Computing Visibility on Terrains in External Memory

Herman Haverkort

TU. Eindhoven
Netherlands

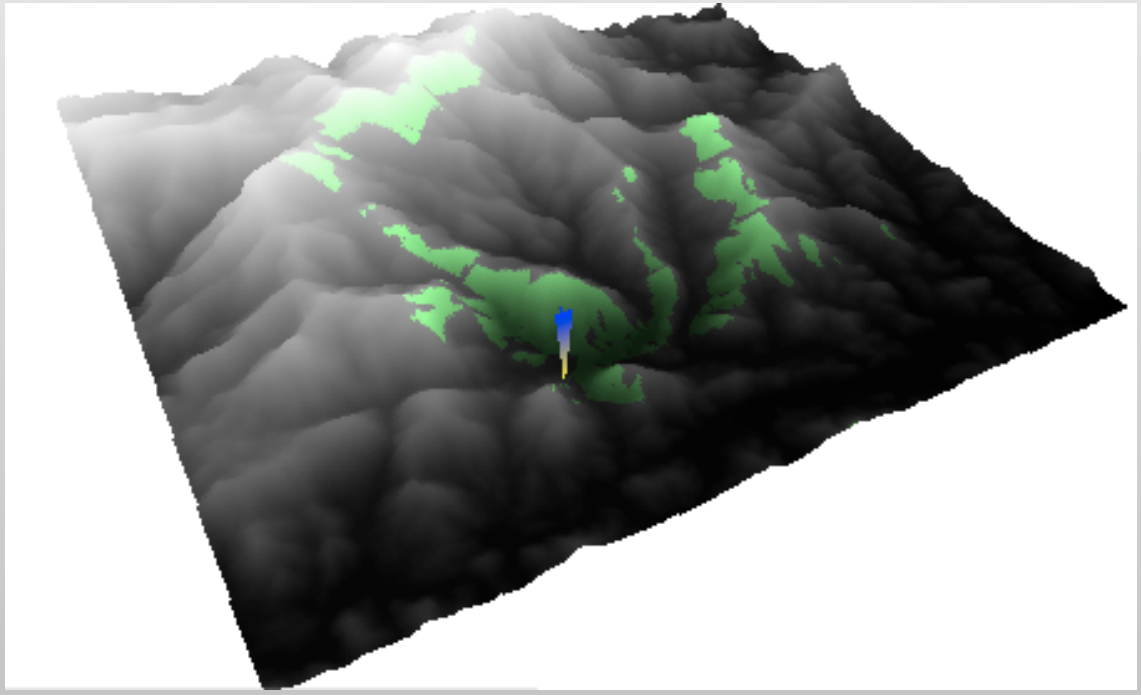
Laura Toma

Yi Zhuang

Bowdoin College
USA

Visibility

- Problem: visibility map (viewshed) of v
  - terrain T
  - arbitrary viewpoint v
  - the set of points in T visible from v



Sierra Nevada, 30m resolution

Visibility

- Problem: visibility map (viewshed) of v
  - terrain T
  - arbitrary viewpoint v
  - the set of points in T visible from v

- Applications
  - graphics
  - games
  - GIS
    - military applications, path planning, navigation
    - placement of fire towers, radar sites, cell phone towers (terrain guarding)

Massive terrains

- Why massive terrains?
  - Large amounts of data are becoming available
    - NASA SRTM project: 30m resolution over the entire globe (~10TB)
    - LIDAR data: sub-meter resolution
- Traditional algorithms don't scale
  - Buy more RAM?
    - Data grows faster than memory
  - Data on disk
  - Disks are MUCH slower than memory
- => I/O-bottleneck

I/O-efficient algorithms

- I/O-model [AV'88]
  - Data on disk, arranged in blocks
  - I/O-operation = reading/writing one block from/to disk

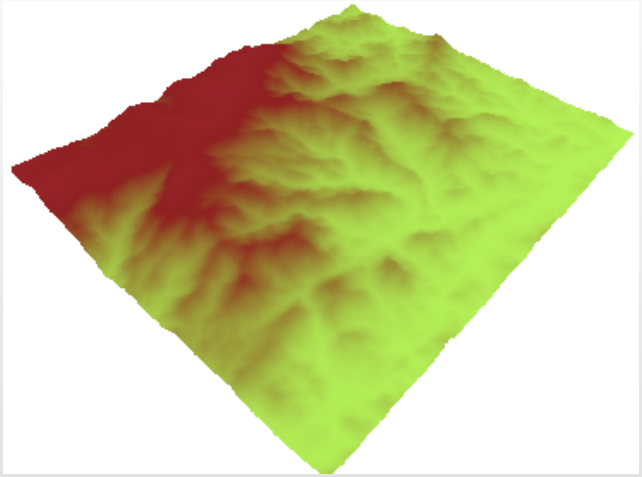
n=input size	M=memory size	B=block size

- I/O-complexity: nb. I/O-operations

- Basic I/O bounds

$$\text{scan}(n) = \Theta\left(\frac{n}{B}\right) \quad < \quad \text{sort}(n) = \Theta\left(\frac{n}{B}\log_{M/B} n/M\right) \quad \ll \quad n$$

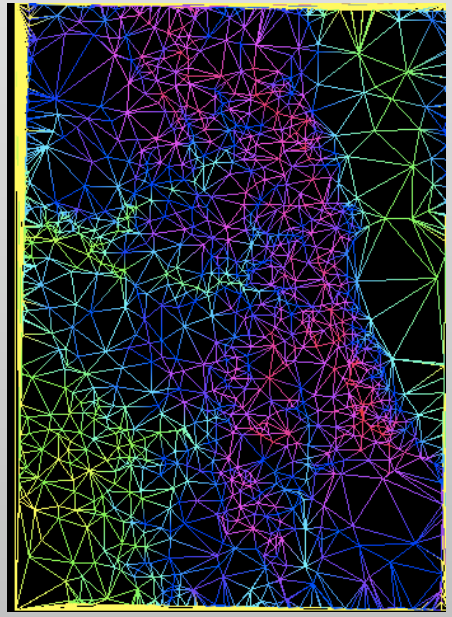
Terrain data



- Most often: grid terrain

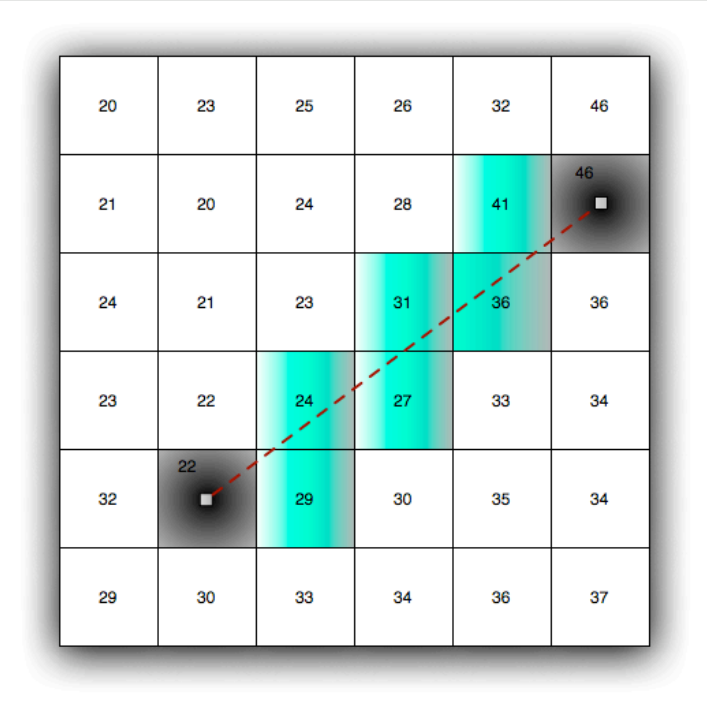
	20	23	25	26	32	46	
	21	20	24	28	41	46	
	24	21	23	31	36	36	
	23	22	24	27	33	34	
	32	22	29	30	35	34	
	29	30	33	34	36	37	

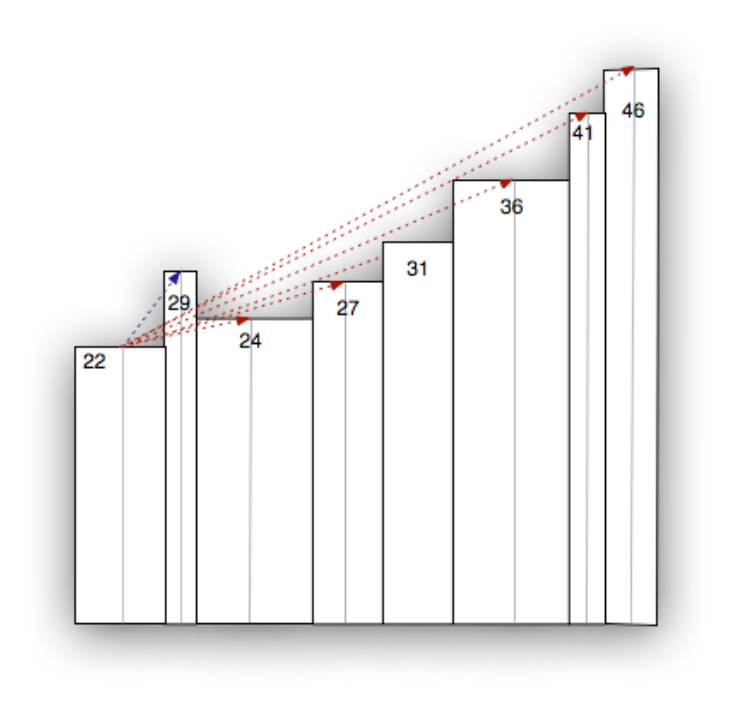
- TIN (triangulated polyhedral terrain)



Visibility on grids

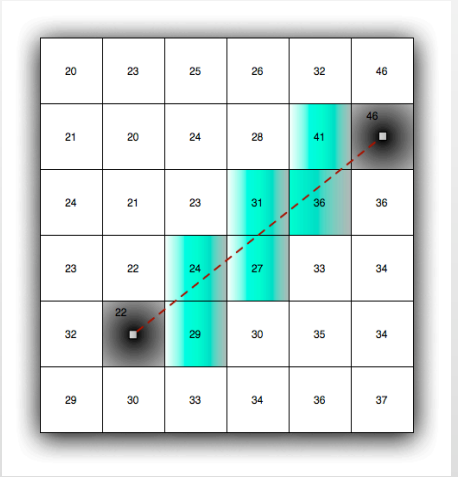
- Line-of-sight model
  - a grid cell with center q is visible from viewpoint v iff the line segment vq does not cross any cell that is above vq



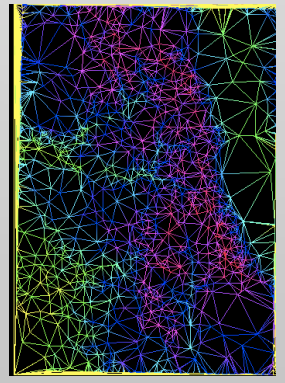


Visibility: Related work

- Grids
  - straightforward algorithm $O(n^2)$
  - $O(n \lg n)$ by van Kreveld
  - experimental
    - Fisher [F93, F94], Franklin & Ray [FR94], Franklin [F02]
    - no worst-case guarantees

