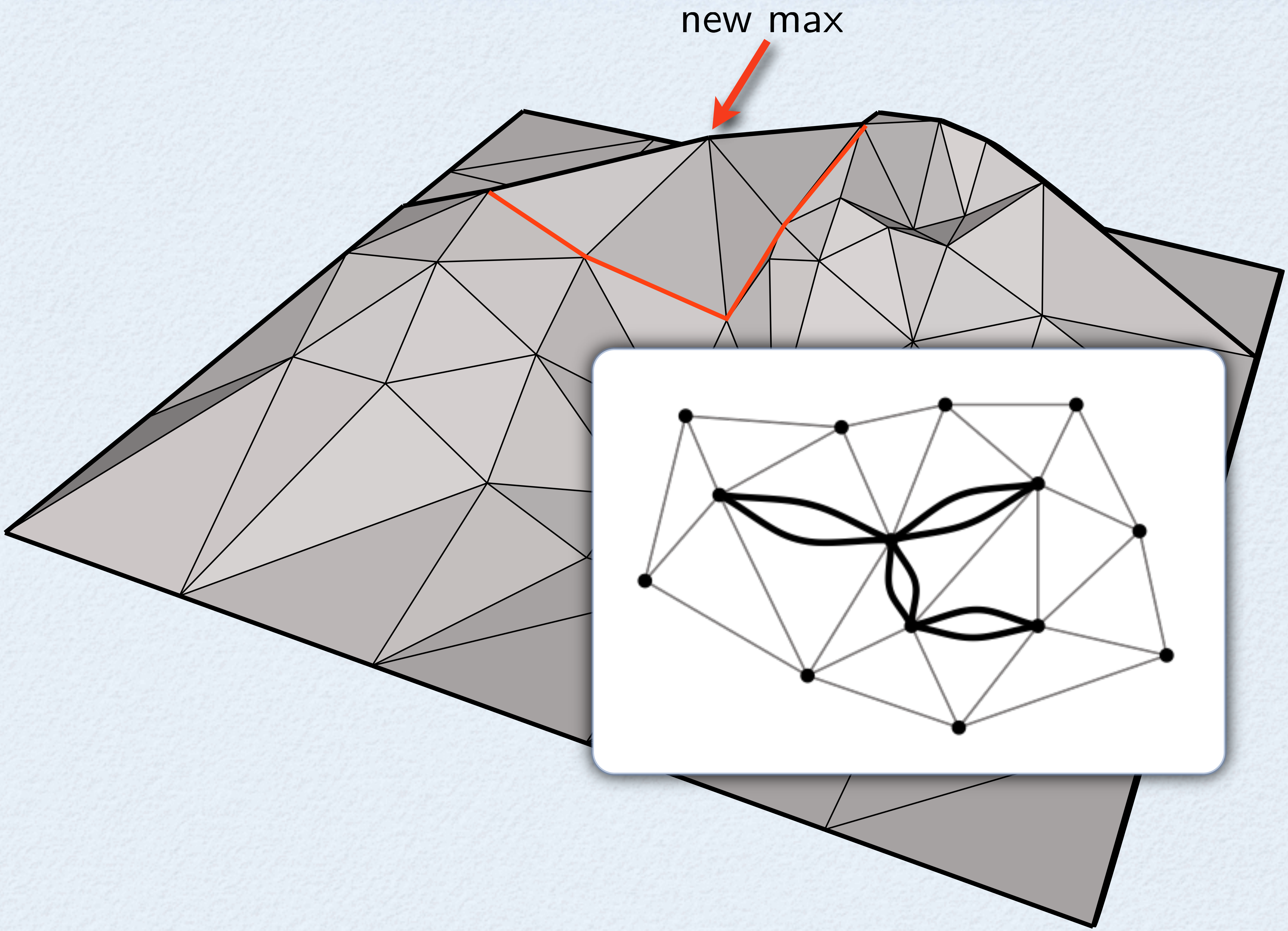
Turning a terrain into an elementary one by surgery



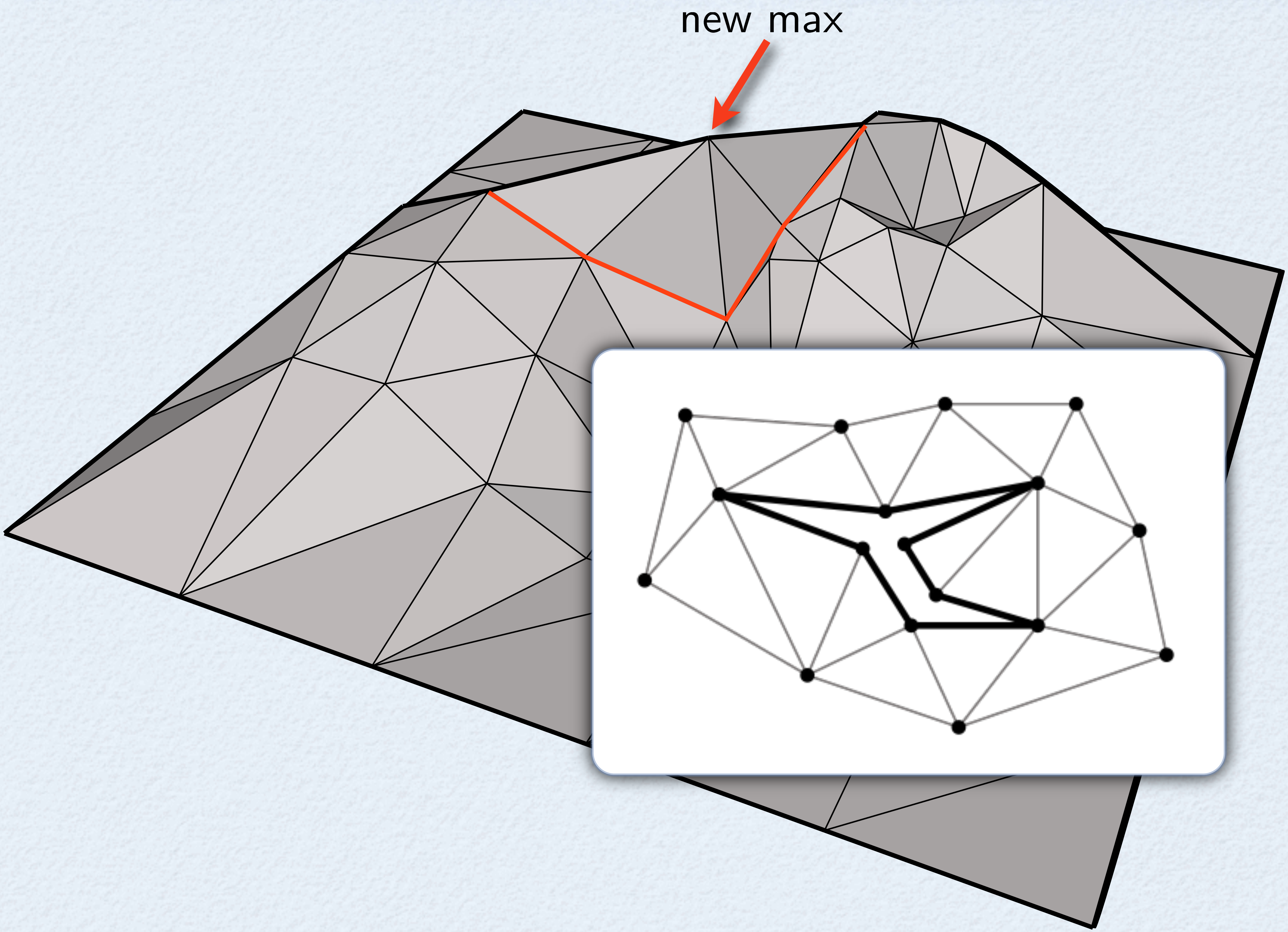
Turning a terrain into an elementary one by surgery



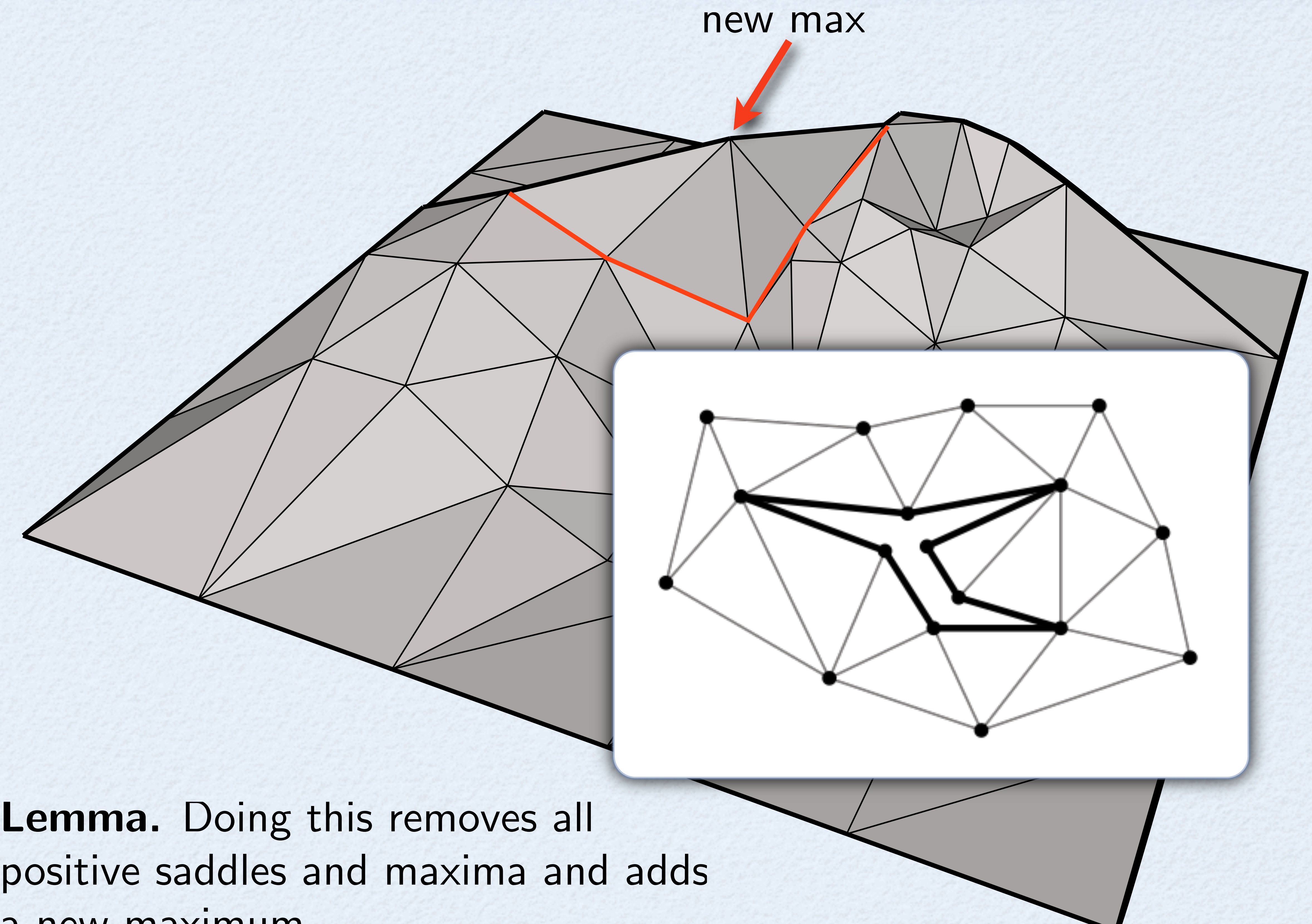
Turning a terrain into an elementary one by surgery



Turning a terrain into an elementary one by surgery



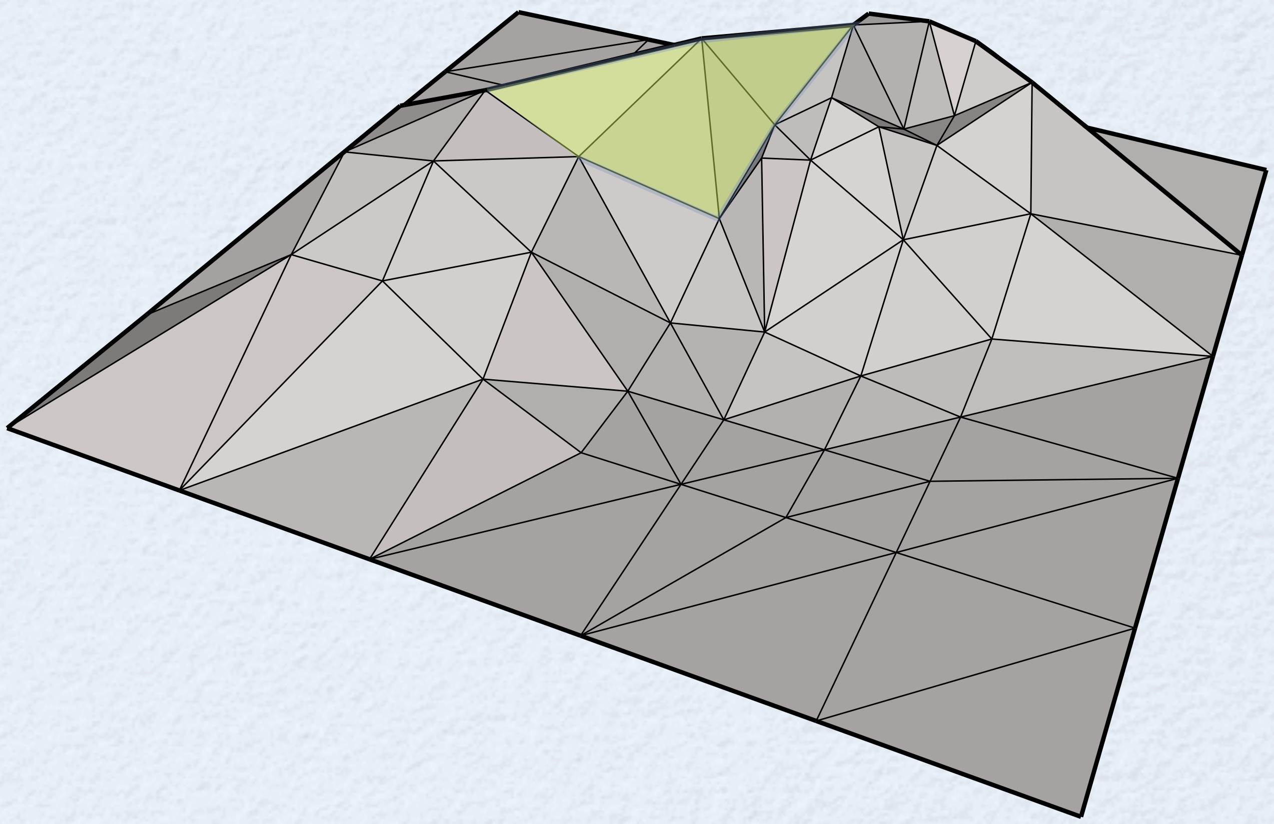
Turning a terrain into an elementary one by surgery



Lemma. Doing this removes all positive saddles and maxima and adds a new maximum.

