Computing Visibility on Terrains in External Memory

Herman Haverkort   Laura Toma   Yi Zhuang

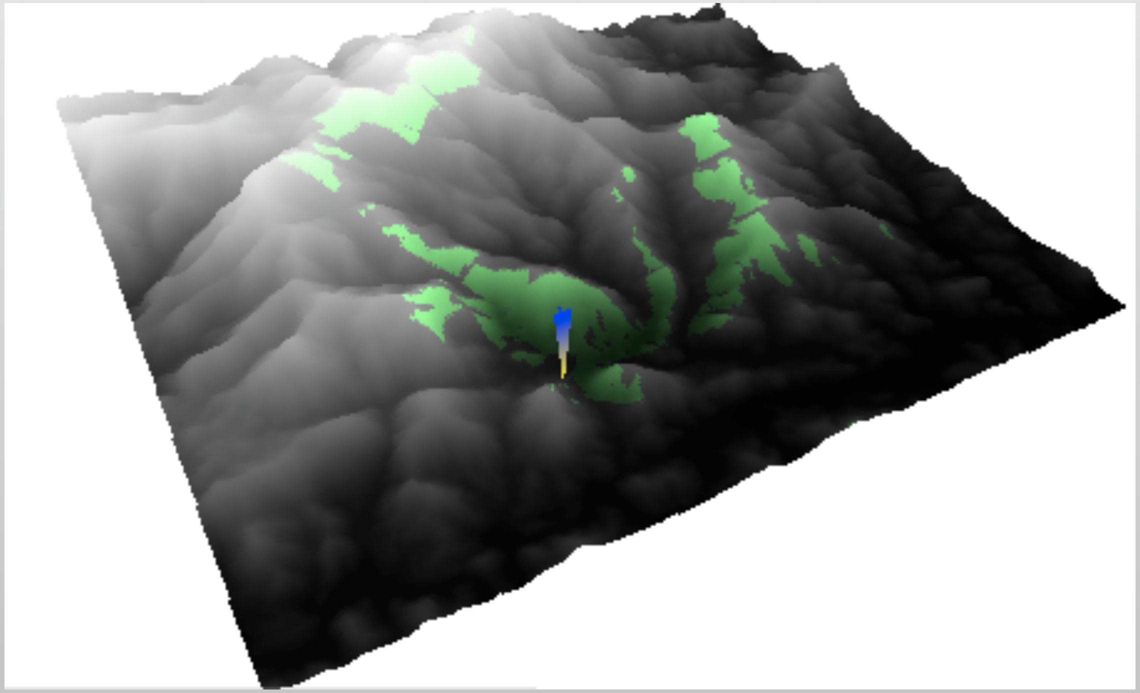
TU. Eindhoven
Netherlands

Bowdoin College
USA

ALENEX 2007
New Orleans, 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

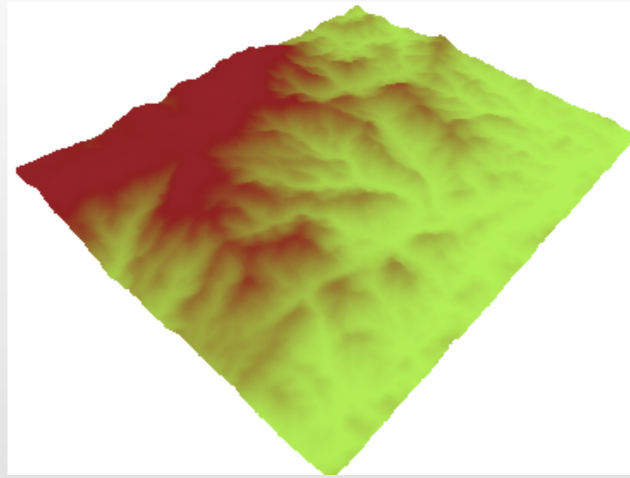
n=grid size	M=memory size	B=block size

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