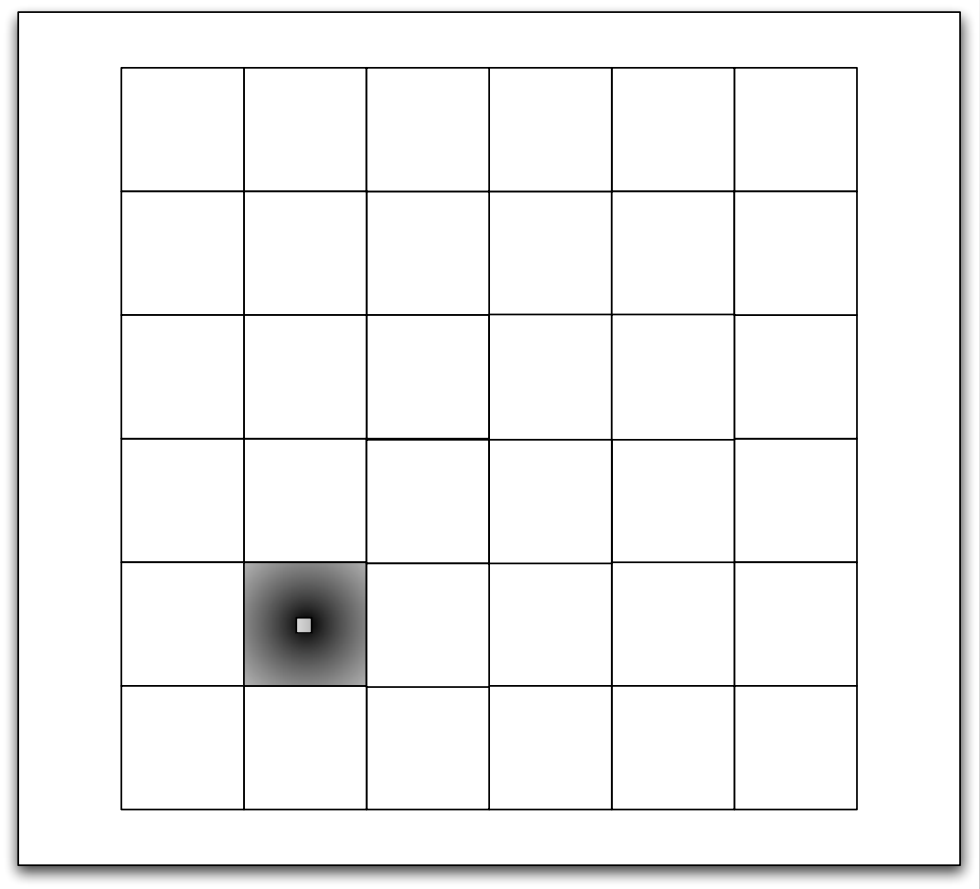


- TINs
  - surveys: de Floriani & Magillo [FM94], Cole & Sharir [CS89]
  - recently: watchtowers and terrain guarding [SoCG'05, SODA'06]