What surgery does to contours?

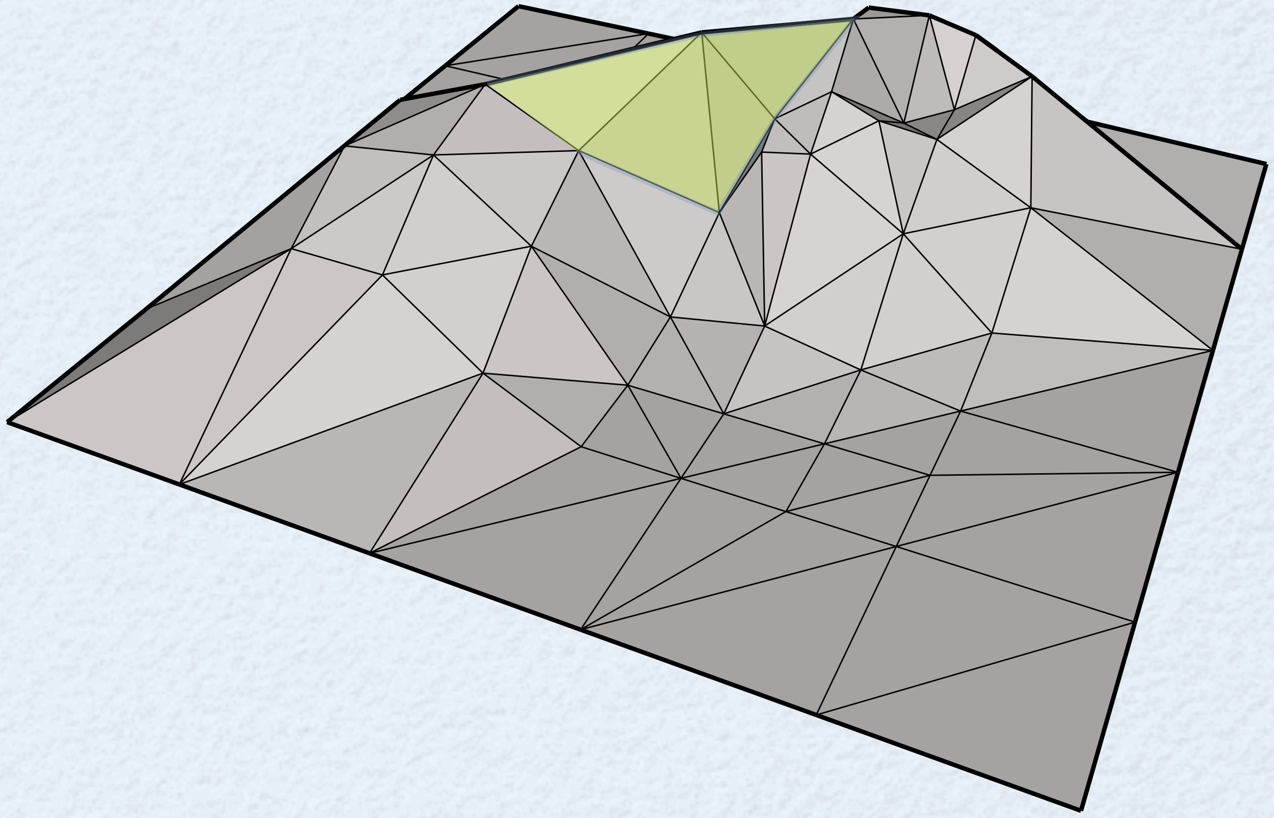
The elementary terrain $\mathbb{M}'$ has all the triangles of $\mathbb{M}$ plus some of "new" triangles.



What surgery does to contours?

The elementary terrain $\mathbb{M}'$ has all the triangles of $\mathbb{M}$ plus some of "new" triangles.

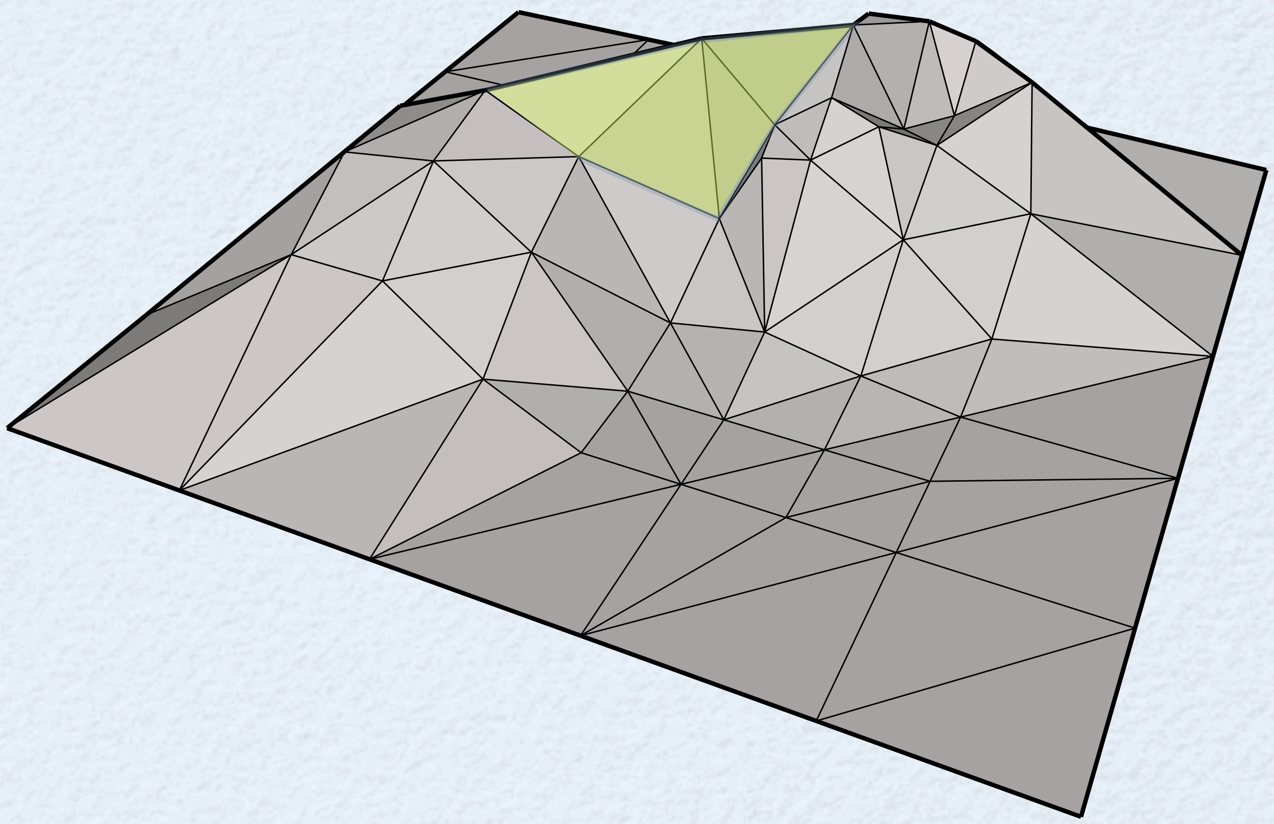
All contours of a level set of $\mathbb{M}$ are combined in a single contour in $\mathbb{M}'$



What surgery does to contours?

The elementary terrain $\mathbb{M}'$ has all the triangles of $\mathbb{M}$ plus some of "new" triangles.

All contours of a level set of $\mathbb{M}$ are combined in a single contour in $\mathbb{M}'$



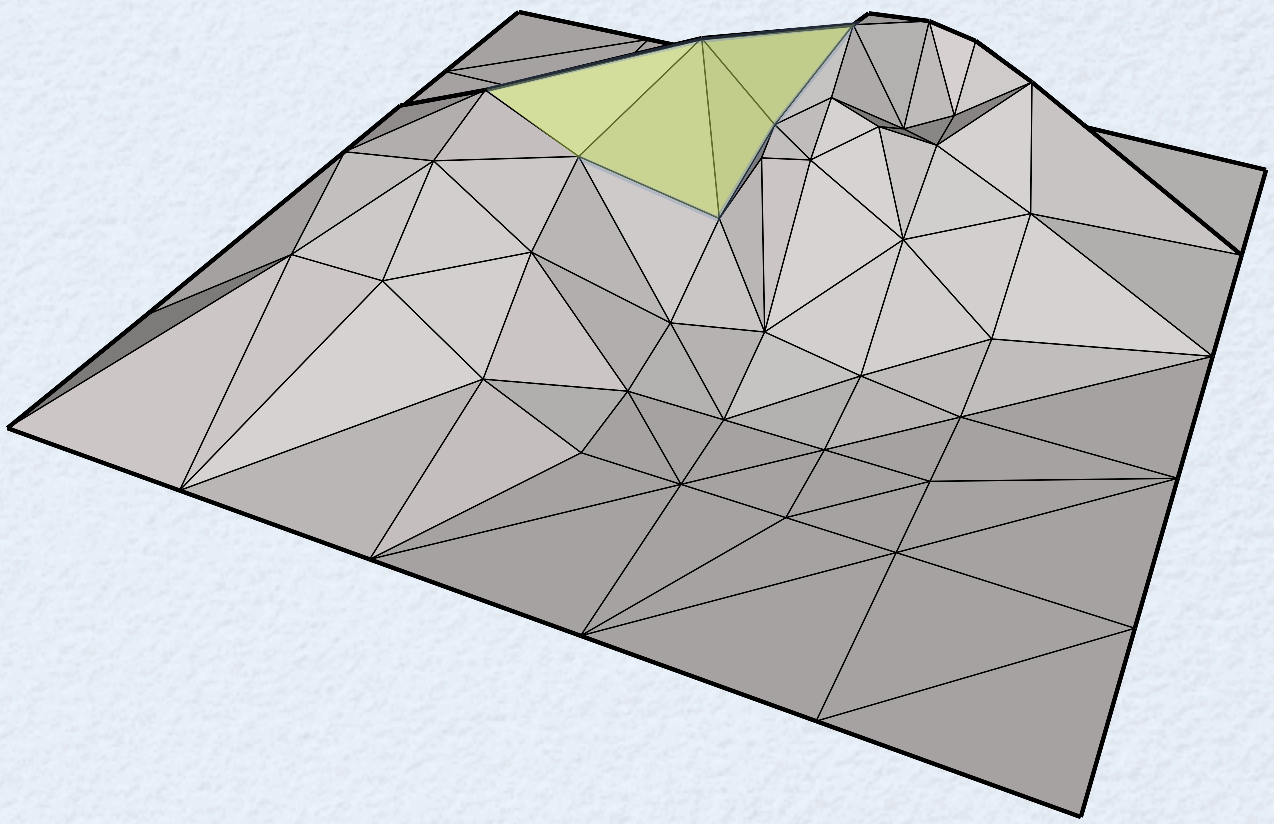
Theorem. In a contour of $\mathbb{M}'$, corresponding contours of $\mathbb{M}$ are broken (by segments from new triangles) in a nested (parenthesized) manner.

$$a_1a_2 * * b_1 * * * c_1c_2c_3 * * * * b_2b_3 * d_1d_2 * * b_4 * * a_3a_4a_5$$

What surgery does to contours?

The elementary terrain $\mathbb{M}'$ has all the triangles of $\mathbb{M}$ plus some of "new" triangles.

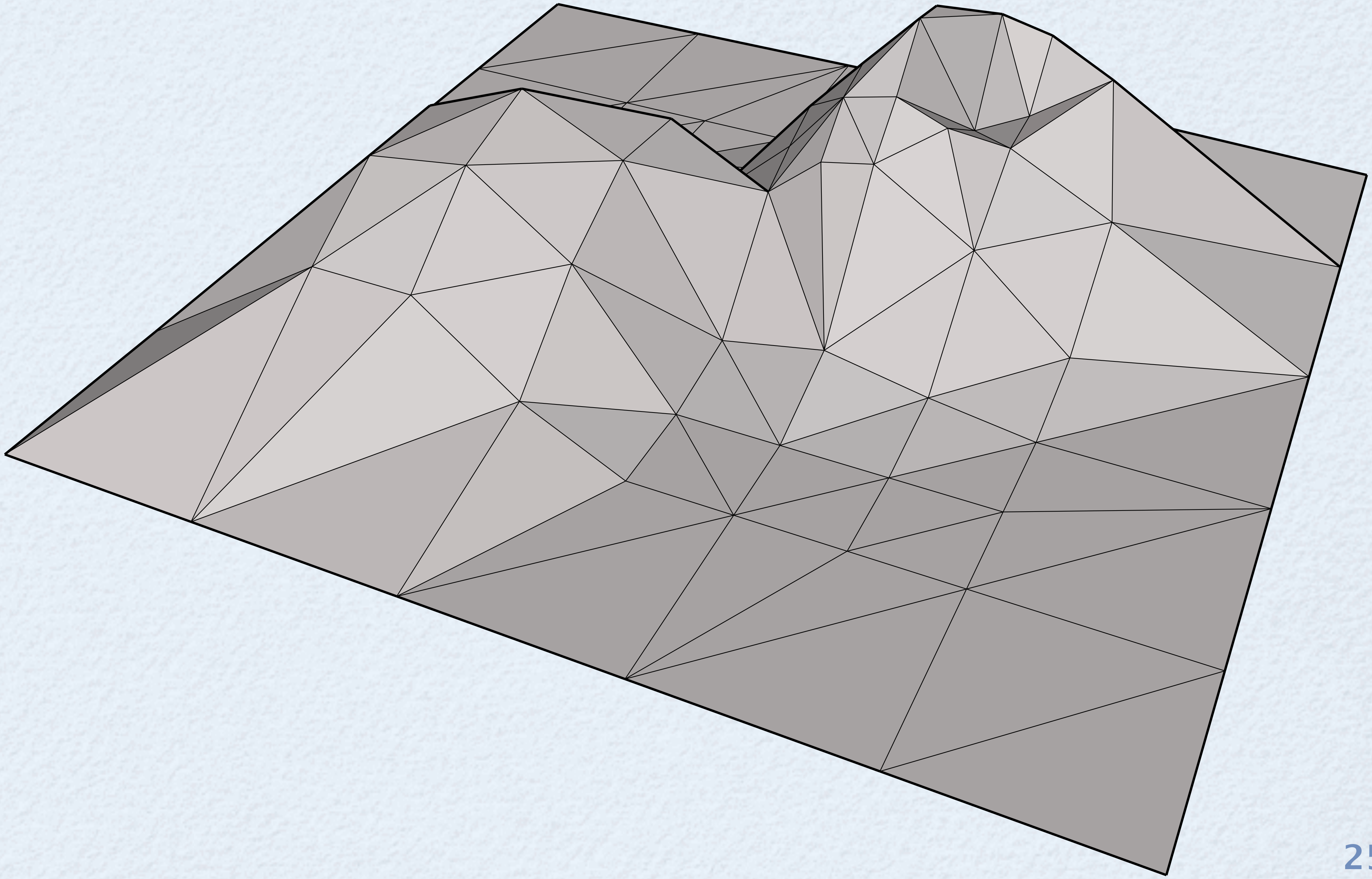
All contours of a level set of $\mathbb{M}$ are combined in a single contour in $\mathbb{M}'$