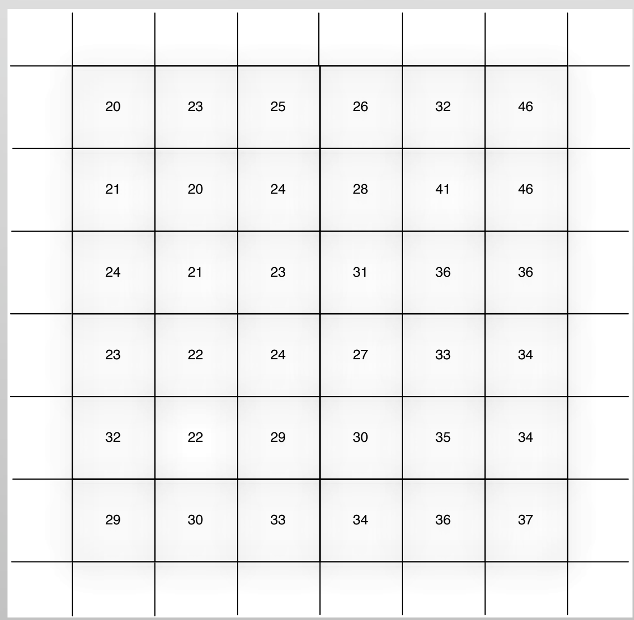
- Basic I/O bounds

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

Terrain data

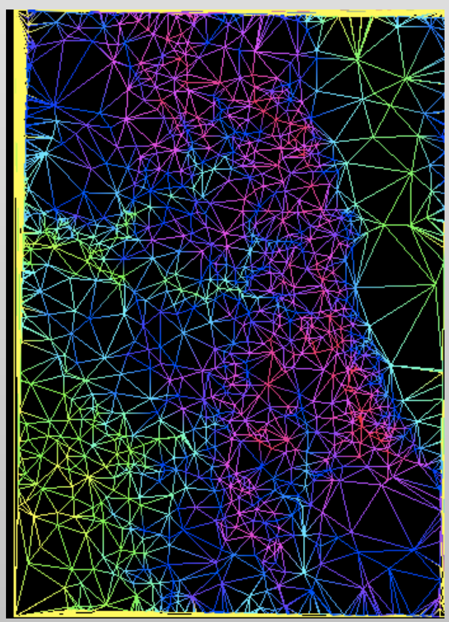


- Most often: grid terrain



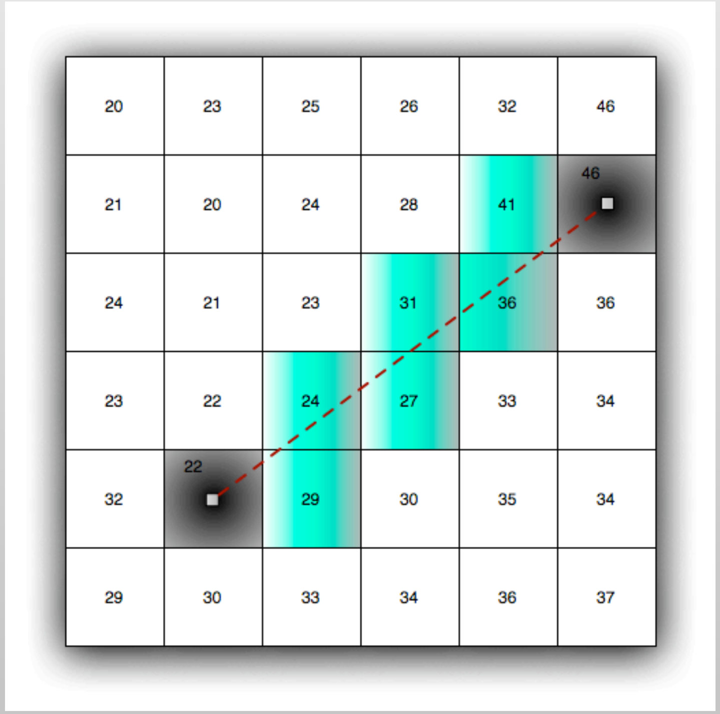
20	23	25	26	32	46
21	20	24	28	41	46
24	21	23	31	36	36
23	22	24	27	33	34
32	22	29	30	35	34
29	30	33	34	36	37