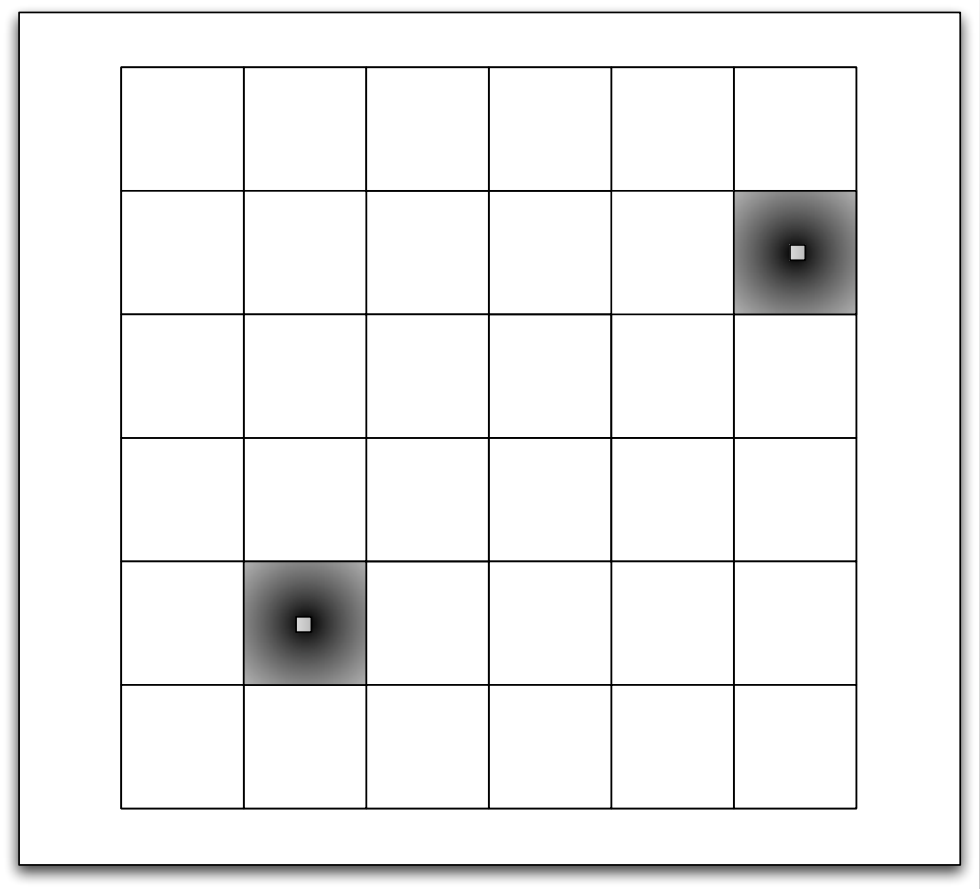


van Kreveld's algorithm

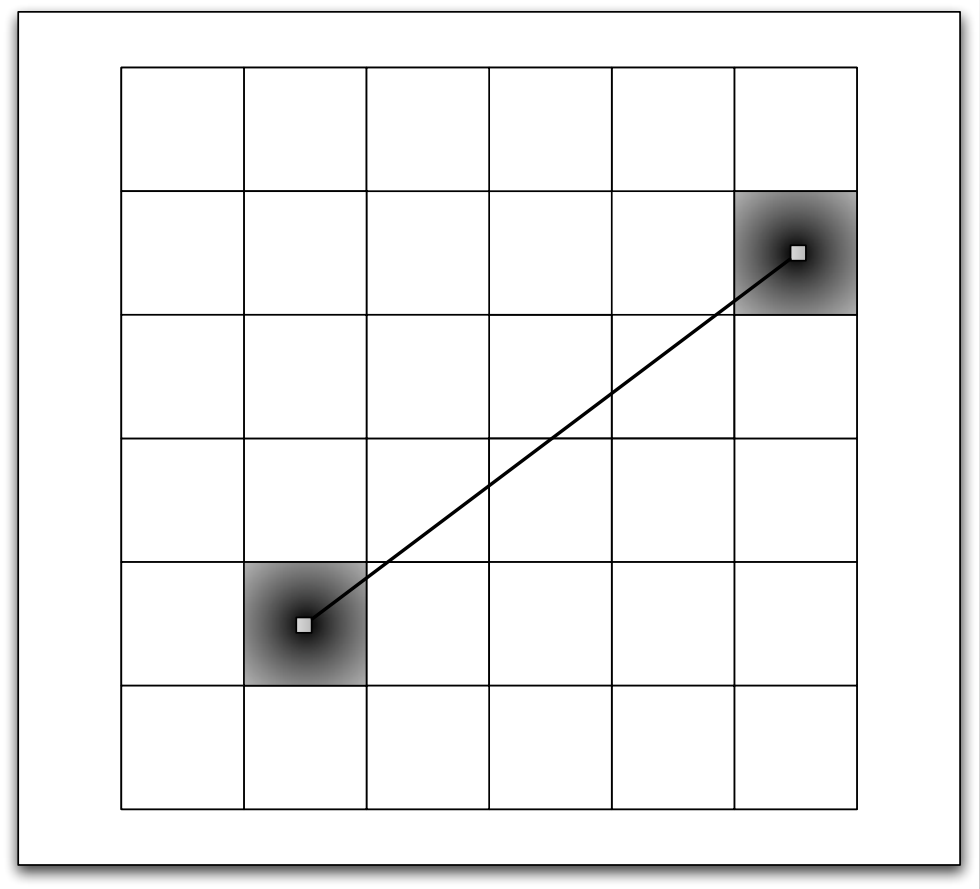
van Kreveld's algorithm



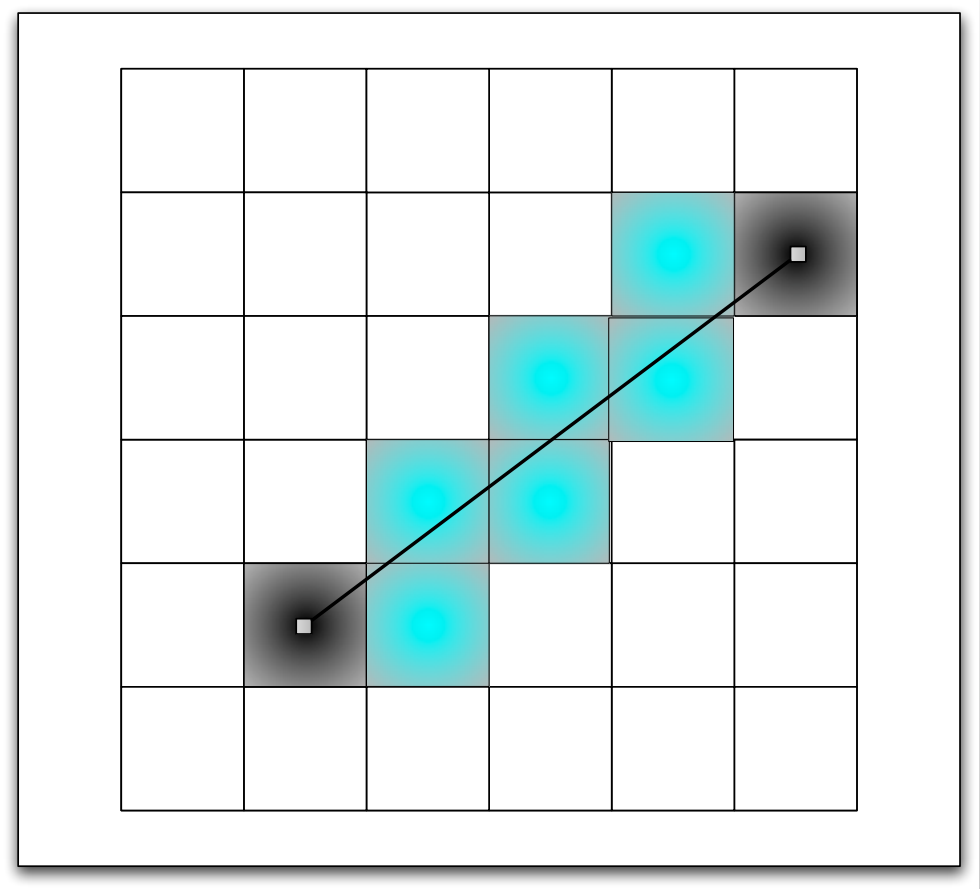
van Kreveld's algorithm



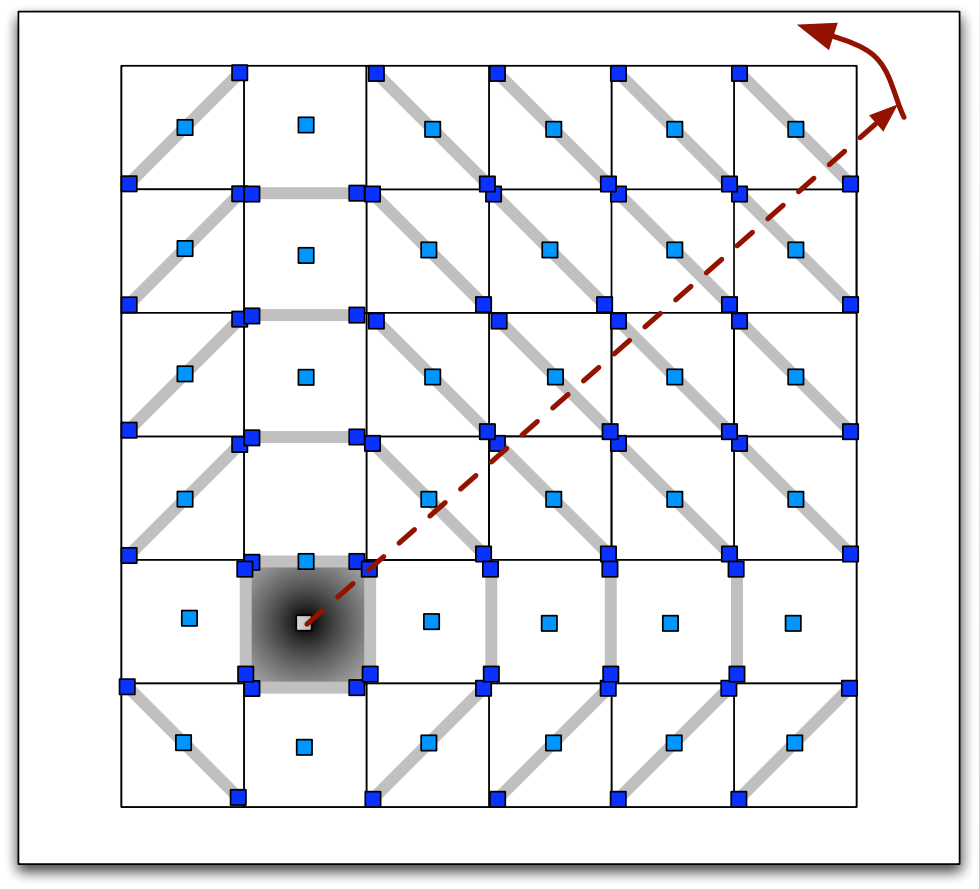
van Kreveld's algorithm



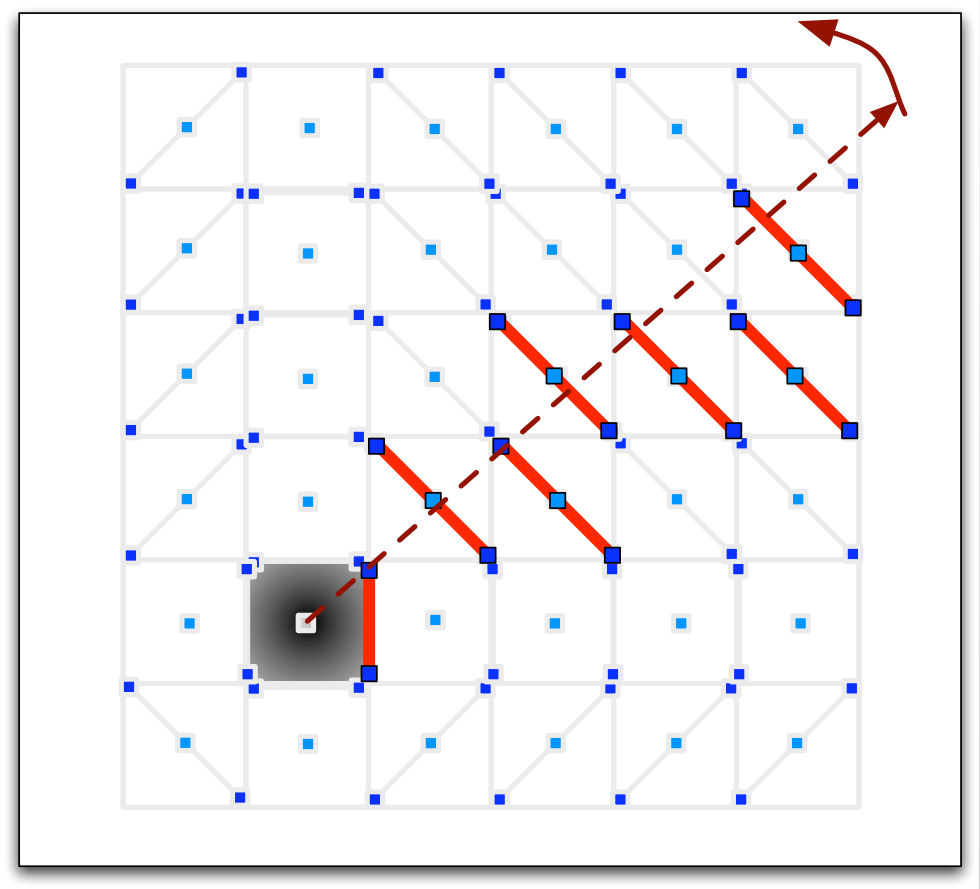
van Kreveld's algorithm



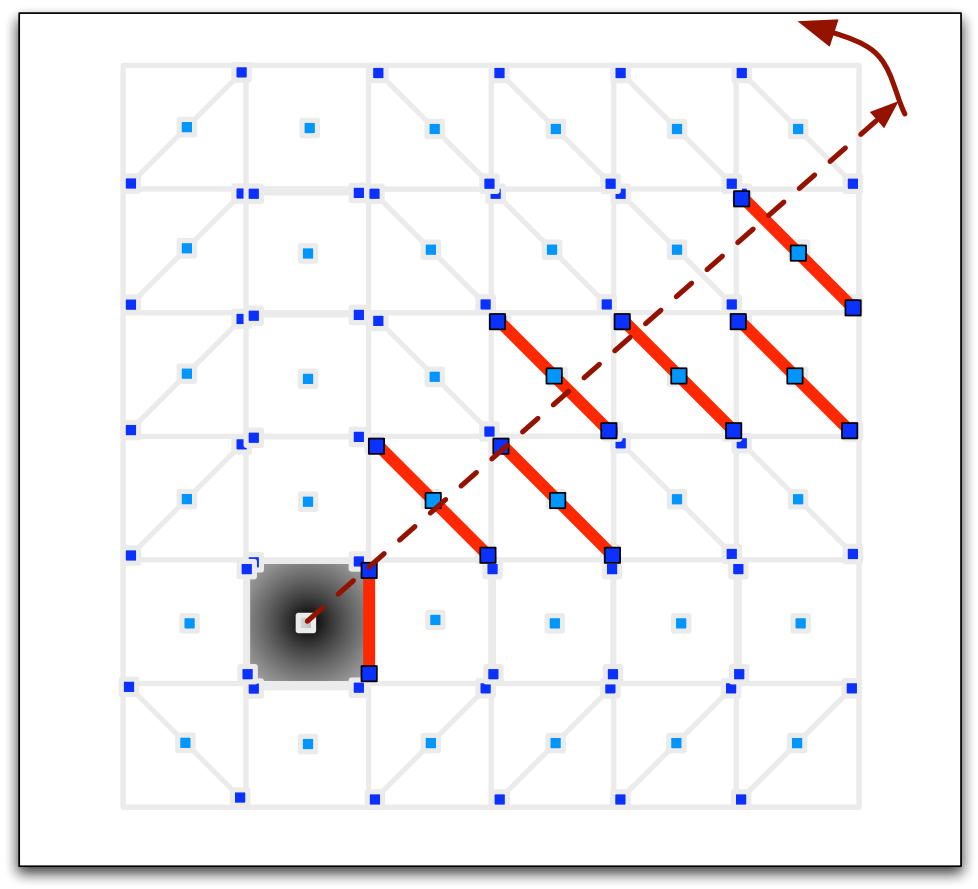
van Kreveld's algorithm

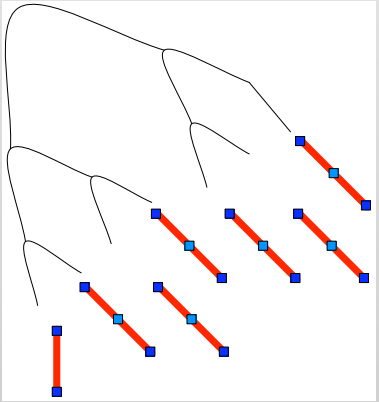


van Kreveld's algorithm

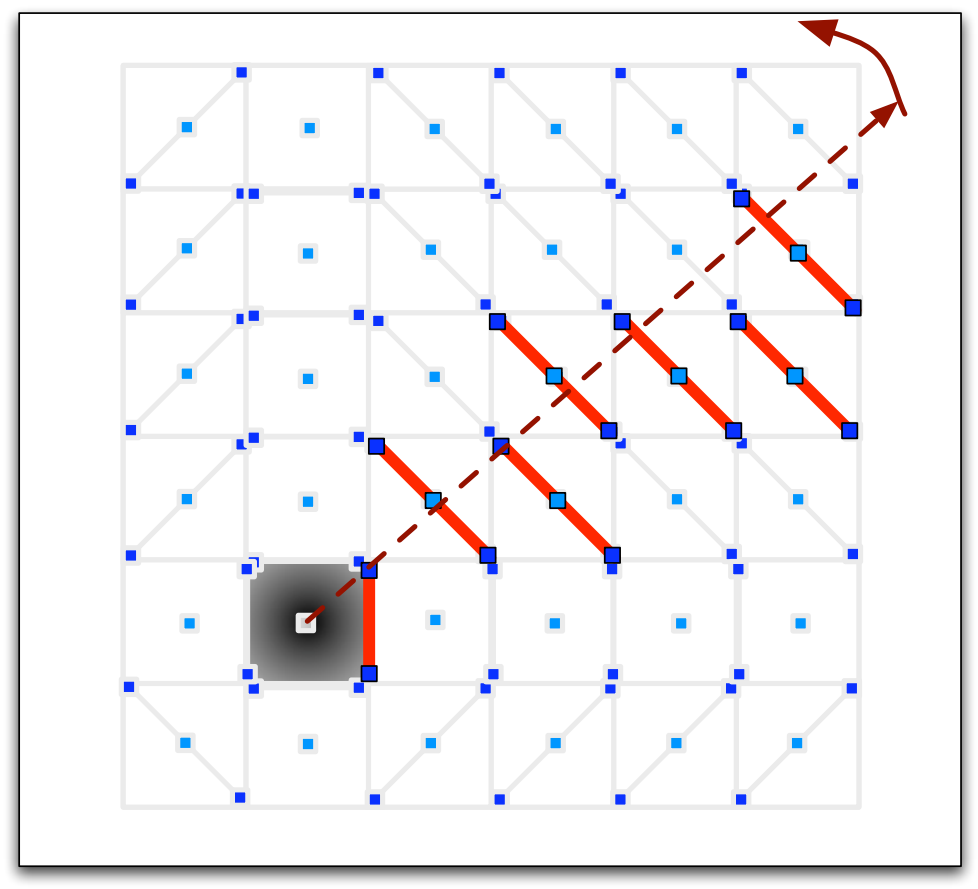


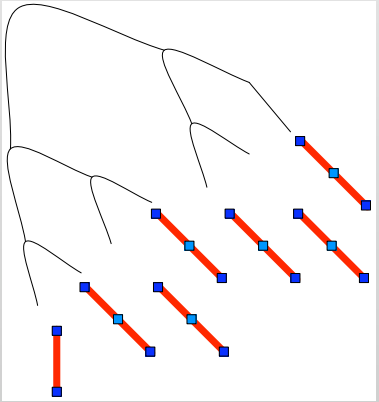
van Kreveld's algorithm





van Kreveld's algorithm





- 3n events, O(lg n) per event --> O(n lg n) CPU time

van Kreveld's algorithm

-in external memory-

van Kreveld's algorithm

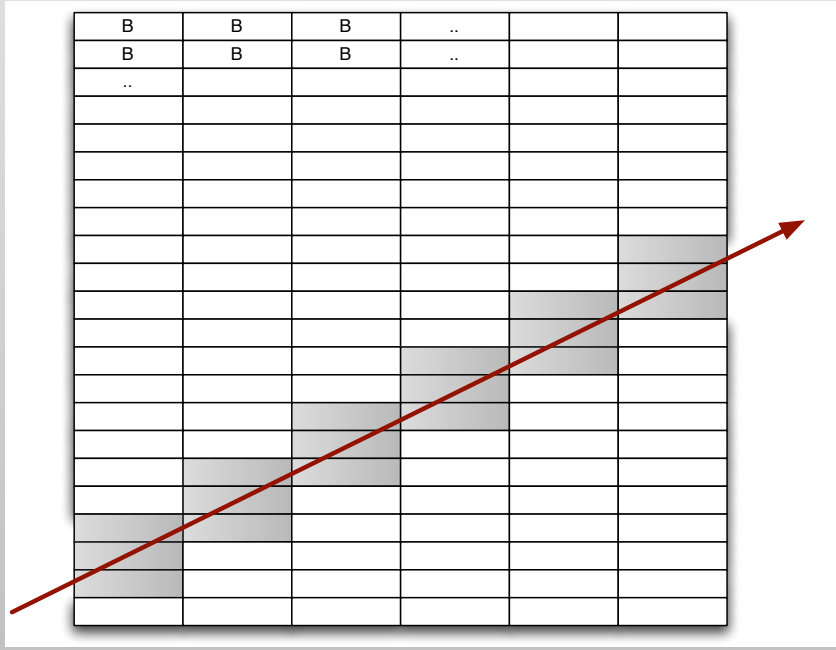
-in external memory-

- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, accessed in rotational order
  - event list
  - active structure