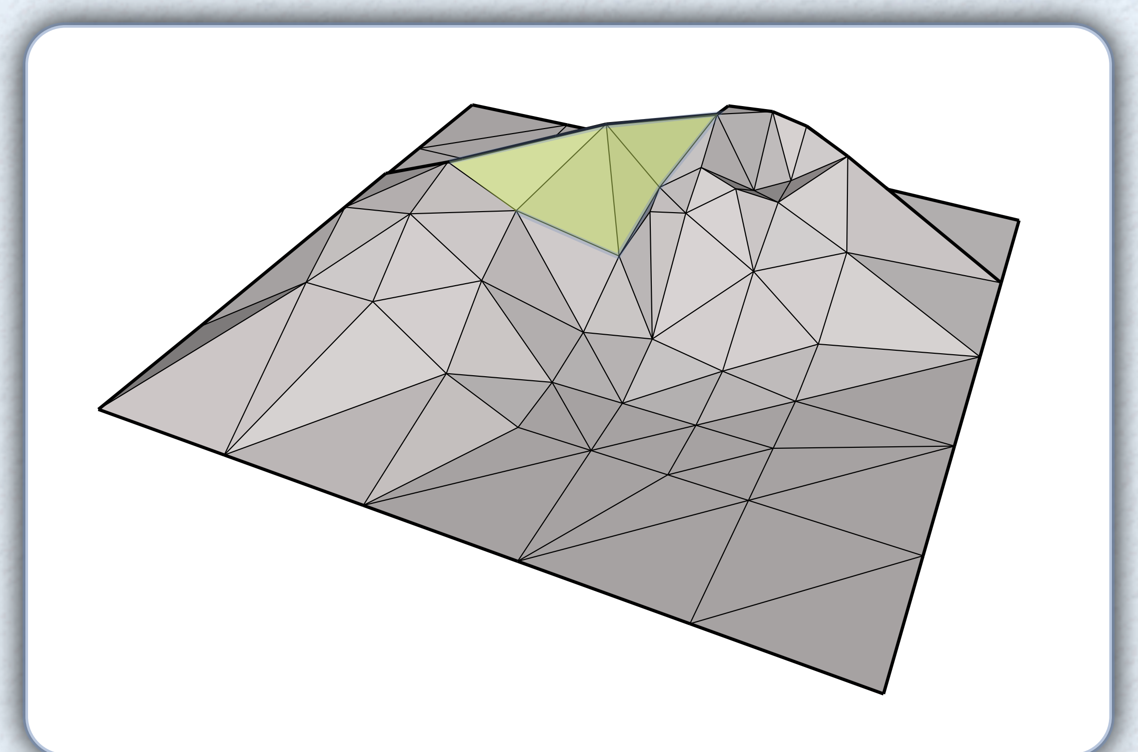
Theorem. In a contour of $\mathbb{M}'$, corresponding contours of $\mathbb{M}$ are broken (by segments from new triangles) in a nested (parenthesized) manner.

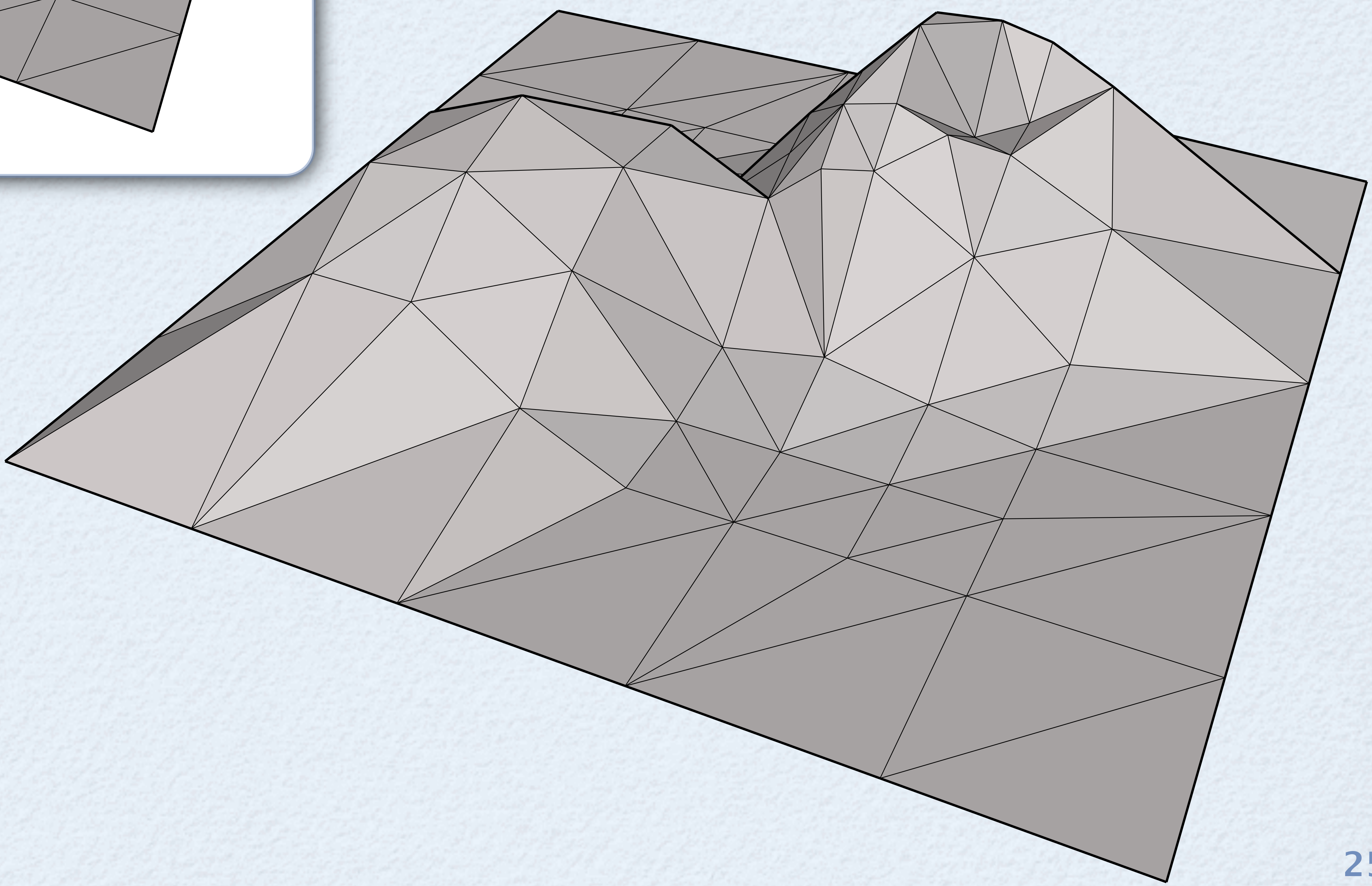
$$[a_1 a_2 * * [b_1 * * * [c_1 c_2 c_3] * * * * b_2 b_3 * [d_1 d_2] * * b_4] * * a_3 a_4 a_5]$$

Where is nesting coming from?

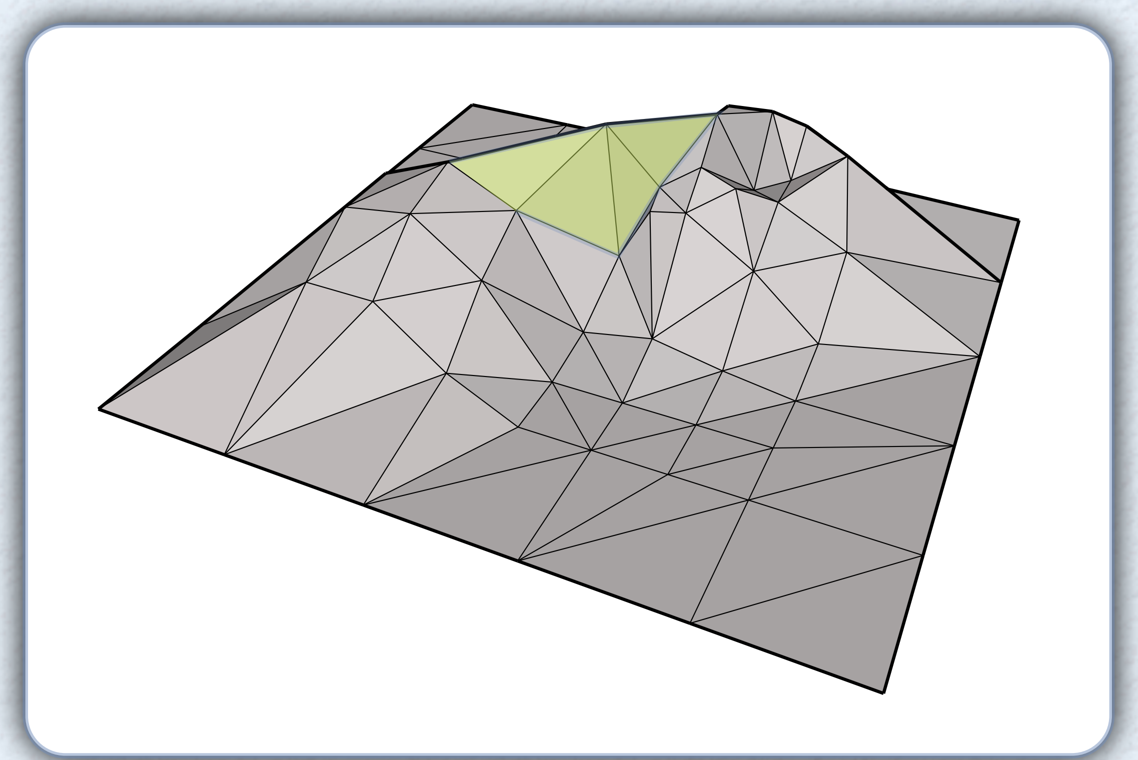


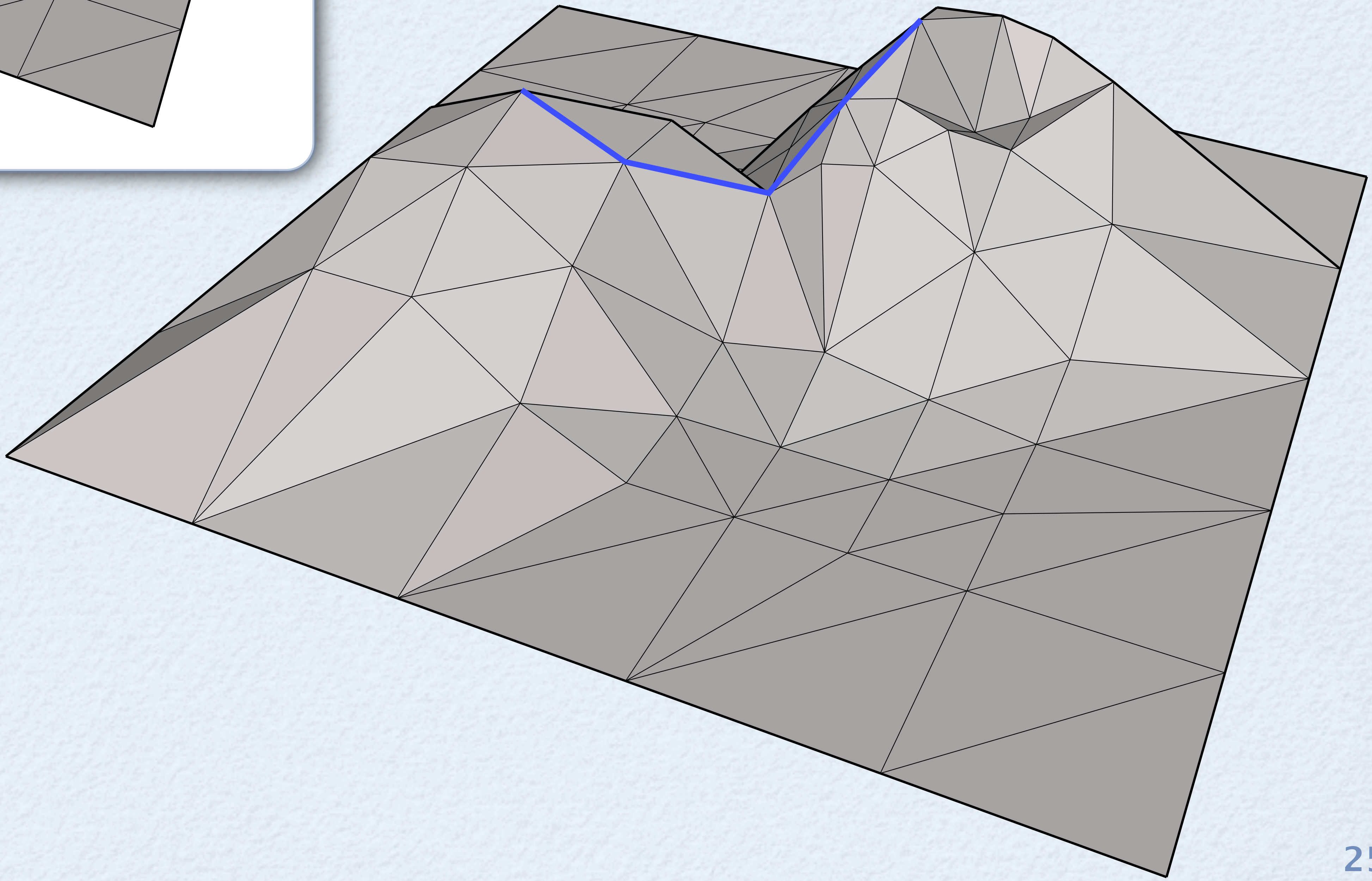
Where is nesting coming from?