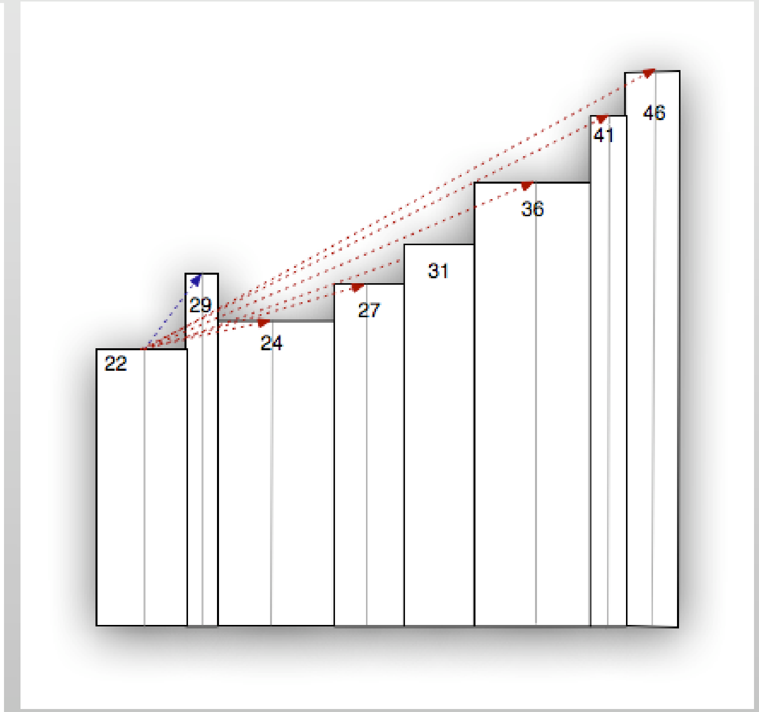
- TIN (triangulated polyhedral terrain)



Visibility on grids

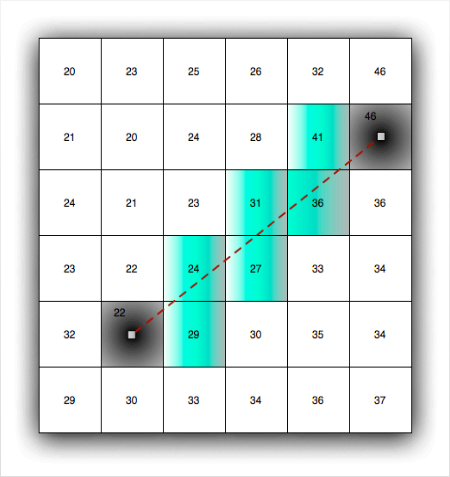
- Line-of-sight model
  - a grid cell with center q is visible from viewpoint v iff the line segment vq does not cross any cell that is above vq