van Kreveld's algorithm

-in external memory-

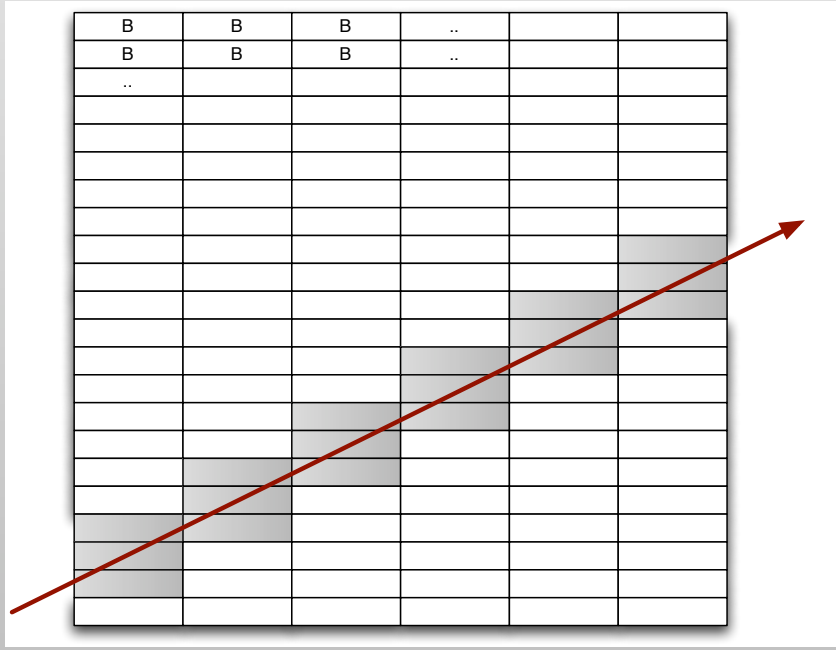
- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, accessed in rotational order
  - event list
  - active structure



van Kreveld's algorithm

-in external memory-

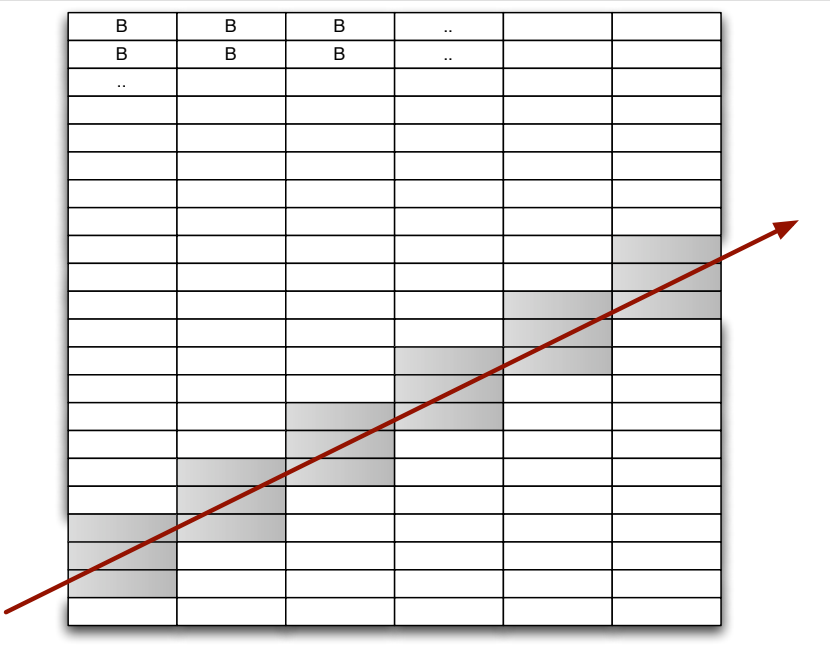
- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, accessed in rotational order
  - event list
  - active structure
- $\Omega(1)$ I/O per cell, $\Omega(n)$ I/Os total

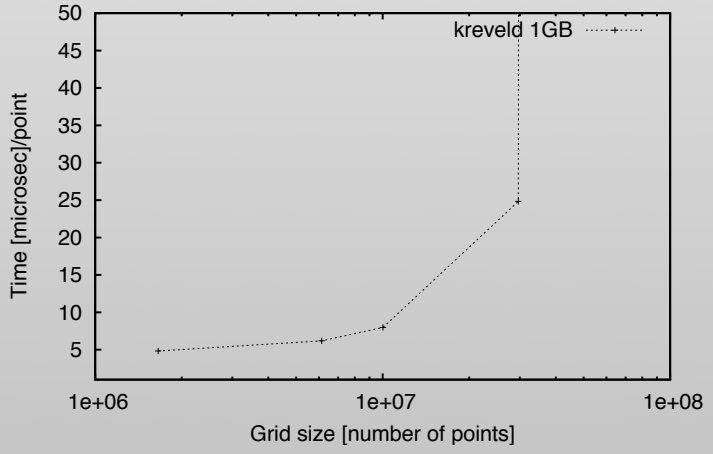


van Kreveld's algorithm

-in external memory-

- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, accessed in rotational order
  - event list
  - active structure
- $\Omega(1)$ I/O per cell, $\Omega(n)$ I/Os total

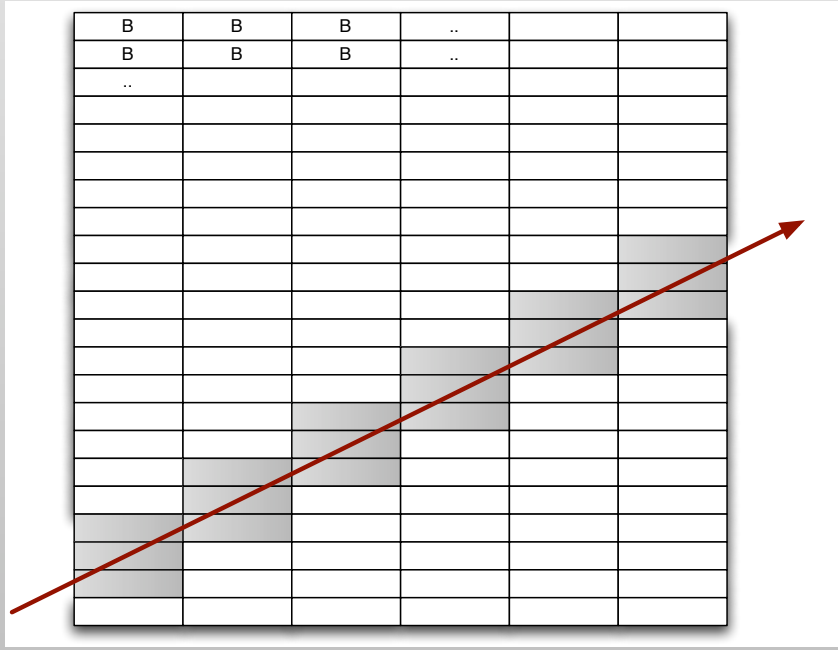


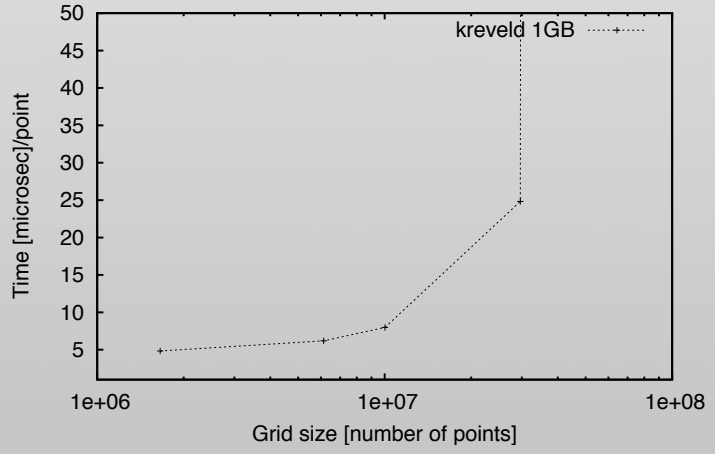


van Kreveld's algorithm

-in external memory-

- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, accessed in rotational order
  - event list
  - active structure
- $\Omega(1)$ I/O per cell, $\Omega(n)$ I/Os total





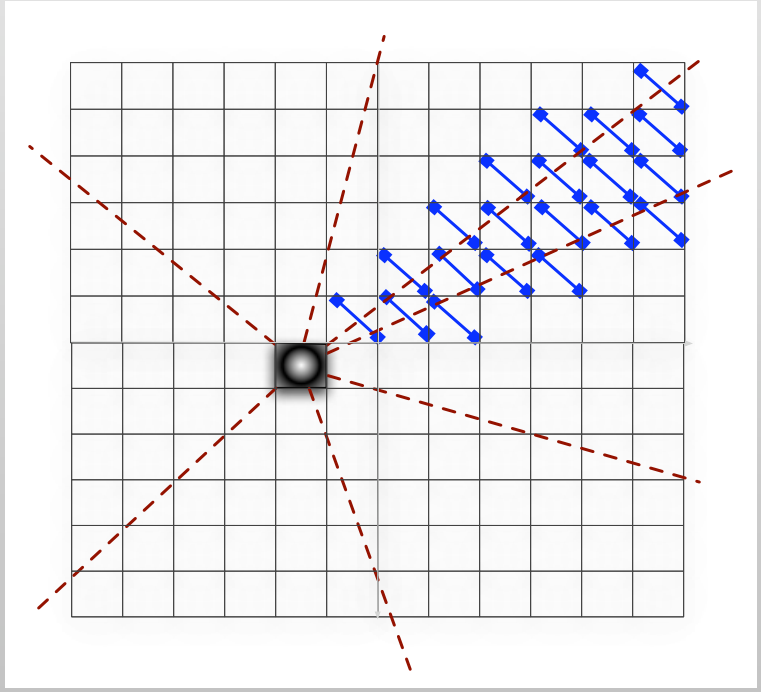
Our results

n = grid size M=memory size size B=block size

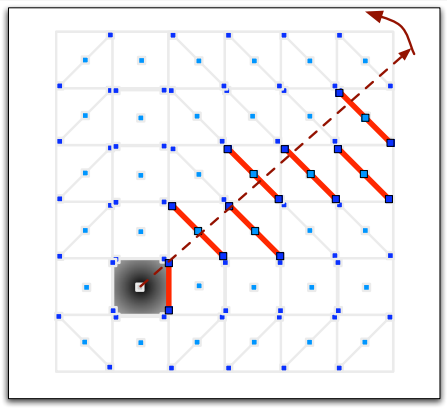
- The visibility grid of an arbitrary viewpoint on a grid of size n can be computed with O(n) space and O(sort(n)) I/Os
- Experimental evaluation
  - ioviewshed
  - standard algorithm (Kreveld)
  - visibility algorithm in GRASS GIS