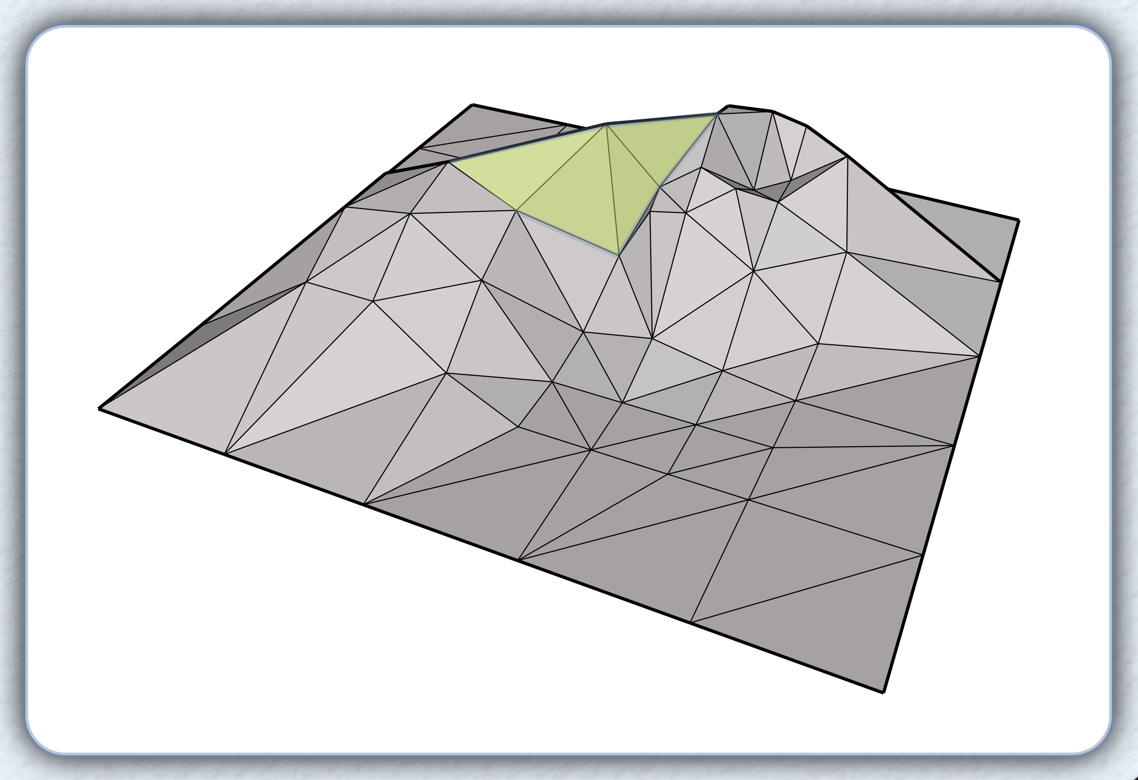


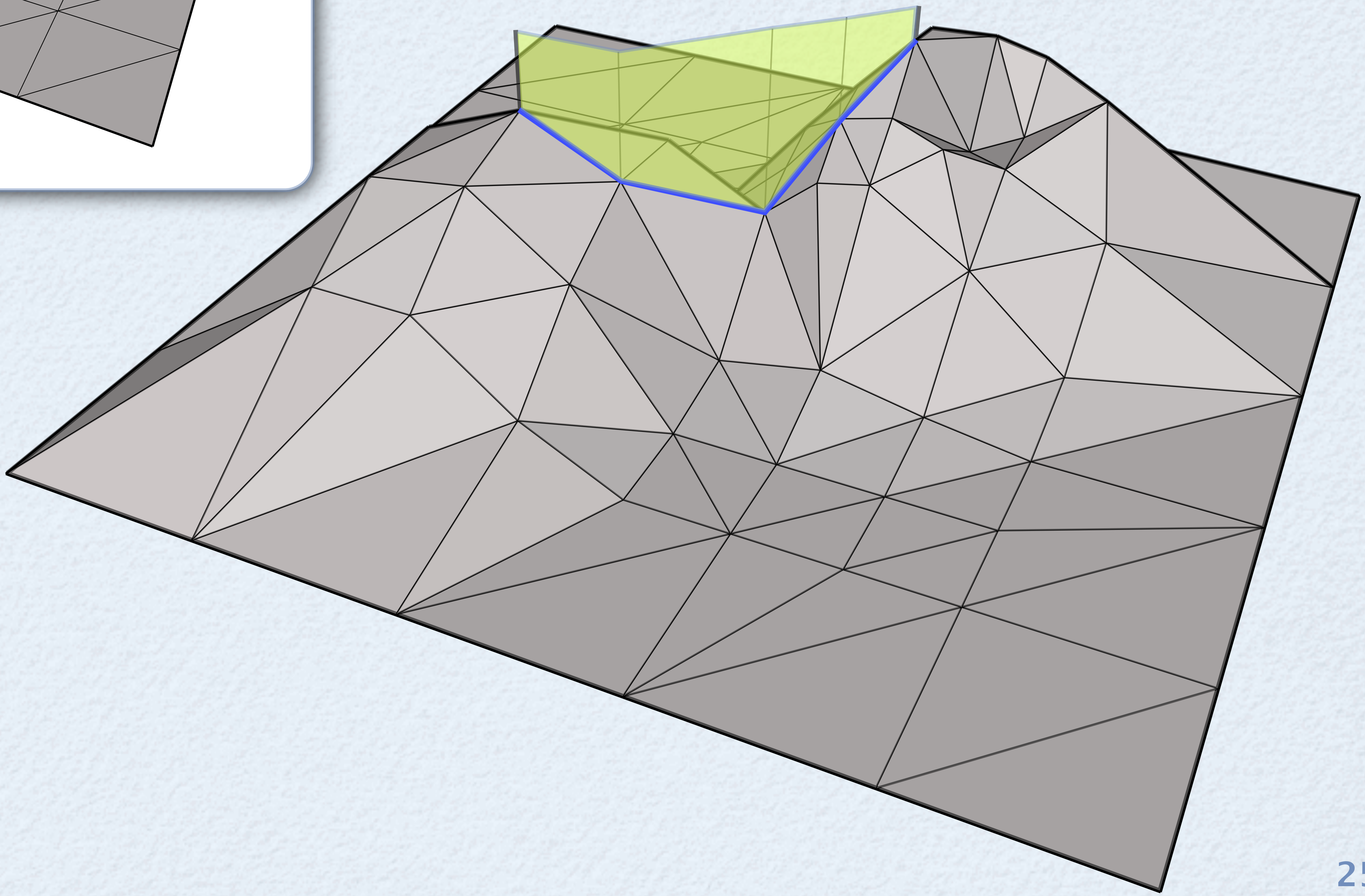
Where is nesting coming from?



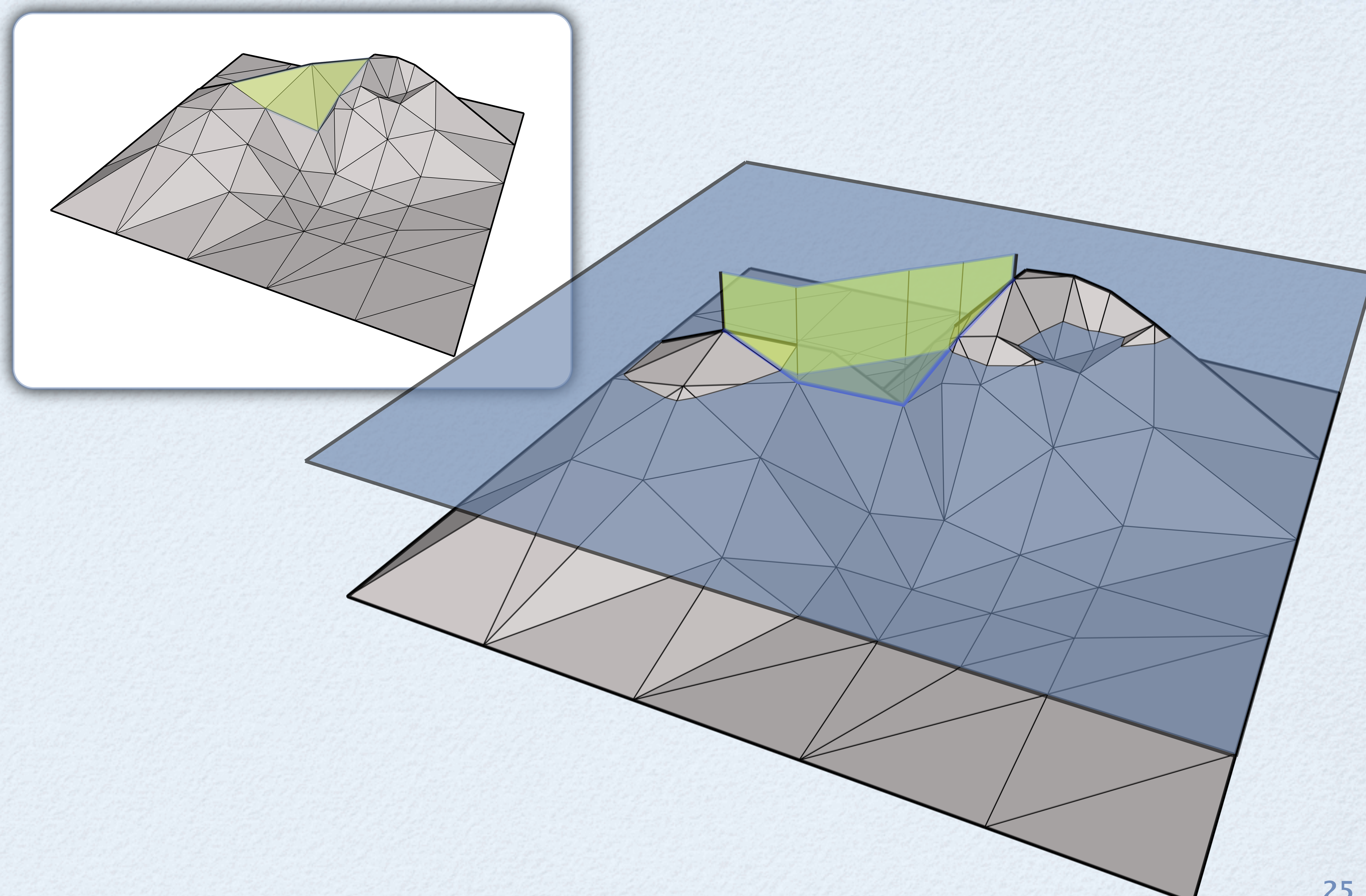


Where is nesting coming from?

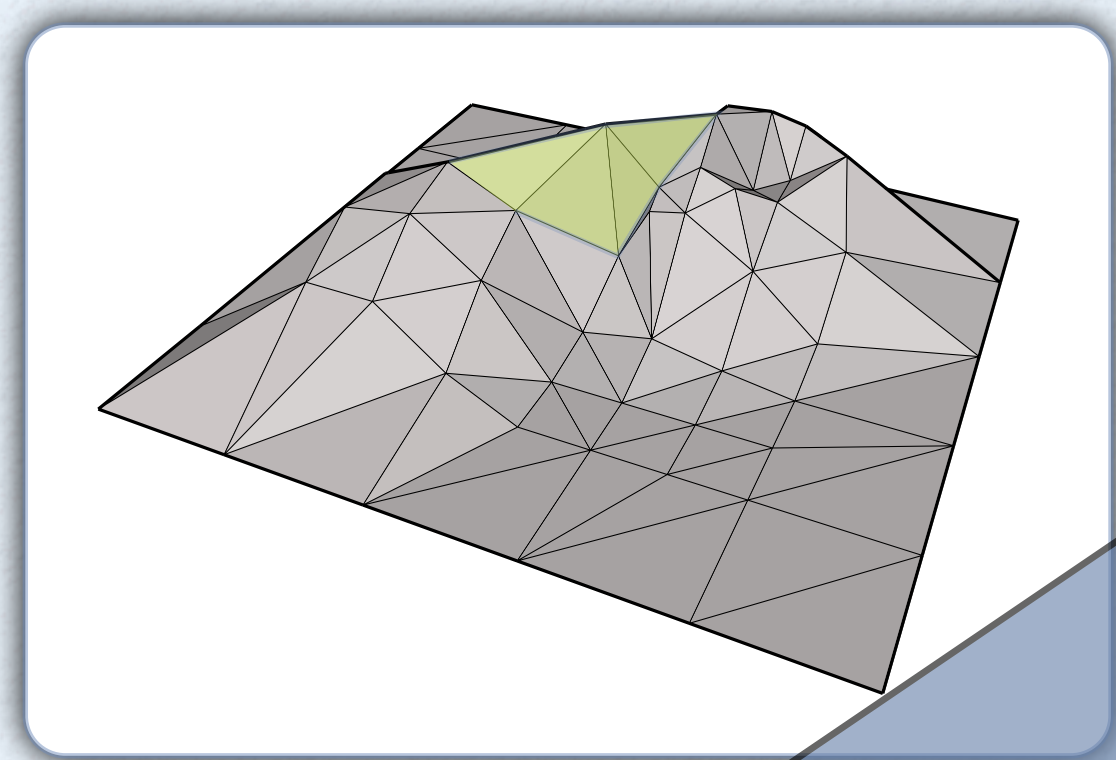


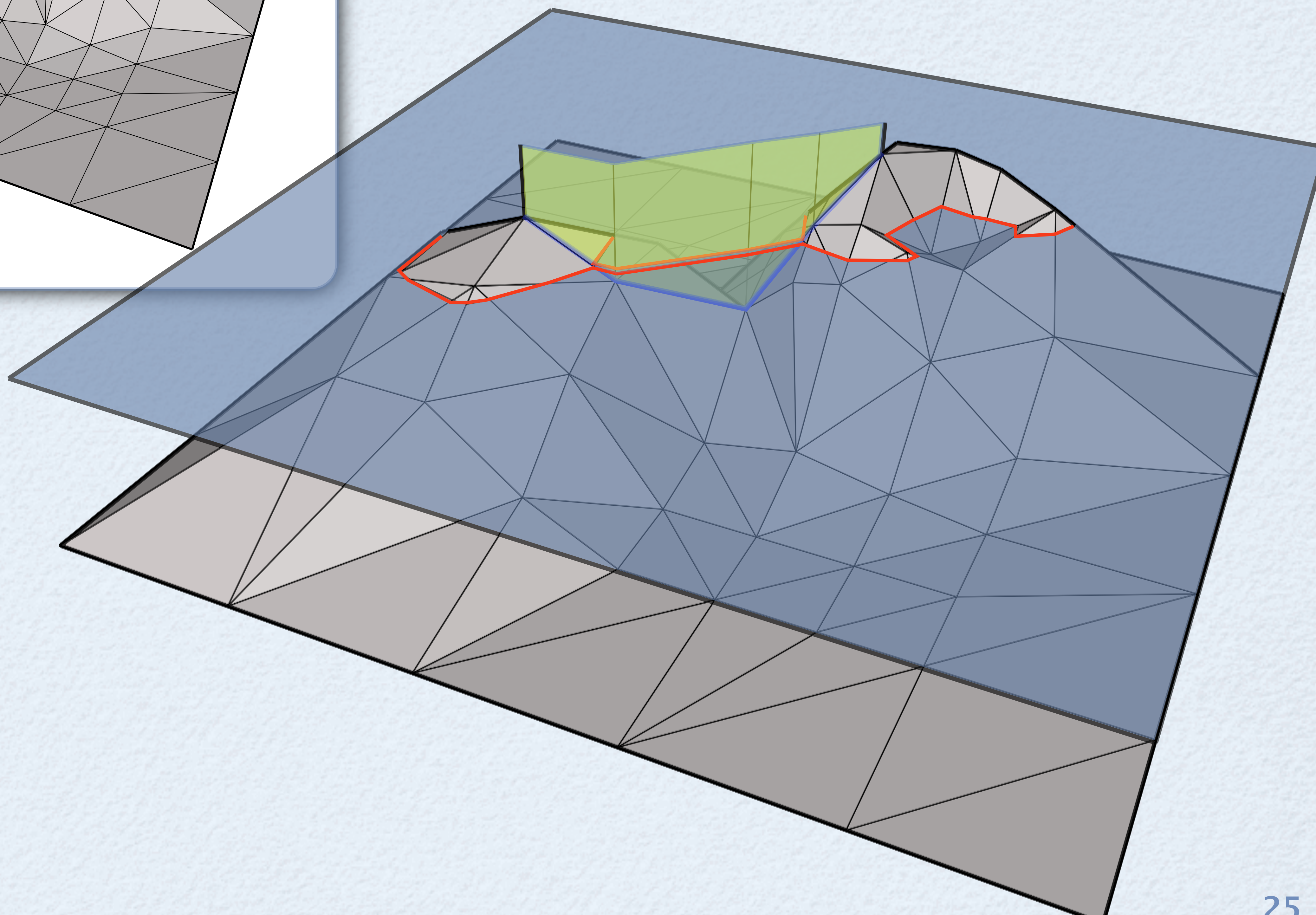


Where is nesting coming from?