Computing visibility in external memory

- Distribution sweeping [GTVV FOCS93]
  - divide input in M/B sectors each containing an equal nb. of points
  - solve each sector recursively
  - handle sector interactions



The base case

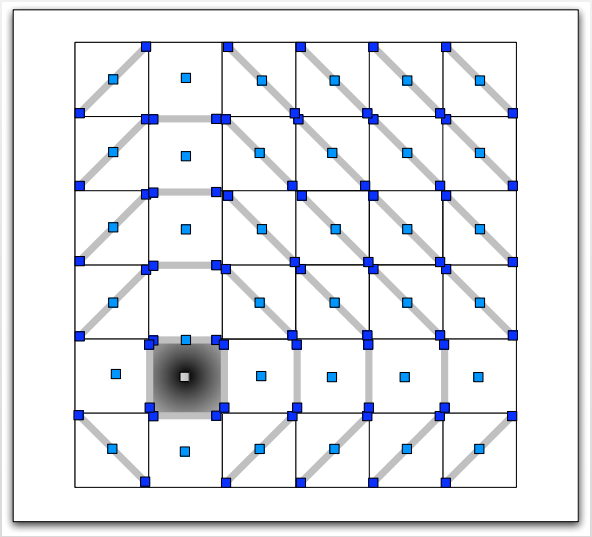


- Usually, stop recursion when n < M
- Our idea: stop when status structure fits in memory

- Run modified Kreveld
  - elevation grid: encode elevation in event
  - event list: store events in a sorted stream on disk
  - visibility grid: when determining visibility of a cell, write it to a stream. Sort the stream at the end to get visibility grid

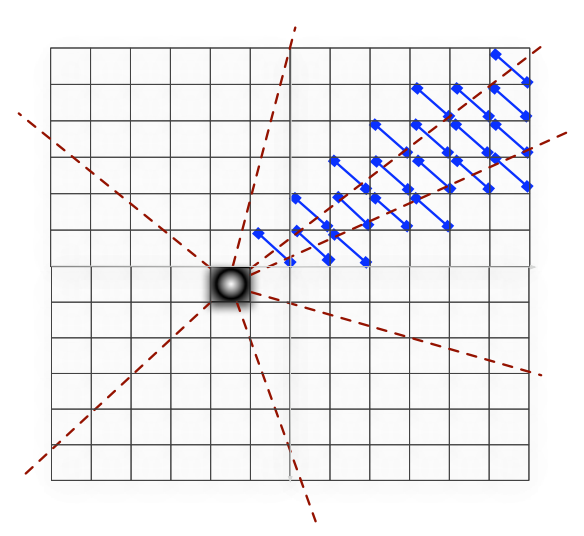
- Total: O(sort(n)) I/Os

The recursion



- cell <--> {start, end, query}
- 3n events

The recursion

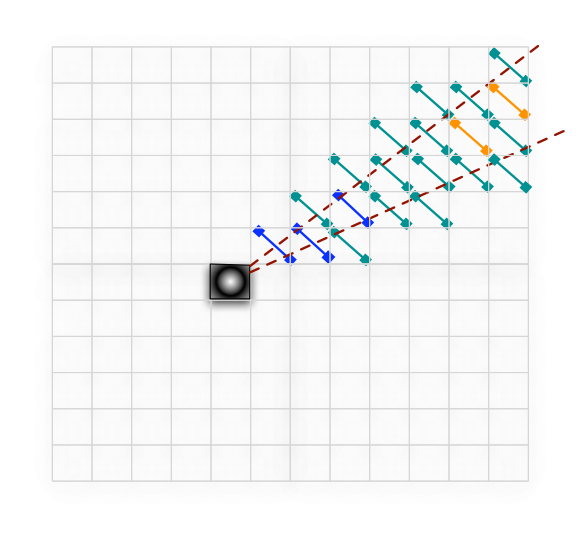


- Divide events into O(M/B) sectors of equal size
- $O(\log_{M/B} n)$ recursion levels

- If O(scan(n)) per recursion level
- --> overall $\mathrm{scan}(n) \cdot O(\log_{M/B} n) = O(\mathrm{sort}(n))$

The recursion:

Distributing events to sectors

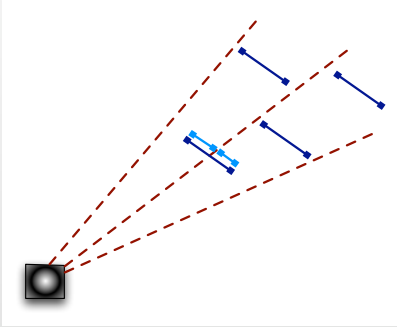


- query points
- narrow cells: crossing at most one sector boundary
- wide cells: crossing at least two sector boundaries

The recursion:

Distributing events to sectors

- narrow cells
  - cut and insert in both sectors

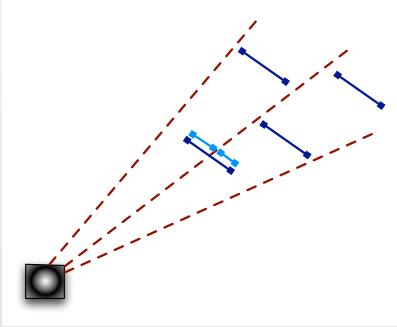


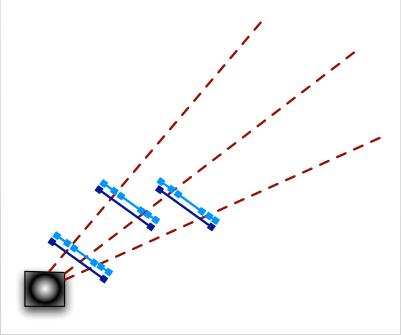
The recursion:

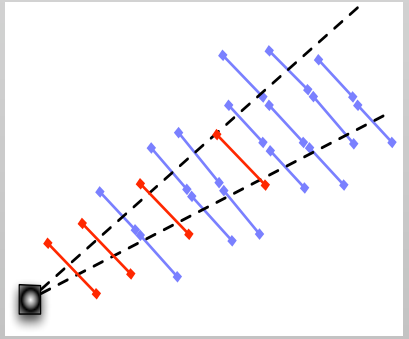
Distributing events to sectors

- narrow cells
  - cut and insert in both sectors

- wide cells
  - the visibility of a cell is determined by
    - all narrow cells in its sector that are closer to the viewpoint
    - the highest of all wide cells that span the sector and are closer to the viewpoint
  - concentric sweep
    - process wide cells spanning the sector interleaved with query points and narrow cells in the sector

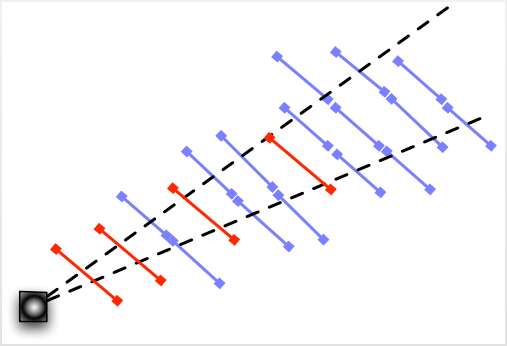






The recursion

- Input: event list in concentric order E_c and in radial order E_r



- Radial sweep: scan E_r
  - find sector boundaries

- Concentric sweep: scan E_c
  - for each sector
    - keep a block of events in memory
    - maintain the current highest wide cell spanning the sector, $High_S$
  - if next event in E_c is
    - narrow cell: if it is not occluded by $High_S$, insert in the buffer of sector. Otherwise skip it.
    - wide cell: for each sector spanned, update $High_S$
    - query point: if it is not occluded by $High_S$, insert it in the buffer of sector. Otherwise, mark it as invisible and output it.
- Recurse on each sector

O(scan(n)) per recursion level -> O(sort(n)) total

Experimental results

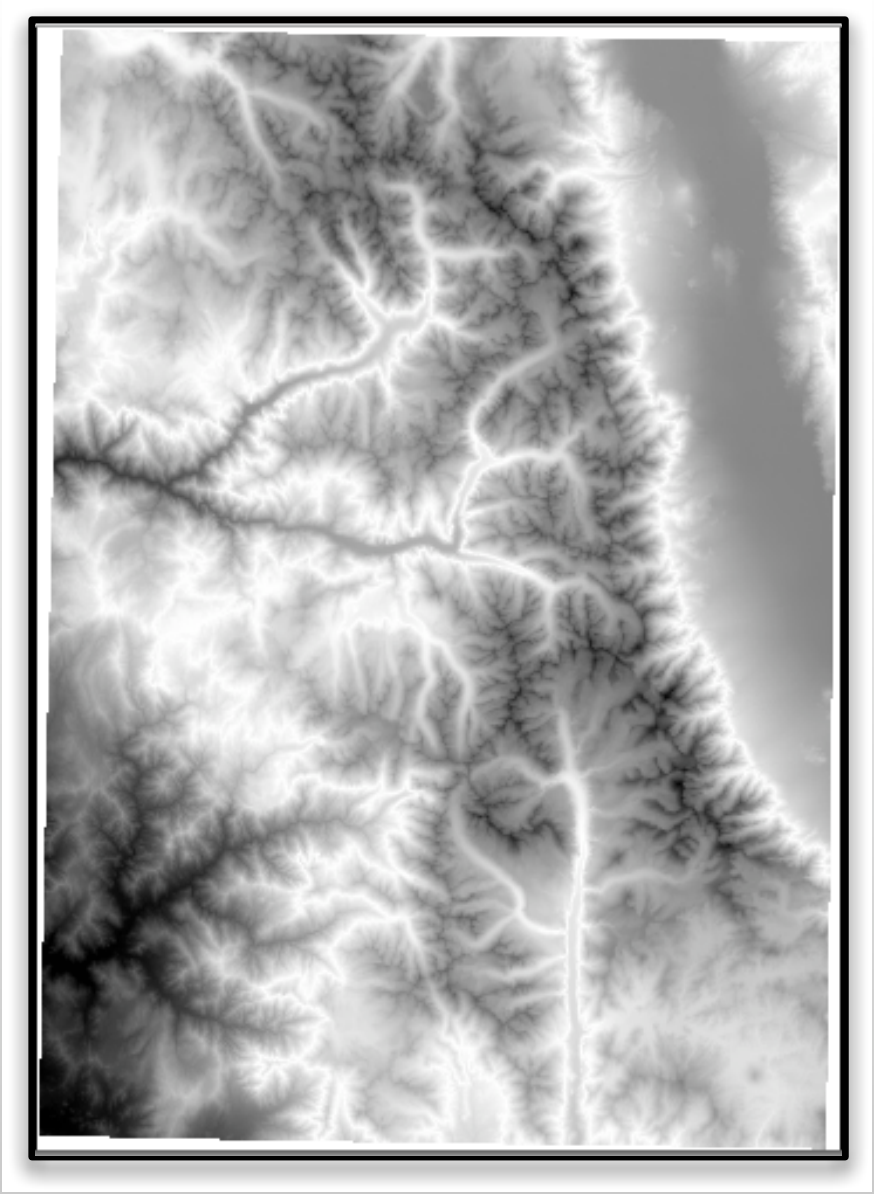
Experimental results

- kreveld
  - C
  - uses virtual memory system
- ioviewshed
  - C++
  - uses an I/O core derived from TPIE library
- GRASS visibility module
  - $O(n^2)$ straightforward algorithm
  - uses GRASS library for virtual memory management
  - program will always run (no malloc() fails) but … slow