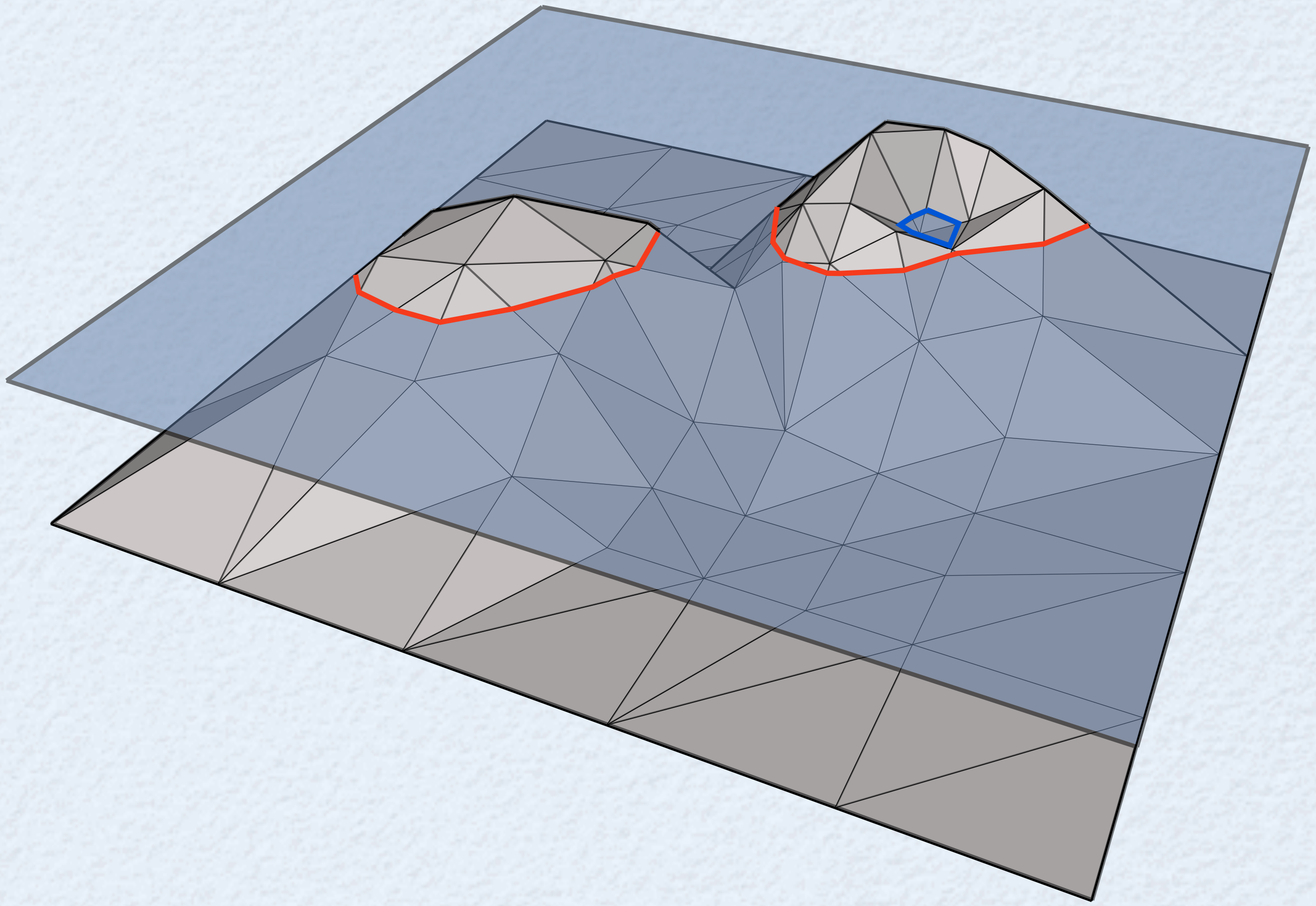
Where is nesting coming from?





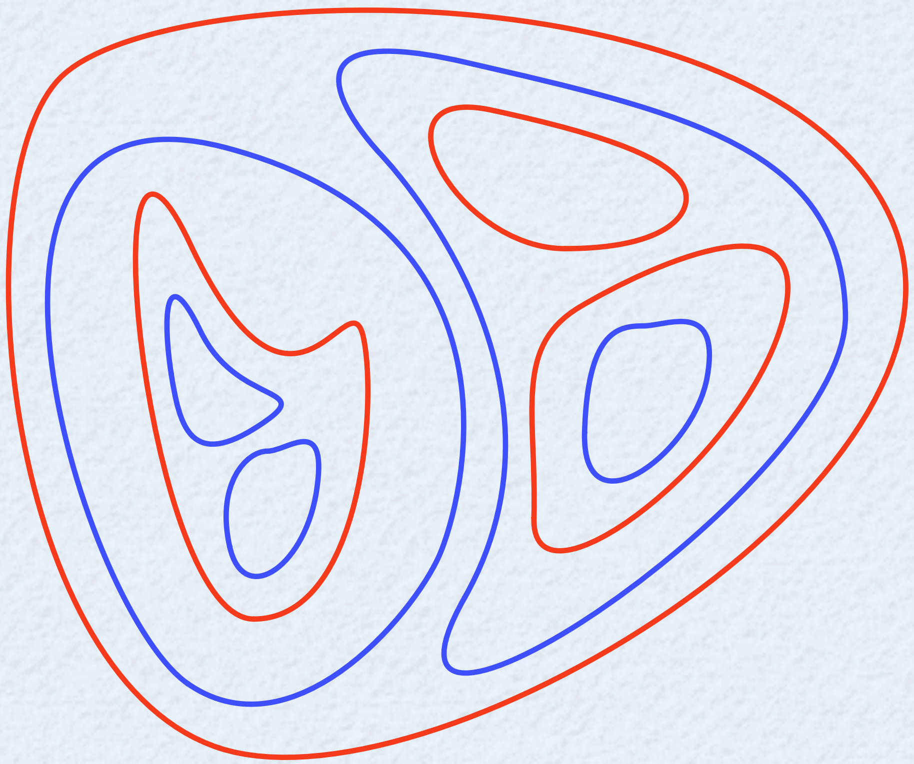
Red and Blue Contours

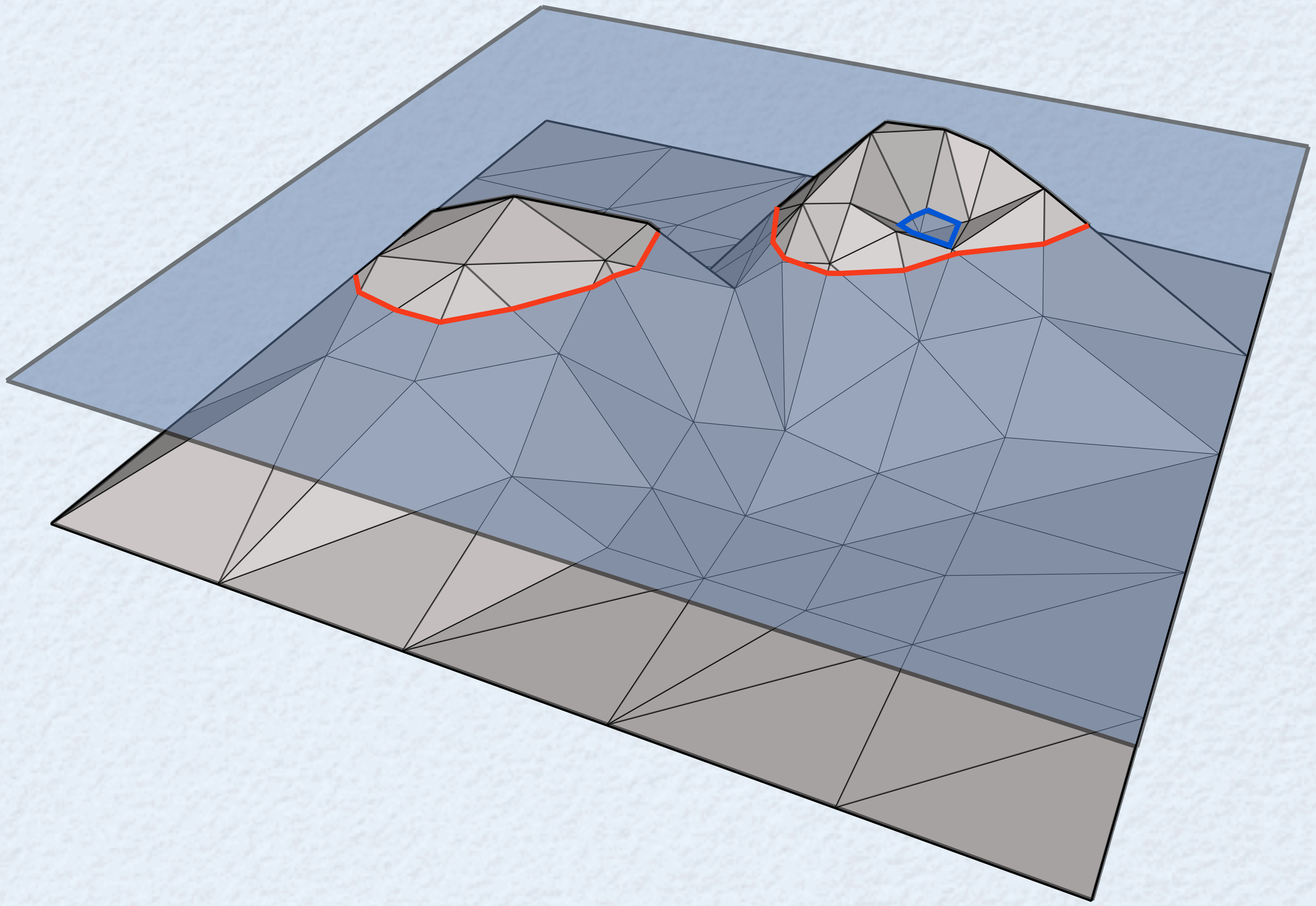
A contour is red (blue) if "locally" the sublevel set is in its outside (inside).



Red and Blue Contours

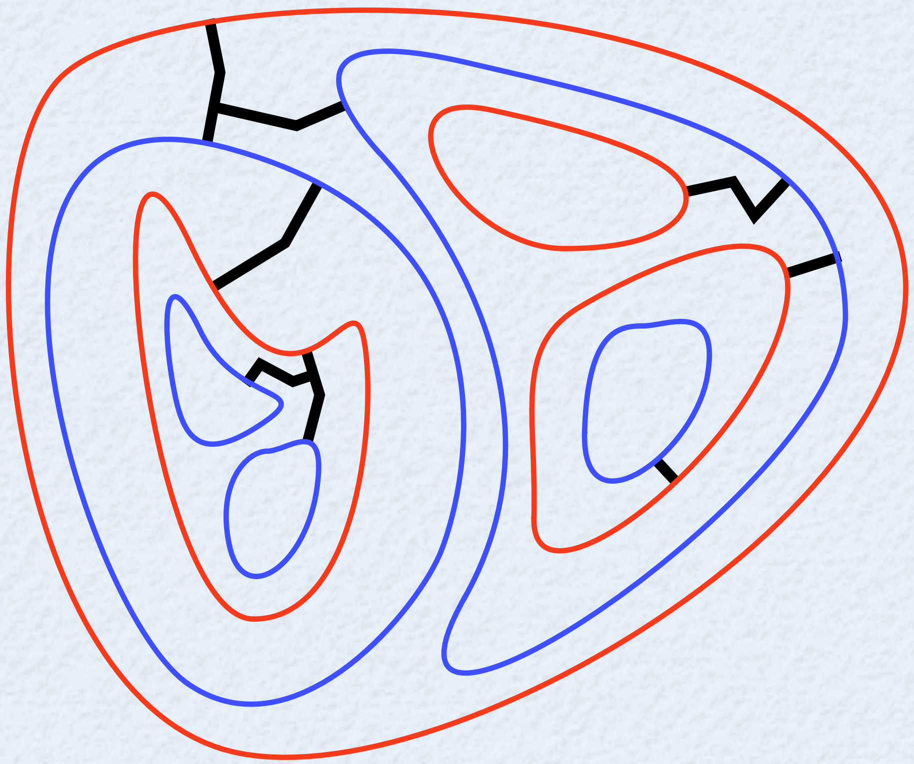
A contour is red (blue) if "locally" the sublevel set is in its outside (inside).

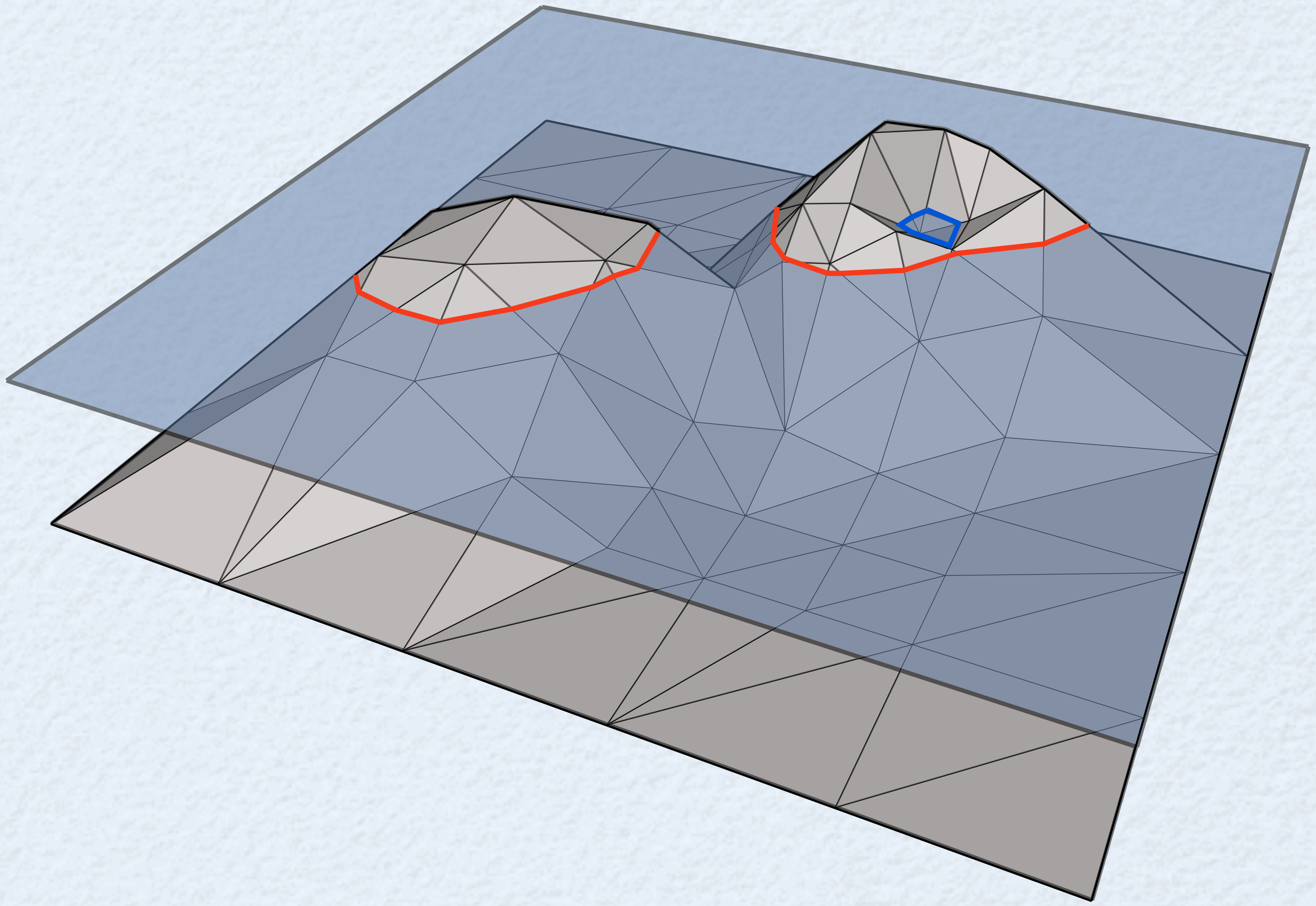




Red and Blue Contours

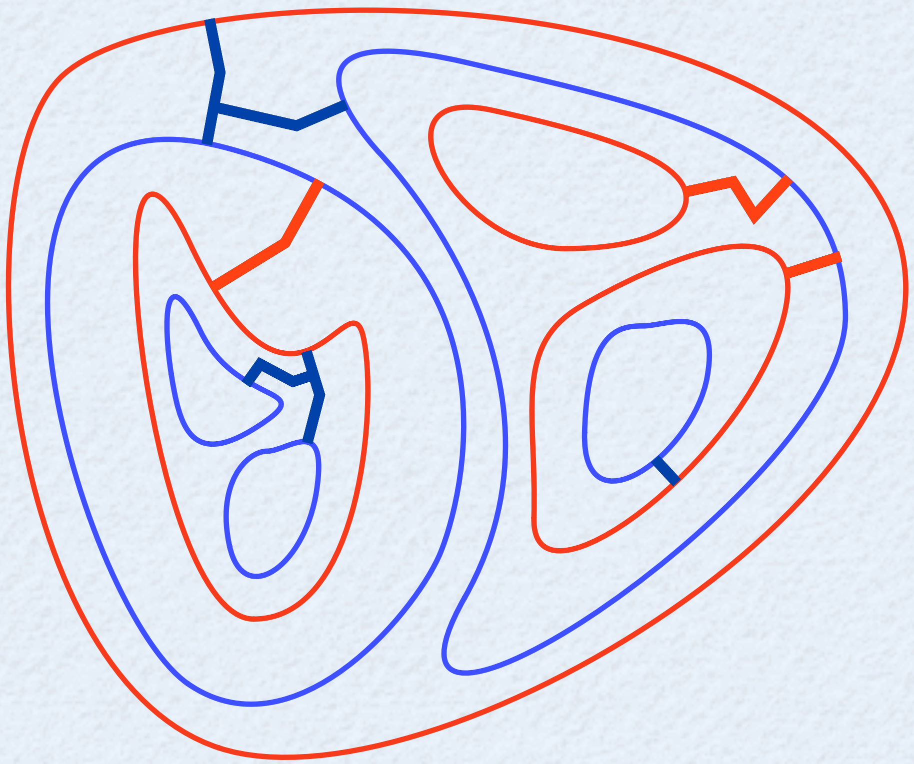
A contour is red (blue) if "locally" the sublevel set is in its outside (inside).

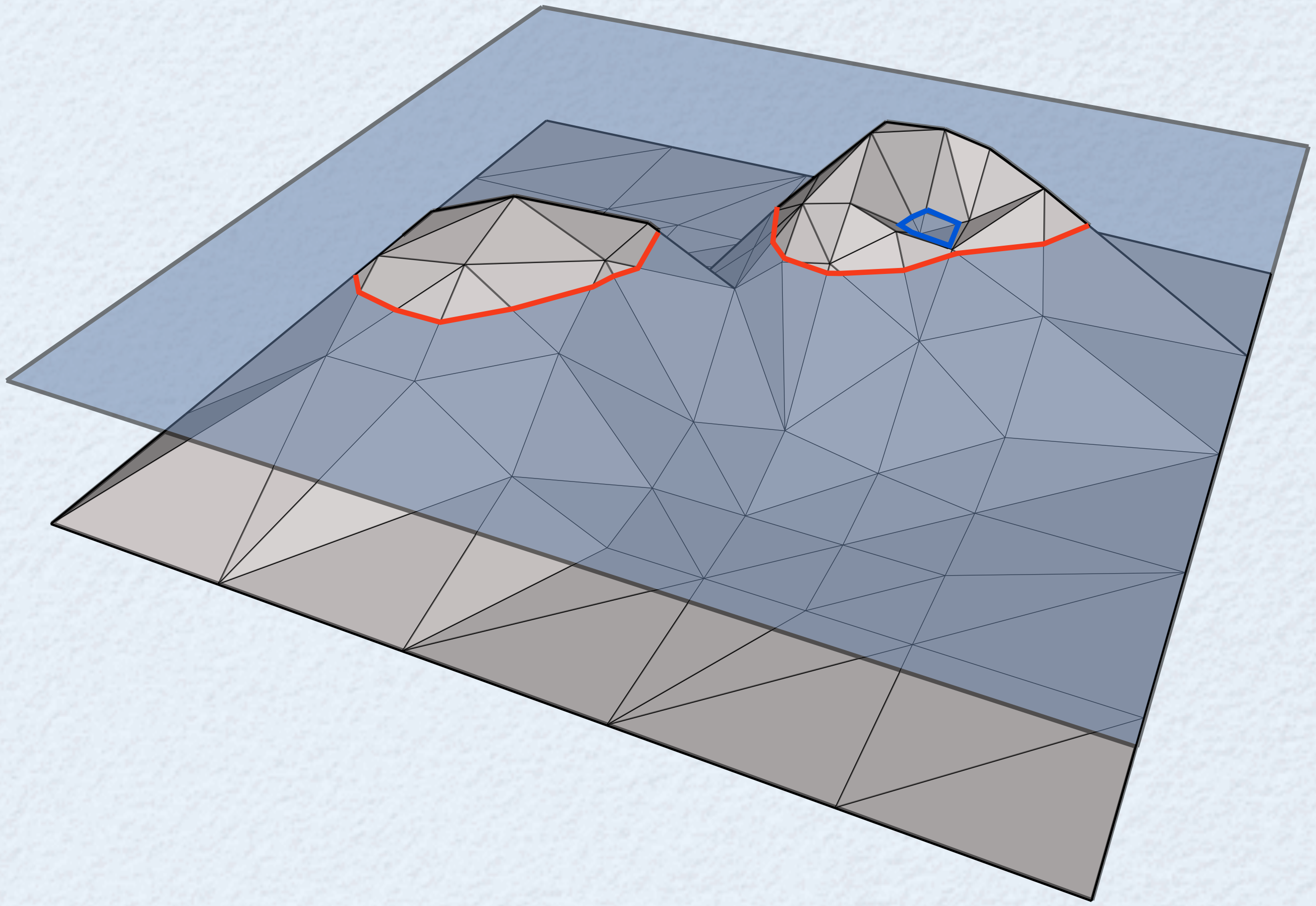




Red and Blue Contours

A contour is red (blue) if "locally" the sublevel set is in its outside (inside).

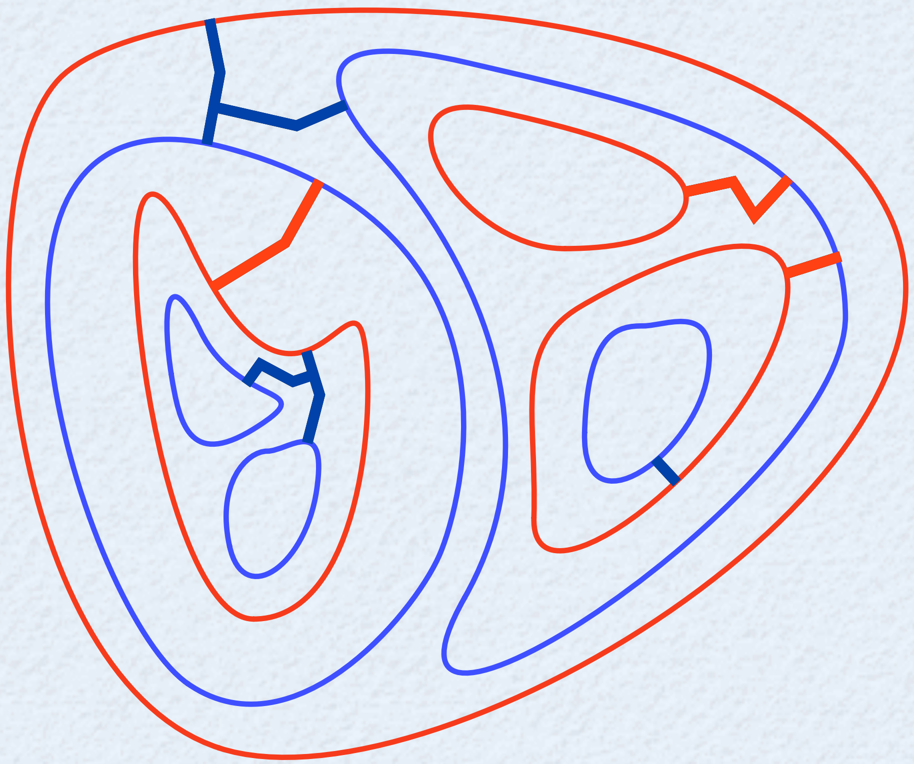


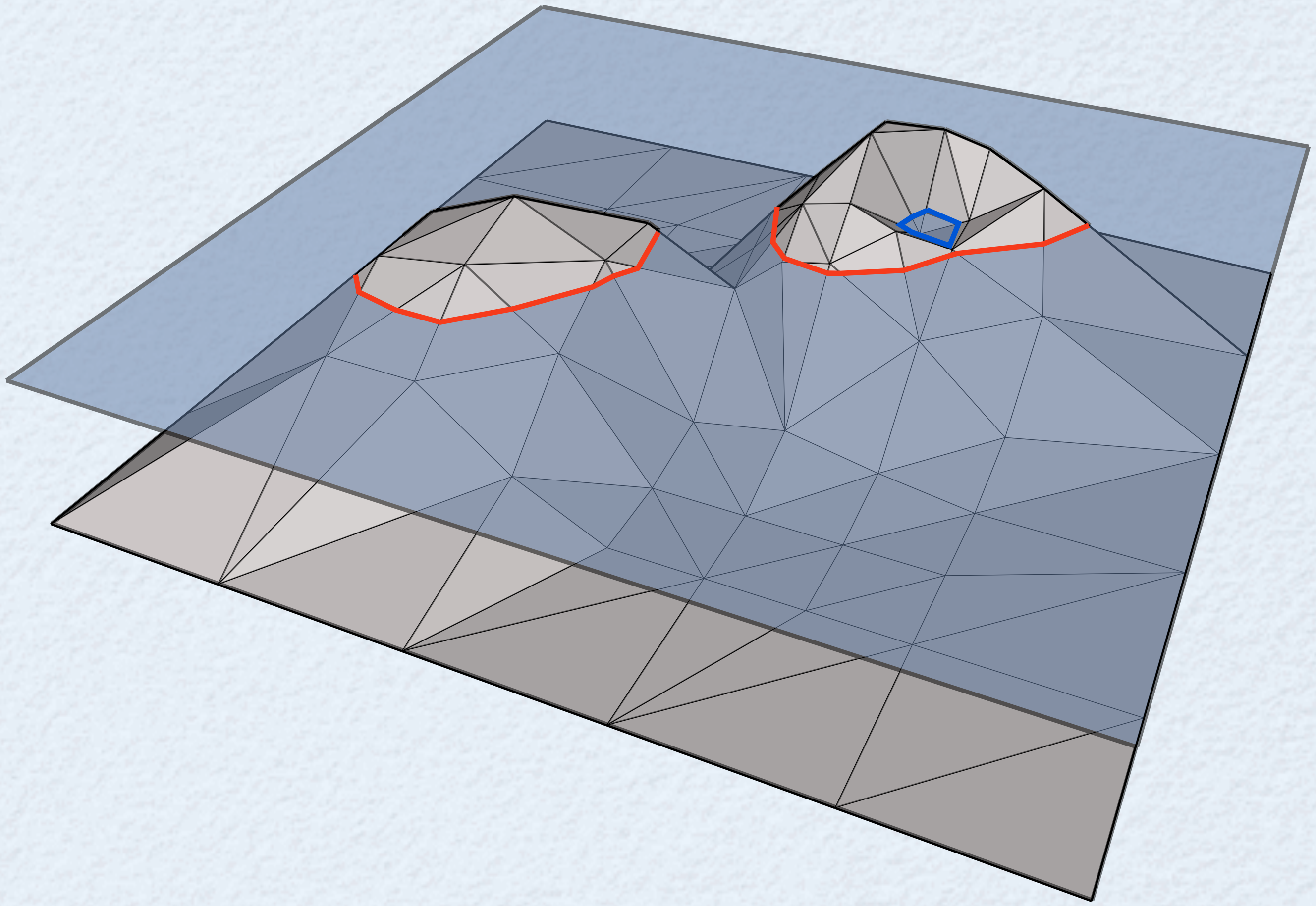


Theorem. Only the red tree connects a blue contour and its red children.

Red and Blue Contours

A contour is red (blue) if "locally" the sublevel set is in its outside (inside).