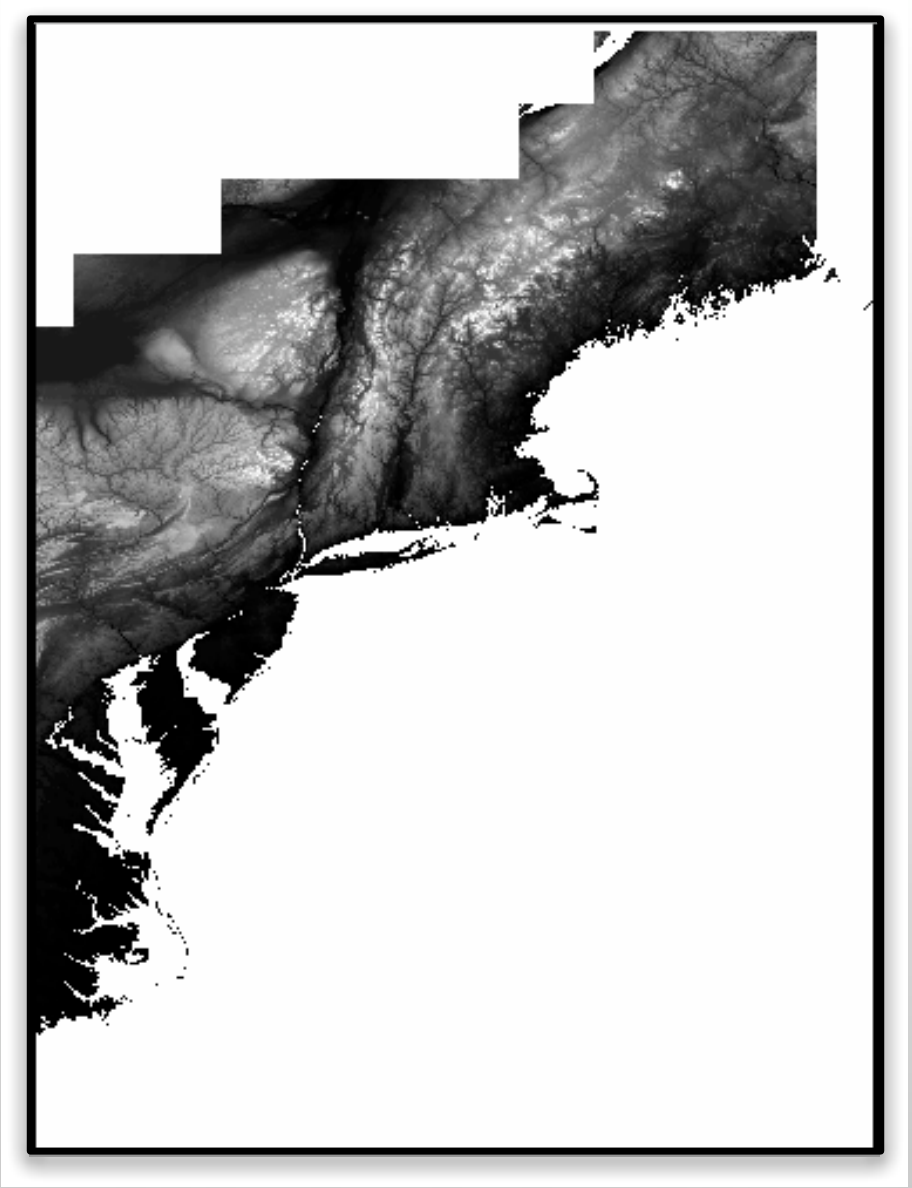
Experimental results

- Experimental Platform
  - Apple Power Mac G5
  - Dual 2.5 GHz processors
  - 512 KB L2 cache
  - 1 GB RAM

Dataset	Grid Size (million elements)	MB (Grid Only)	Valid
Kaweah	1.6	7	56%
Sierra Nevada	9.5	40	96%
Cumberlands	67	267	27%
Lower New England	77.8	311	36%
East Coast USA	246	983	36%
Midwest USA	280	1122	86%
Washington	1066	4264	95%



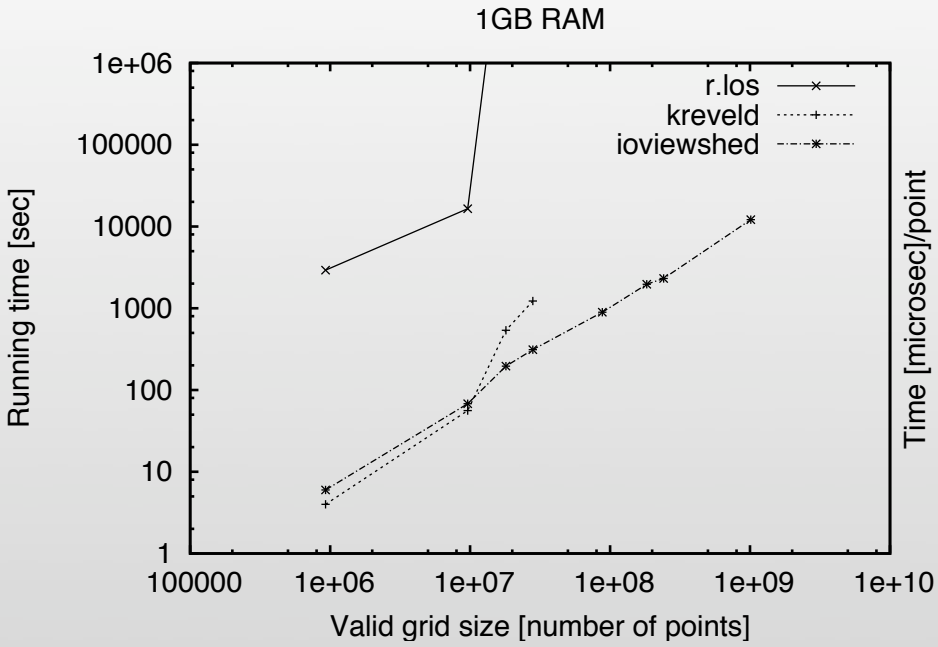
- Sierra Nevada, 30m resolution, 40MB



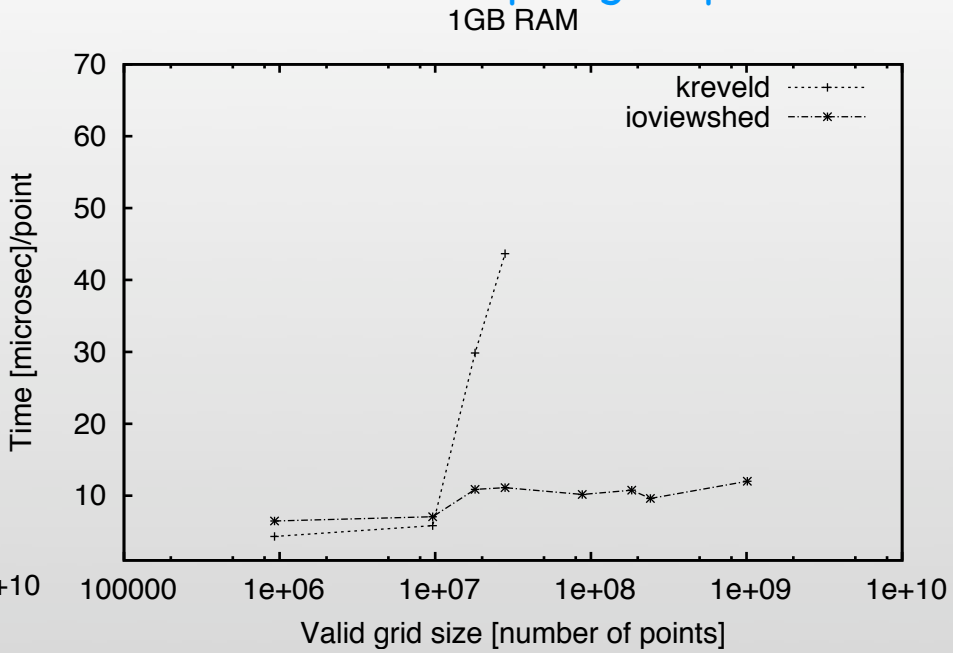
- East-Coast USA, 30m resolution, 983 MB

1GB RAM

total time (seconds)

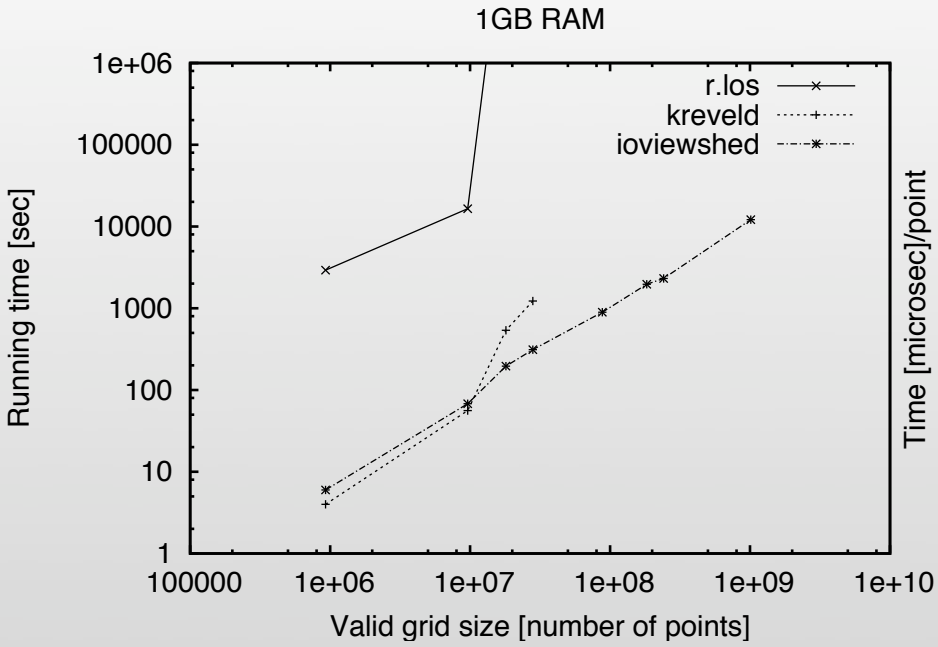


microseconds per grid point

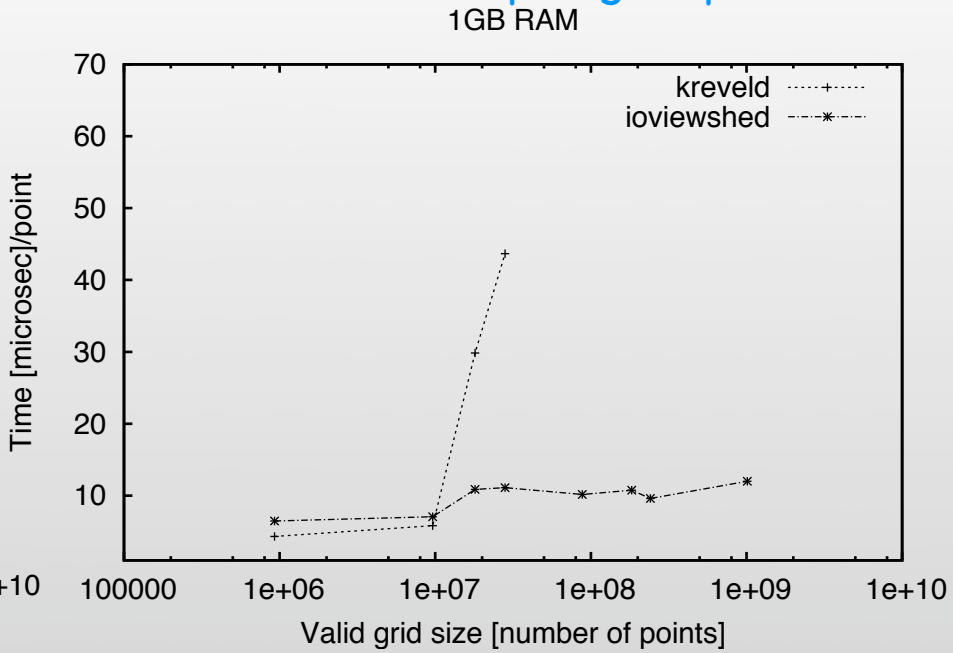


1GB RAM

total time (seconds)