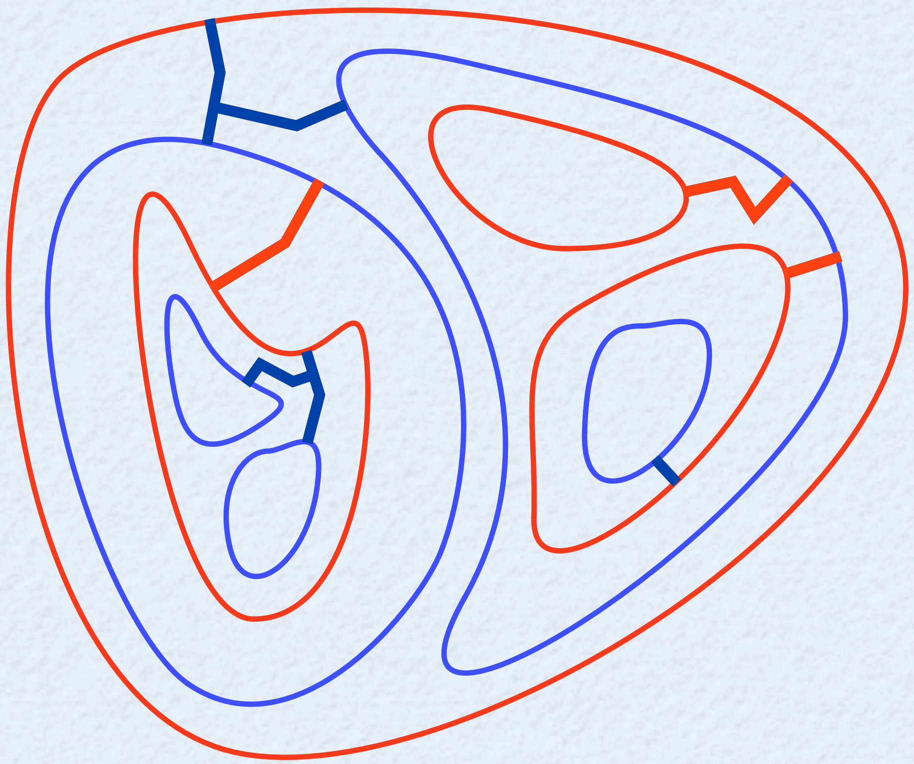


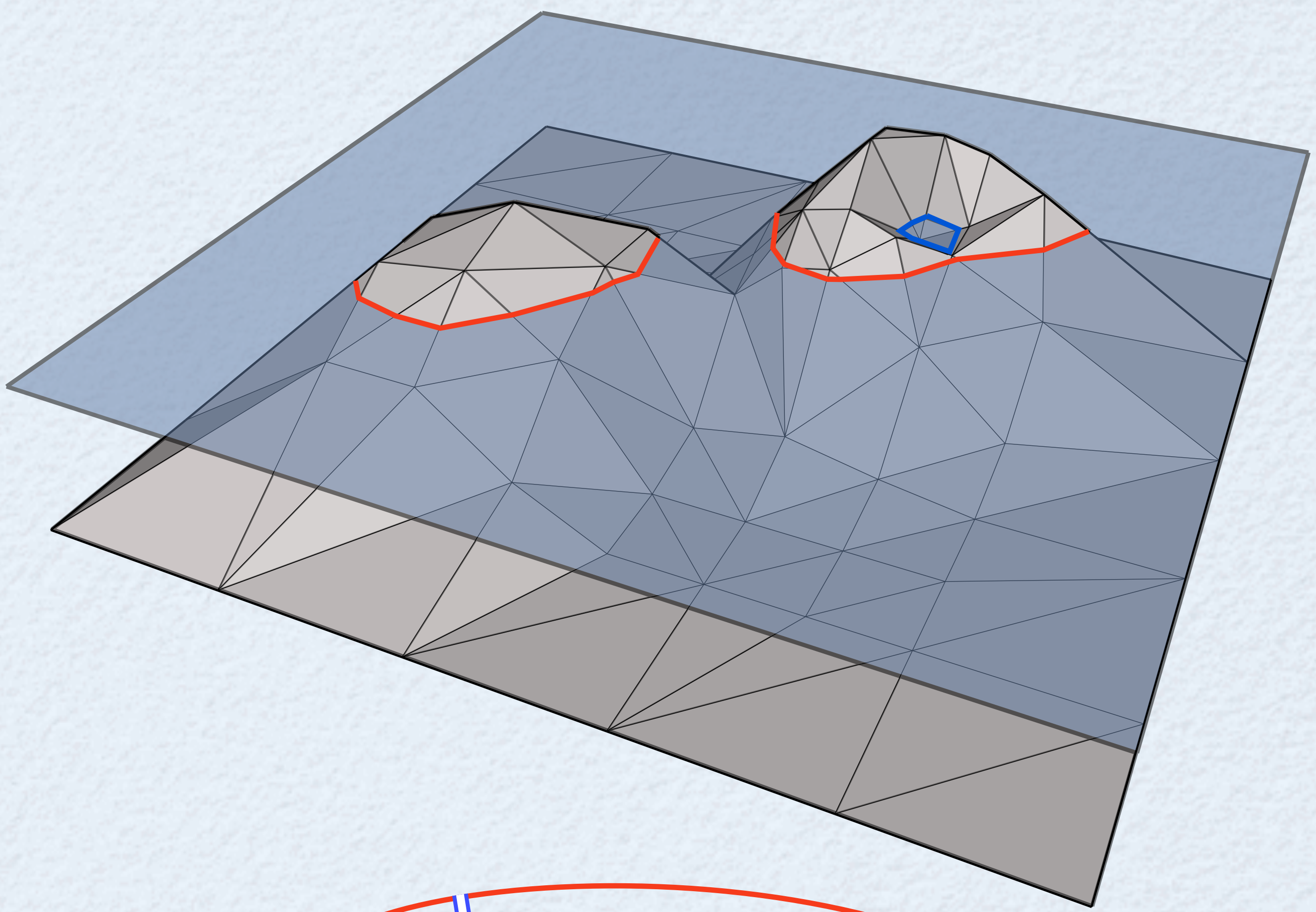
Theorem. Only the red tree connects a blue contour and its red children.

Theorem. If we contract each contour to a point, the result is a tree.

Red and Blue Contours

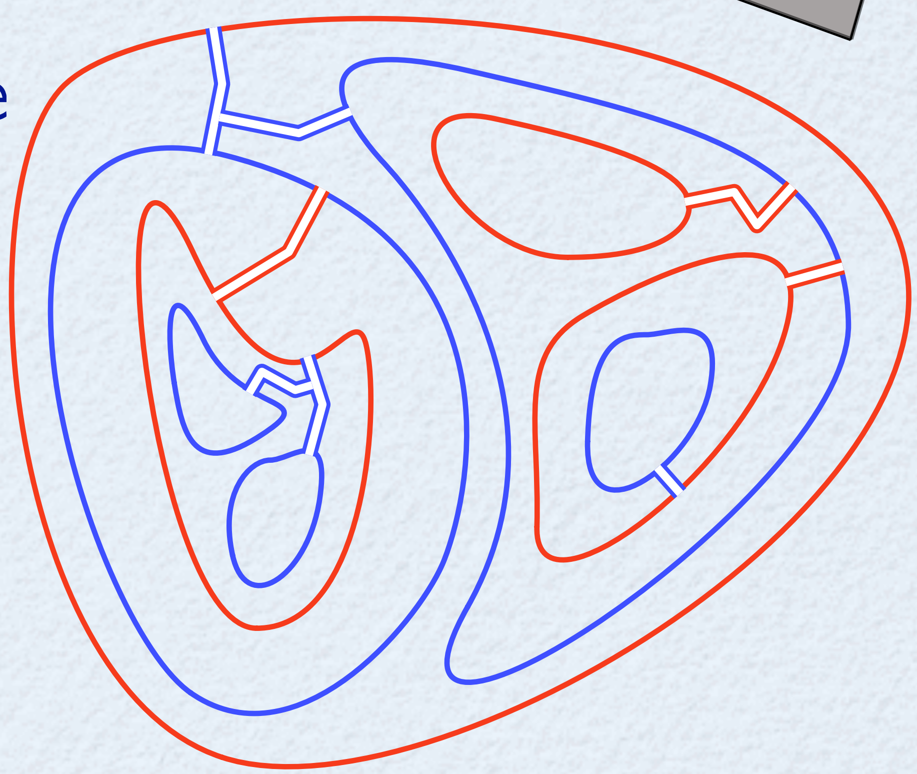
A contour is red (blue) if "locally" the sublevel set is in its outside (inside).





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

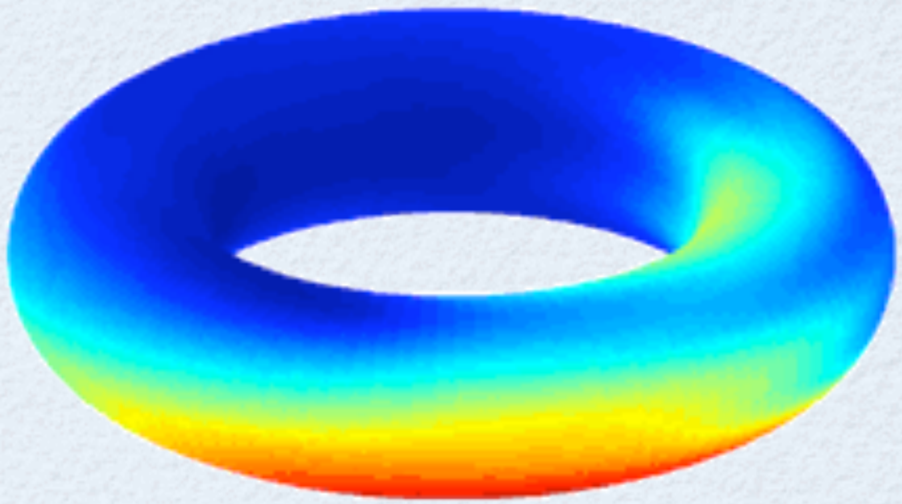
Theorem. If we contract each contour to a point, the result is a tree.



Remarks and Open Problems

The embedding of the terrain is unimportant: the set of triangles that intersect a level set only depends on function value on vertices.

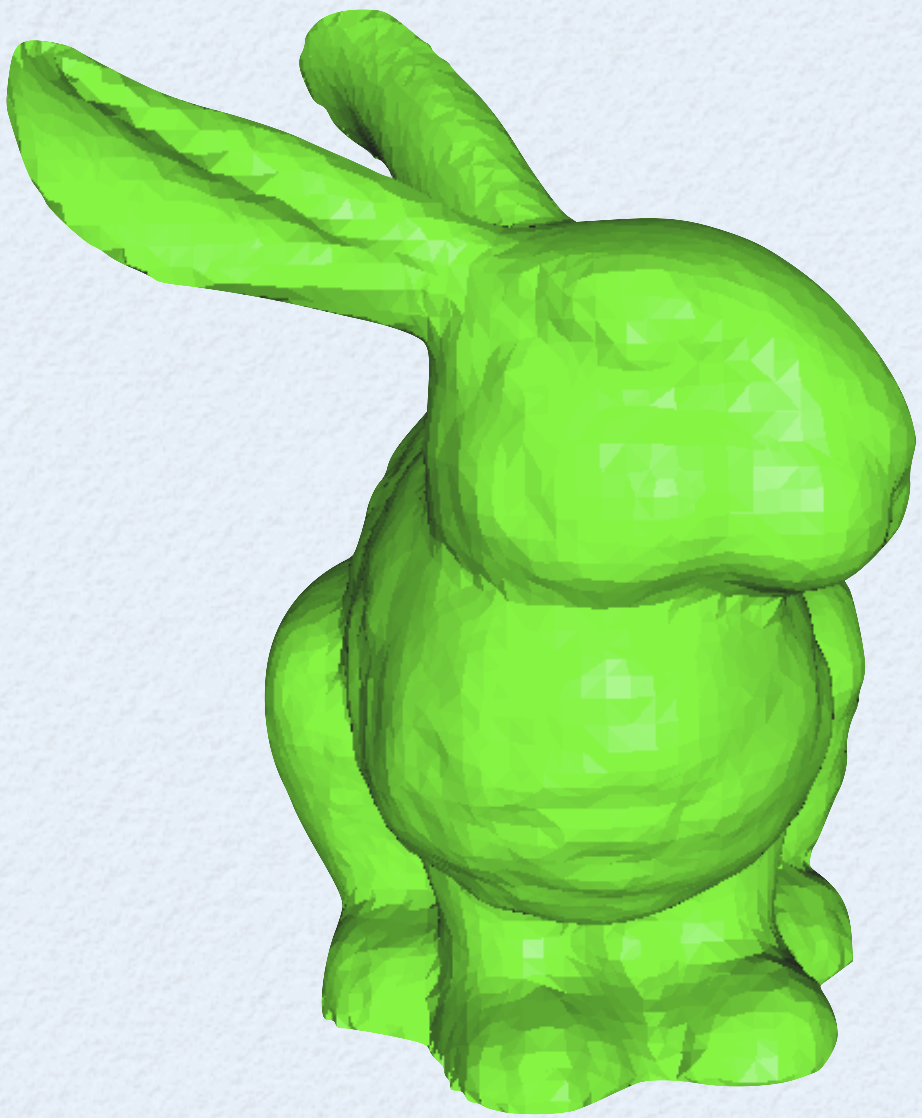
What about surfaces of genus ≥ 1? (Orientable or not)

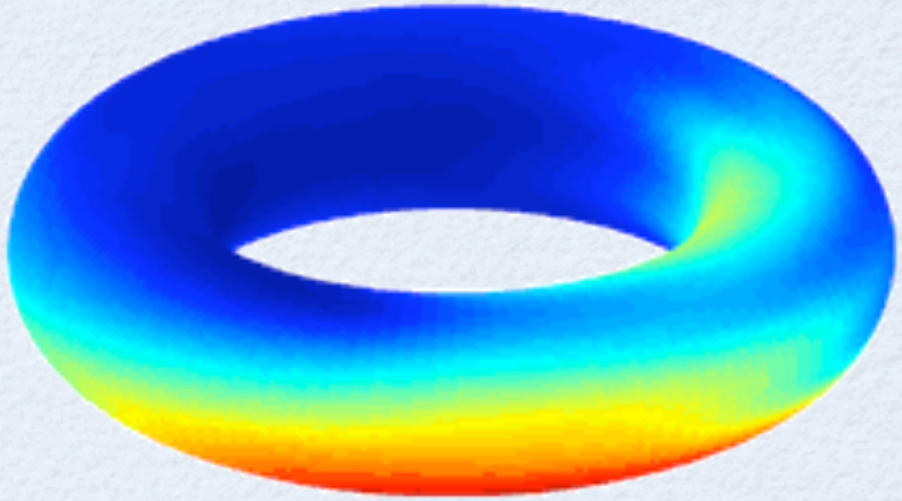


Remarks and Open Problems

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 ≥ 1? (Orientable or not)