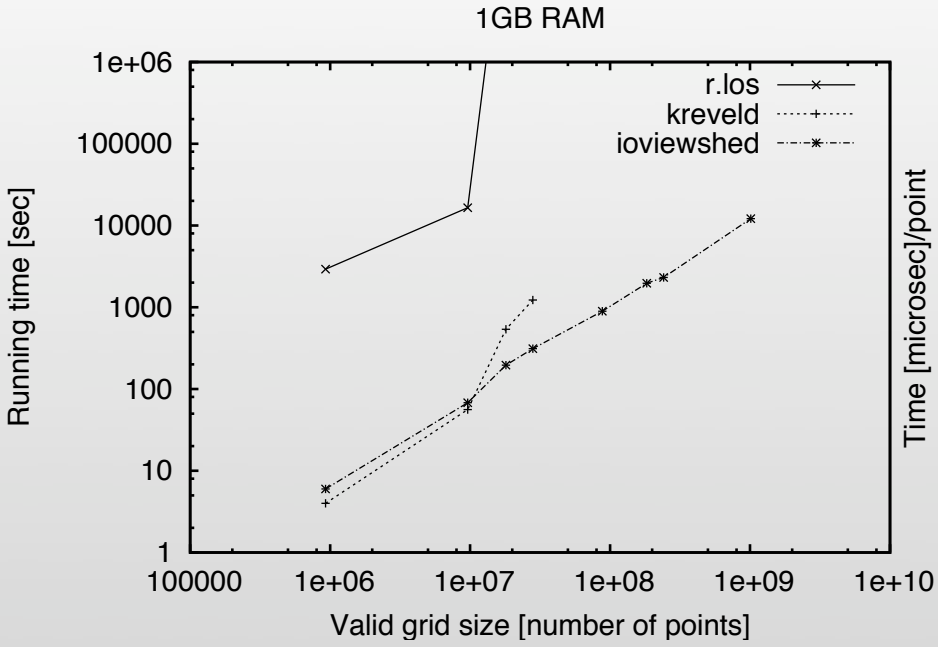
microseconds per grid point



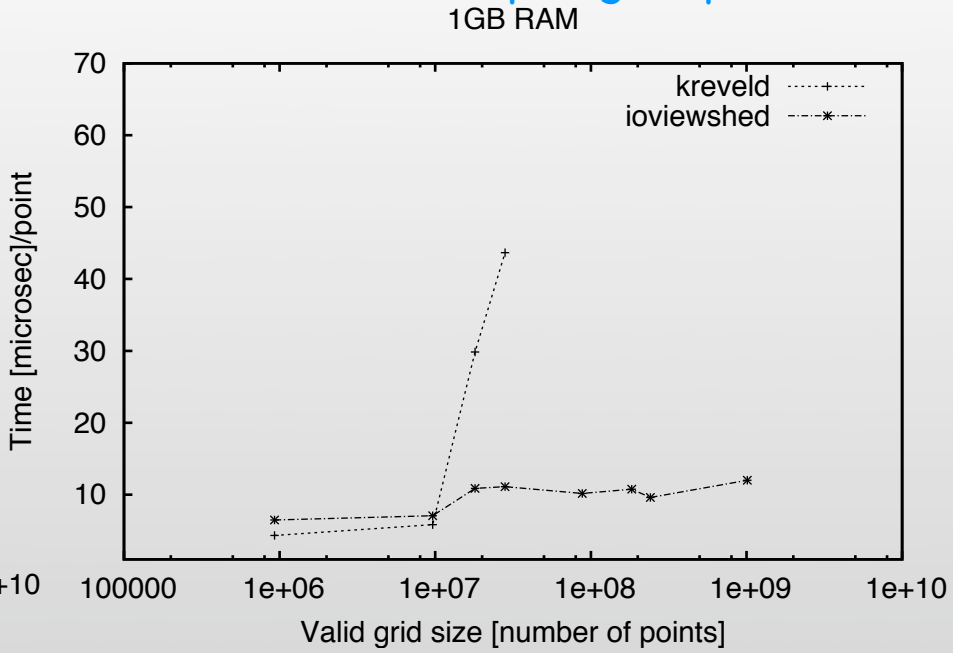
- GRASS
  - program always runs (no malloc() failures) but is very slow
- kreveld
  - starts thrashing on Cumberlands (75% CPU)
  - malloc() fails on East-Coast USA
- ioviewshed
  - finishes Washington in 3.3 hours

1GB RAM

total time (seconds)



microseconds per grid point

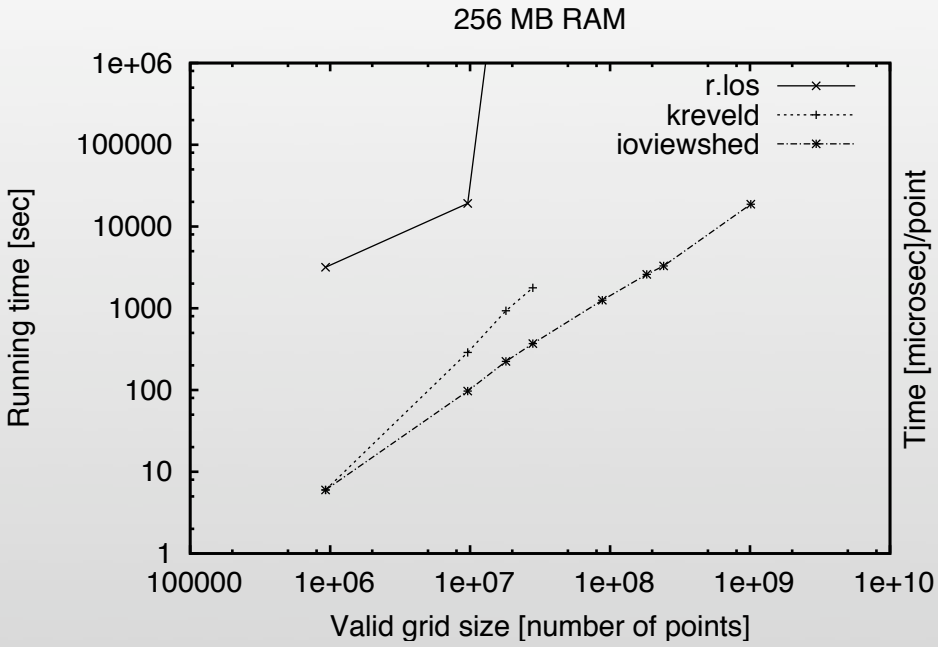


Data set	r.los	kreveld	ioviewshed
Kaweah	2 928	4 (100 %)	6 (88 %)
Sierra Nevada	16 493	56 (100 %)	68 (72 %)
Cumberlands	>1 200 000	538 (78 %)	196 (70 %)
LowerNE		1 226 (72 %)	312 (62 %)
East-Coast USA		malloc fails	894 (65 %)
Horn of Africa			1 969 (61 %)
Midwest USA			2 319 (63 %)
Washington			12 148 (65 %)

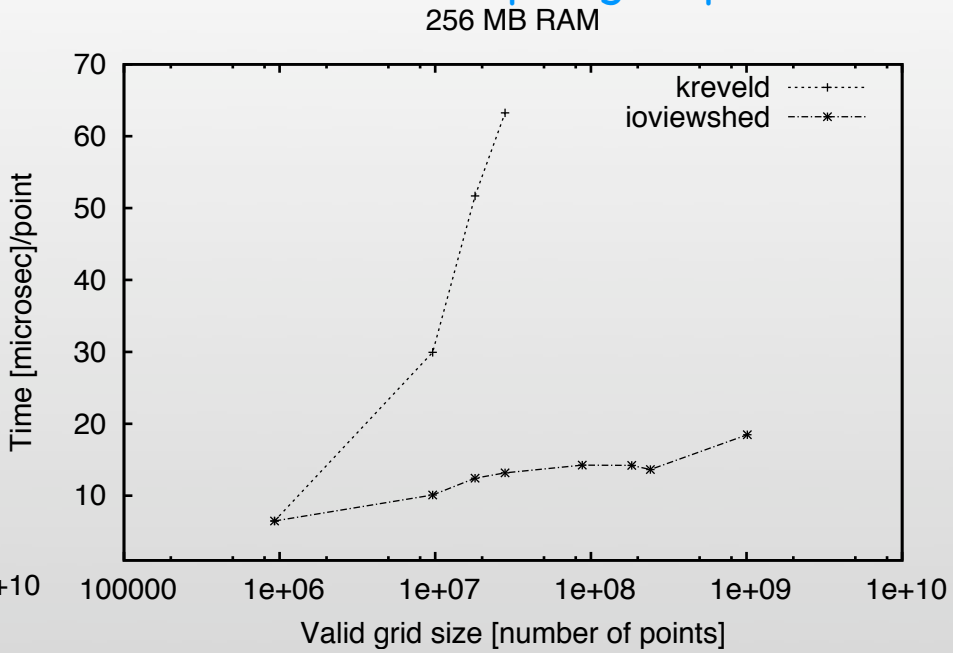
Table II. Running times (seconds) and CPU-utilization (in parentheses) at 1 GB RAM.

256MB RAM

total time (seconds)



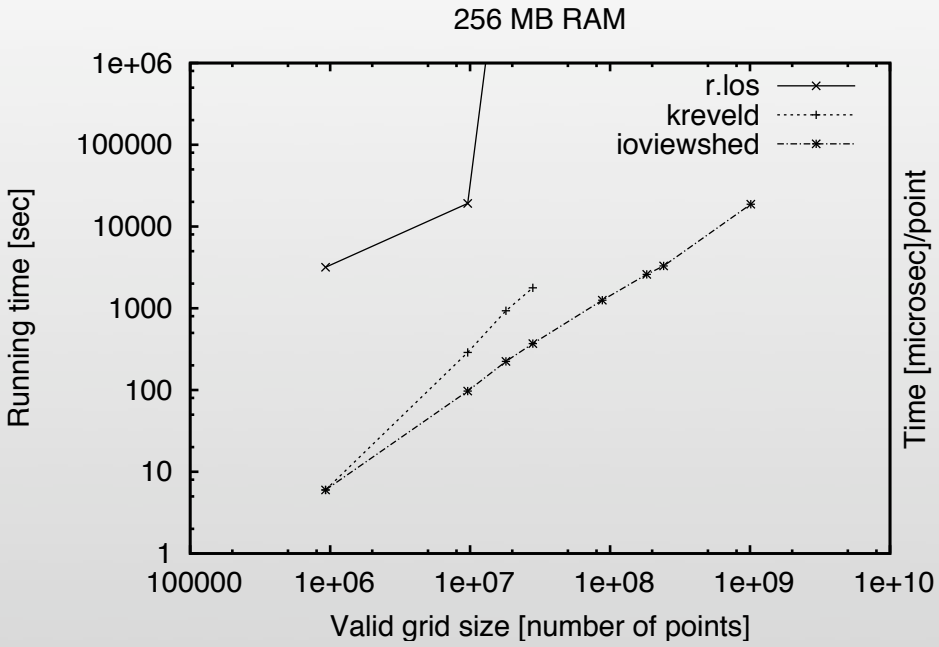
microseconds per grid point



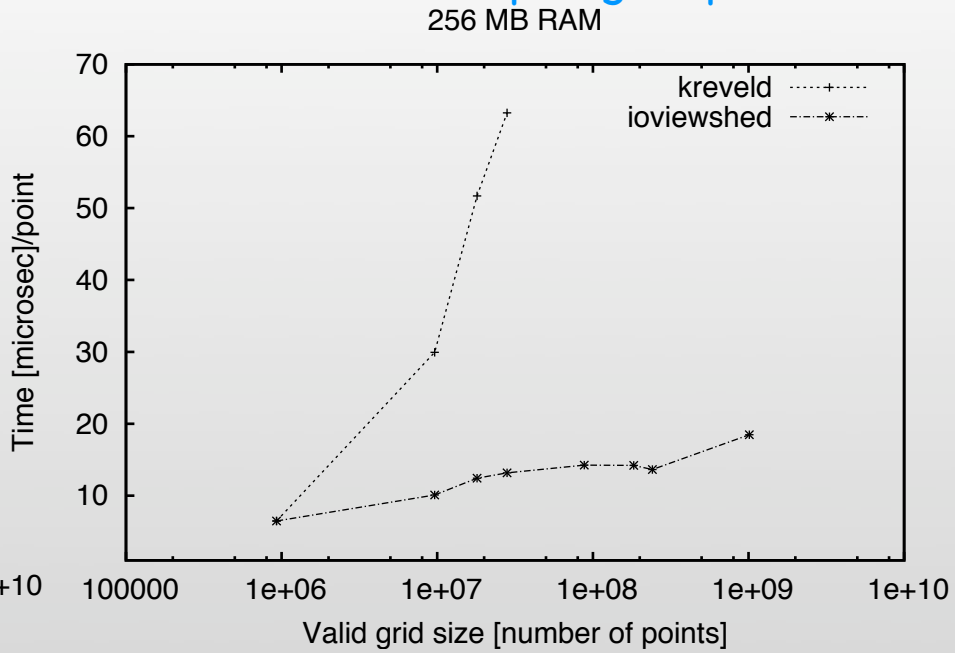
- kreveld starts thrashing earlier (Sierra, 33% CPU)
- ioviewshed scales up

256MB RAM

total time (seconds)



microseconds per grid point



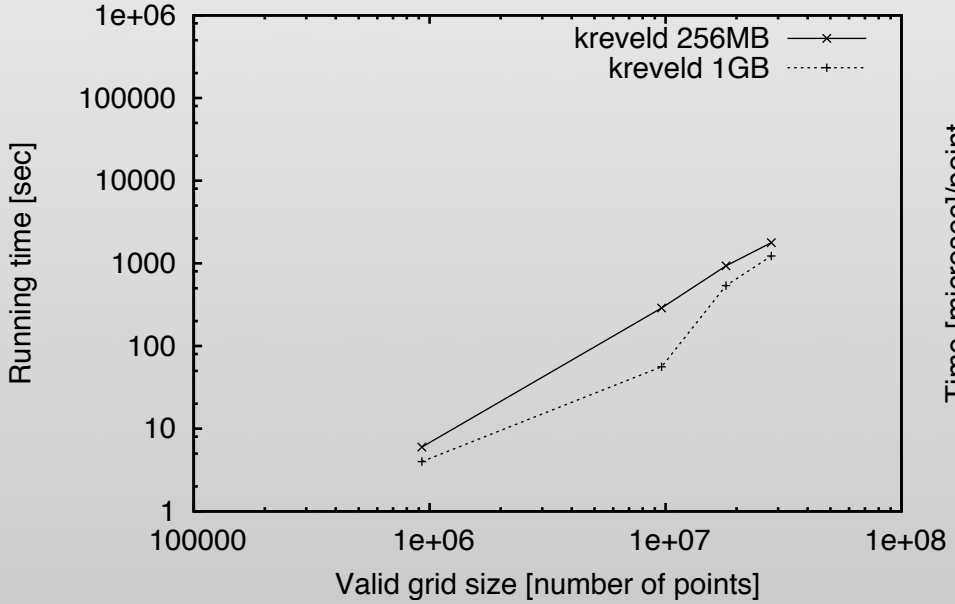
Data set	r.los	kreveld	ioviewshed
Kaweah	3 177	6 (94 %)	6 (86 %)
Sierra Nevada	19 140	288 (33 %)	97 (53 %)
Cumberlands	>1 200 000	932 (52 %)	224 (63 %)
LowerNE		1776 (56 %)	370 (54 %)
East-Coast USA		malloc fails	1254 (49 %)
Horn of Africa			2 601 (52 %)
Midwest USA			3 290 (52 %)
Washington			18 717 (50 %)

Table IV. Running times (seconds) and CPU-utilization (in parentheses) at 256 MB RAM.

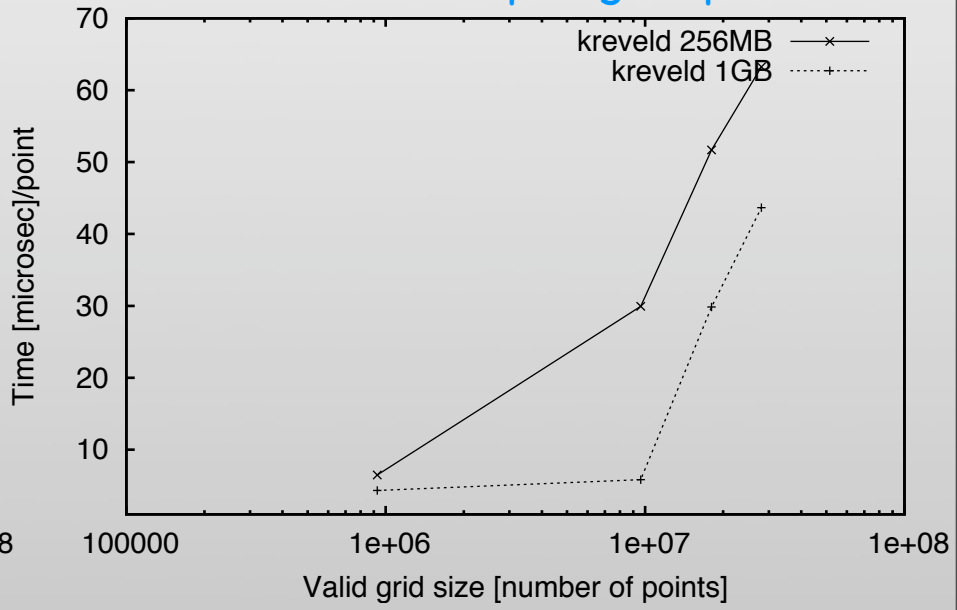
1GB vs. 256MB RAM
kreveld

Kaweah	1.6	7	56%
Sierra Nevada	9.5	40	96%
Cumberlands	67	267	27%
Lower New England	77.8	311	36%
East Coast USA	246	983	36%
Midwest USA	280	1122	86%
Washington	1066	4264	95%

total time (seconds)



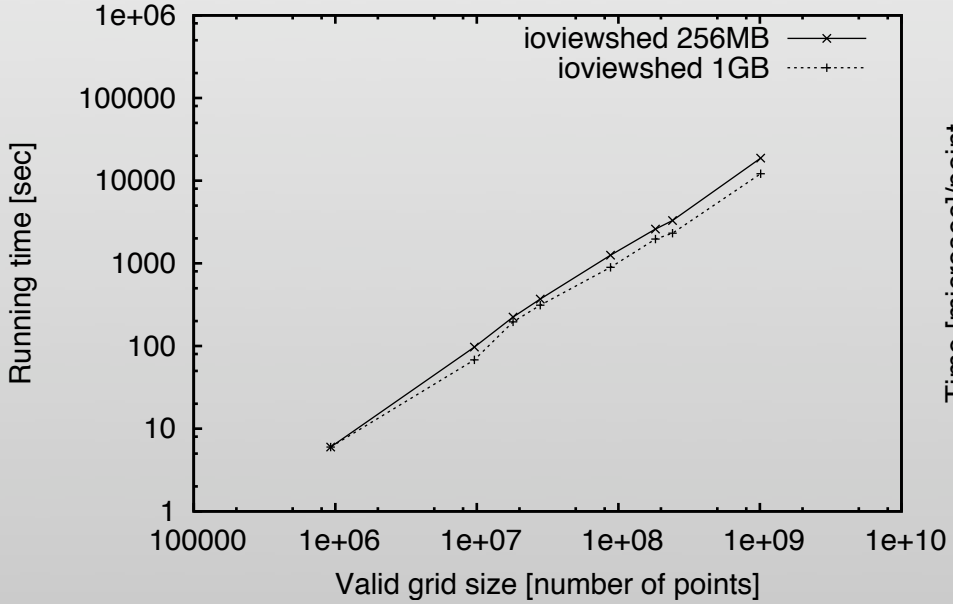
microseconds per grid point



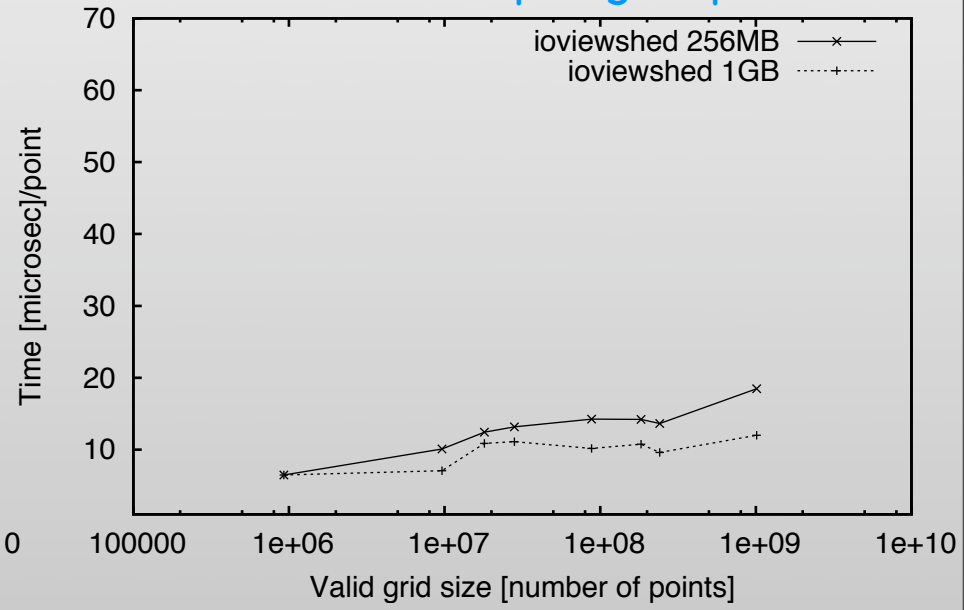
1GB vs. 256MB RAM ioviewshed

Kaweah	1.6	7	56%
Sierra Nevada	9.5	40	96%
Cumberlands	67	267	27%
Lower New England	77.8	311	36%
East Coast USA	246	983	36%
Midwest USA	280	1122	86%
Washington	1066	4264	95%

total time (seconds)



microseconds per grid point



- increase due to sorting
  - may be optimized using customized sorting (STXXL)
- in practice status structure fits in memory, never enters recursion

Conclusion

- I/O-efficient visibility computation
  - Theoretically worst-case optimal algorithm
  - Scalable
    - Can process grids that are out of scope with traditional algorithm
  - Empirical finding:
    - diagonal of dataset fits in memory
    - extended base case, no recursion necessary
- O(sort(n)) I/Os in cache-oblivious model

Thank you.