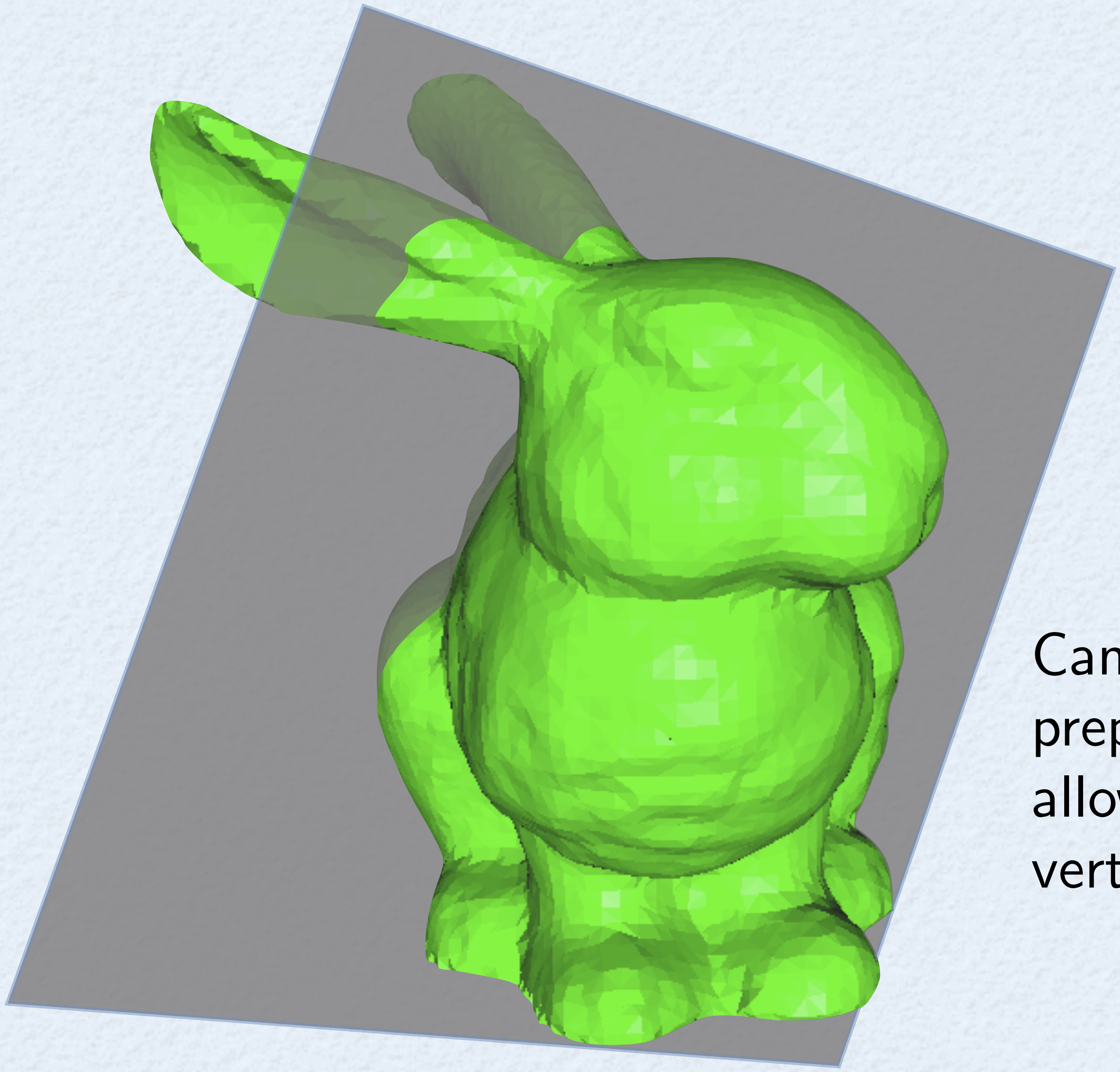


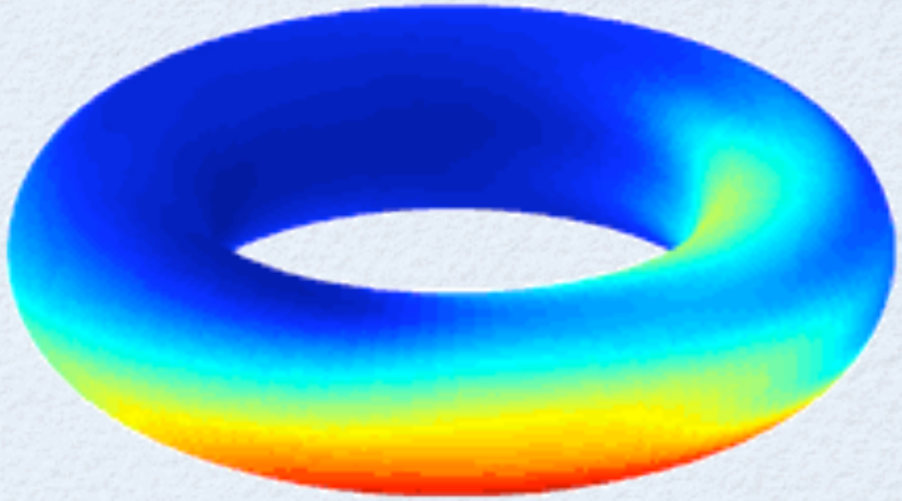
Can a topological sphere $\mathbb{M}$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?

Remarks and Open Problems

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 ≥ 1? (Orientable or not)



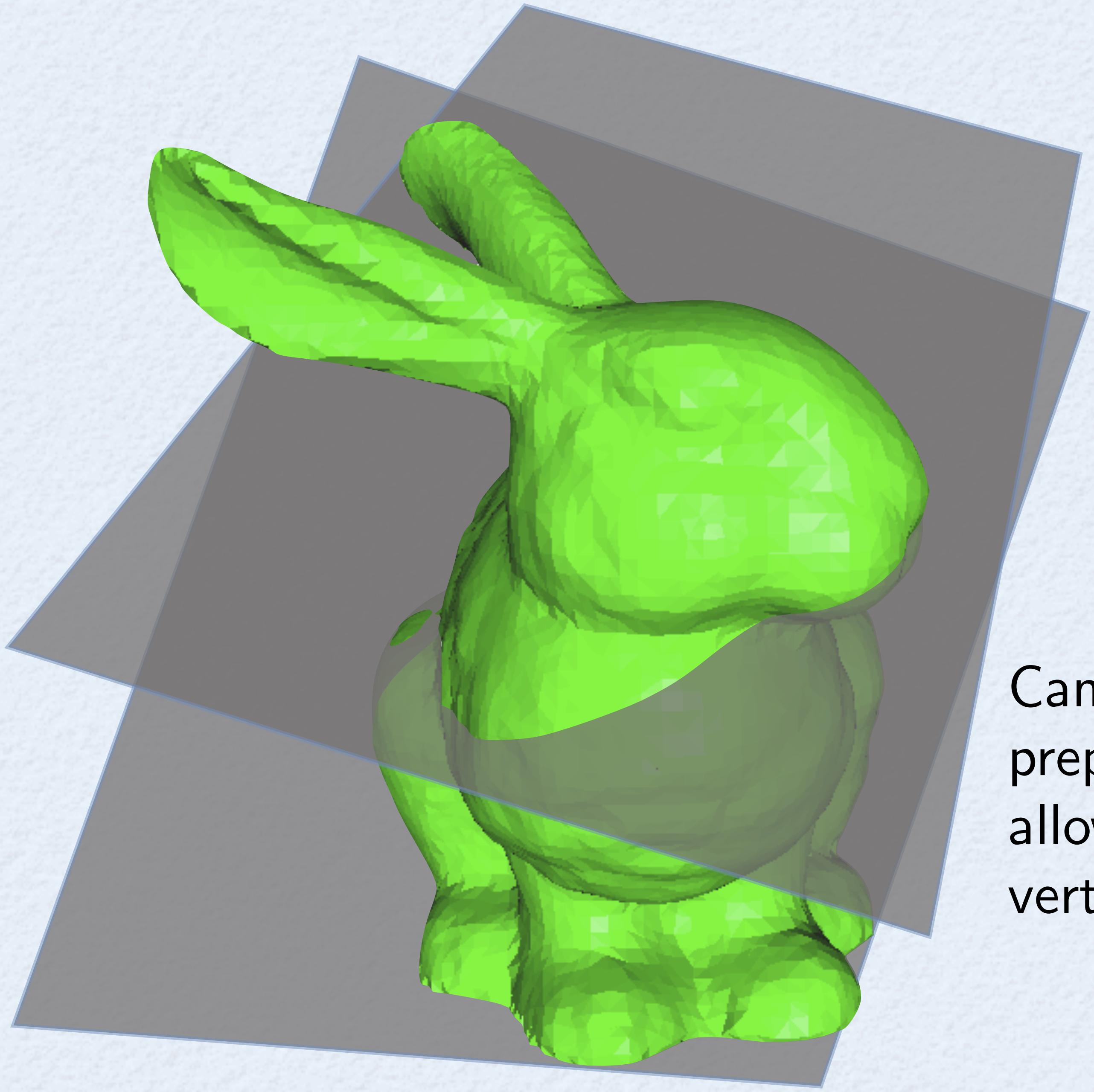


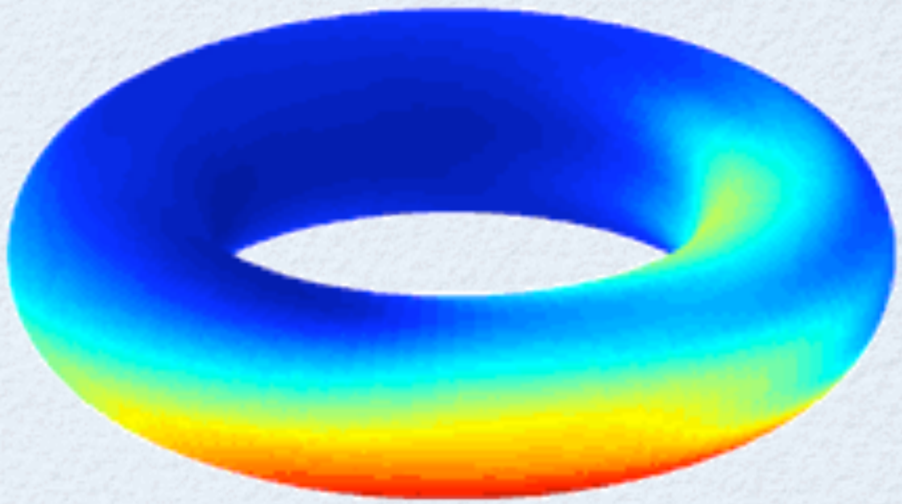
Can a topological sphere $\mathbb{M}$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?

Remarks and Open Problems

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 ≥ 1? (Orientable or not)



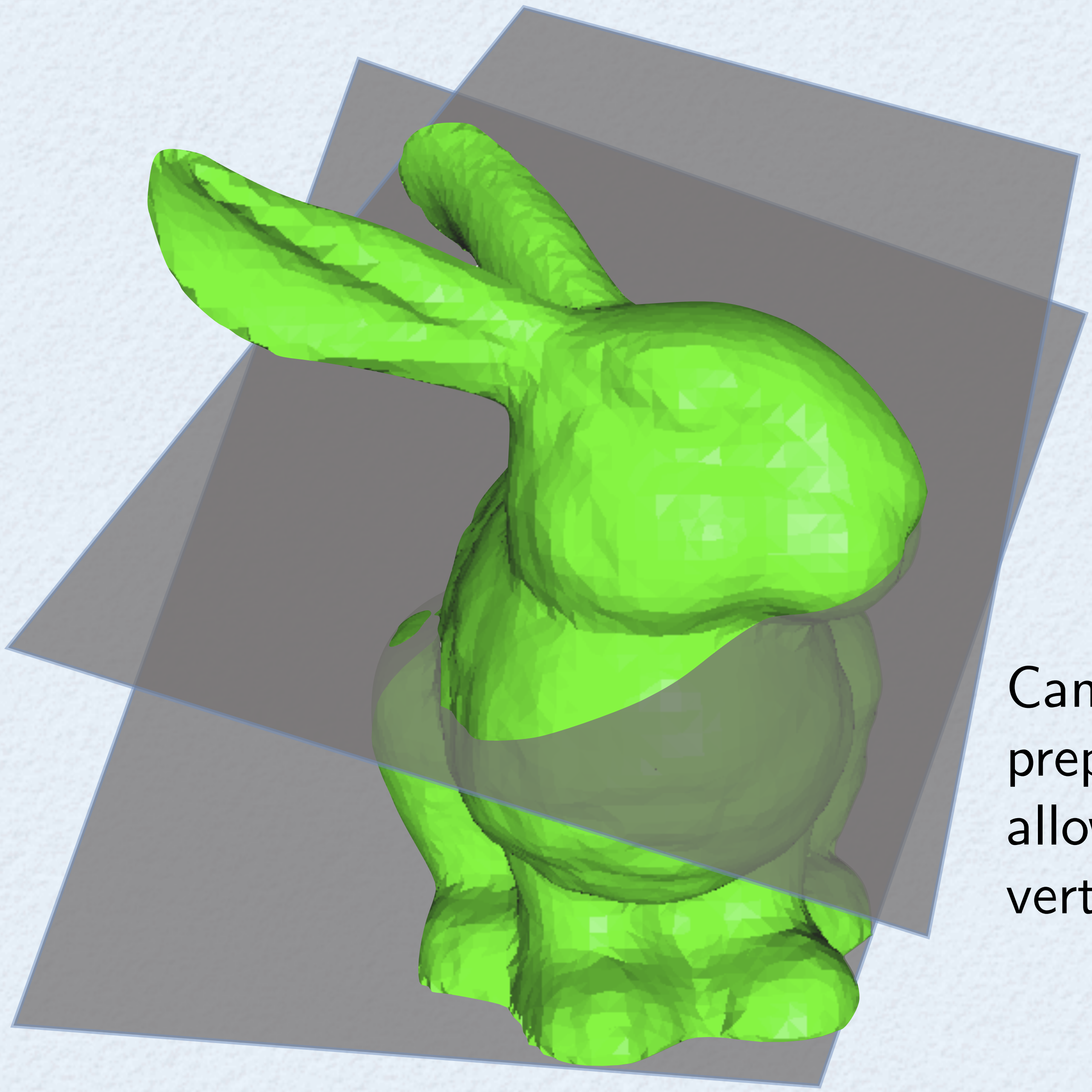


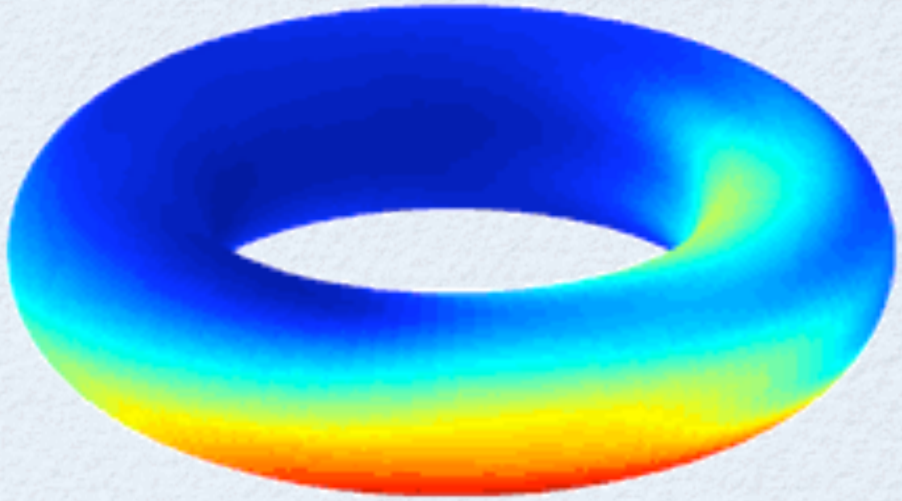
Can a topological sphere $\mathbb{M}$ embedded in $\mathbb{R}^3$ be preprocessed into a (near) linear-size structure that allow efficient answering of queries that specify the vertical direction?

Remarks and Open Problems

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 ≥ 1? (Orientable or not)





Can a topological sphere $\mathbb{M}$ 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!