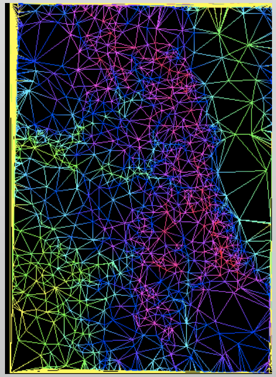


Visibility: Related work

- 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

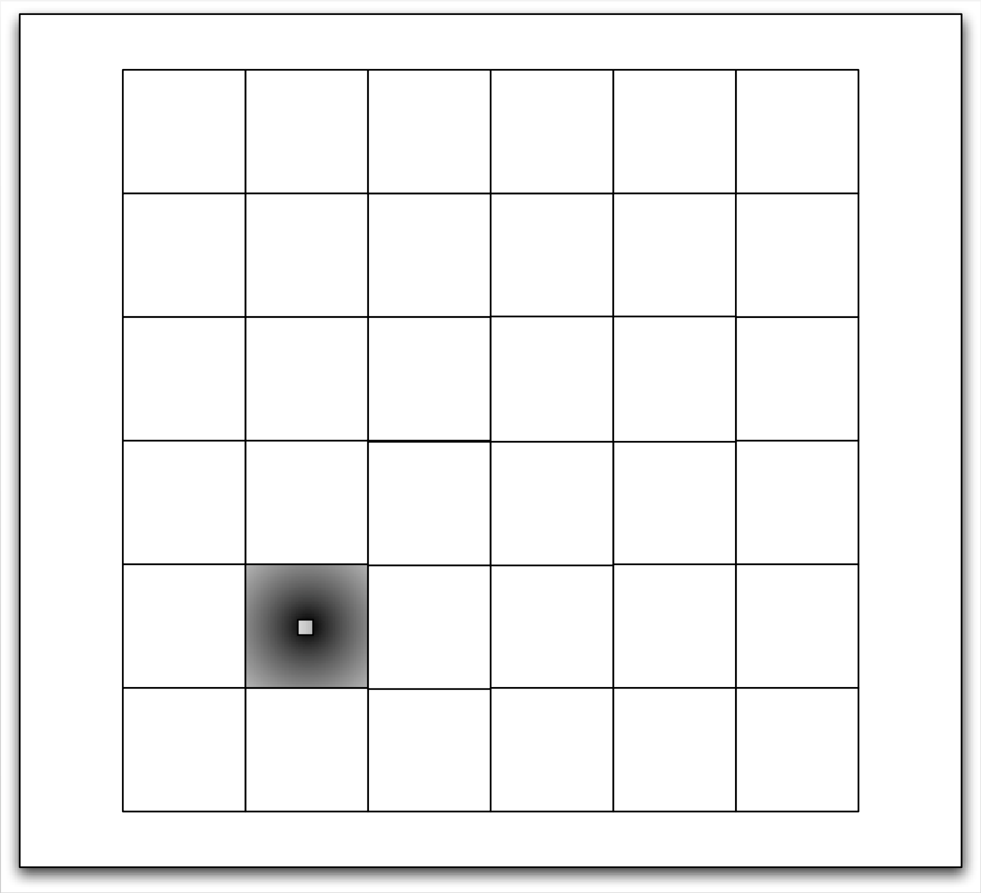


- TINs
  - surveys: de Floriani & Magillo [FM94], Cole & Sharir [CS89]
  - recently: watchtowers and terrain guarding [SoCG'05, SODA'06]

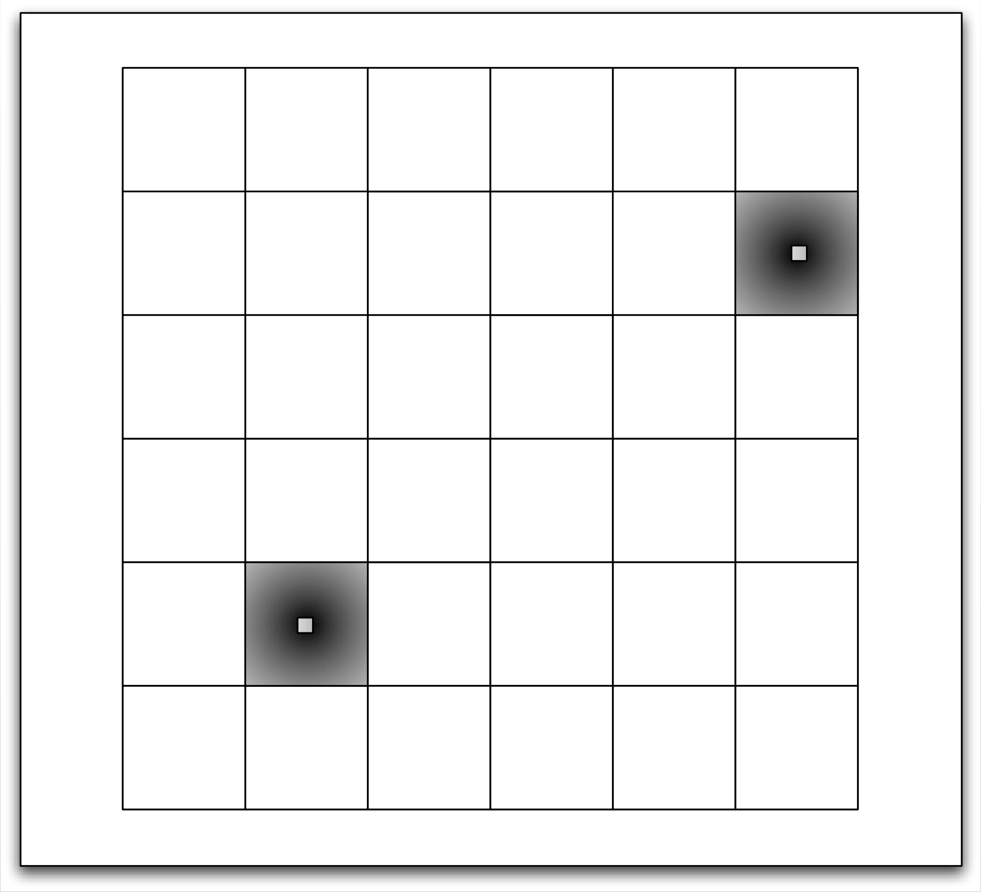


van Kreveld's algorithm

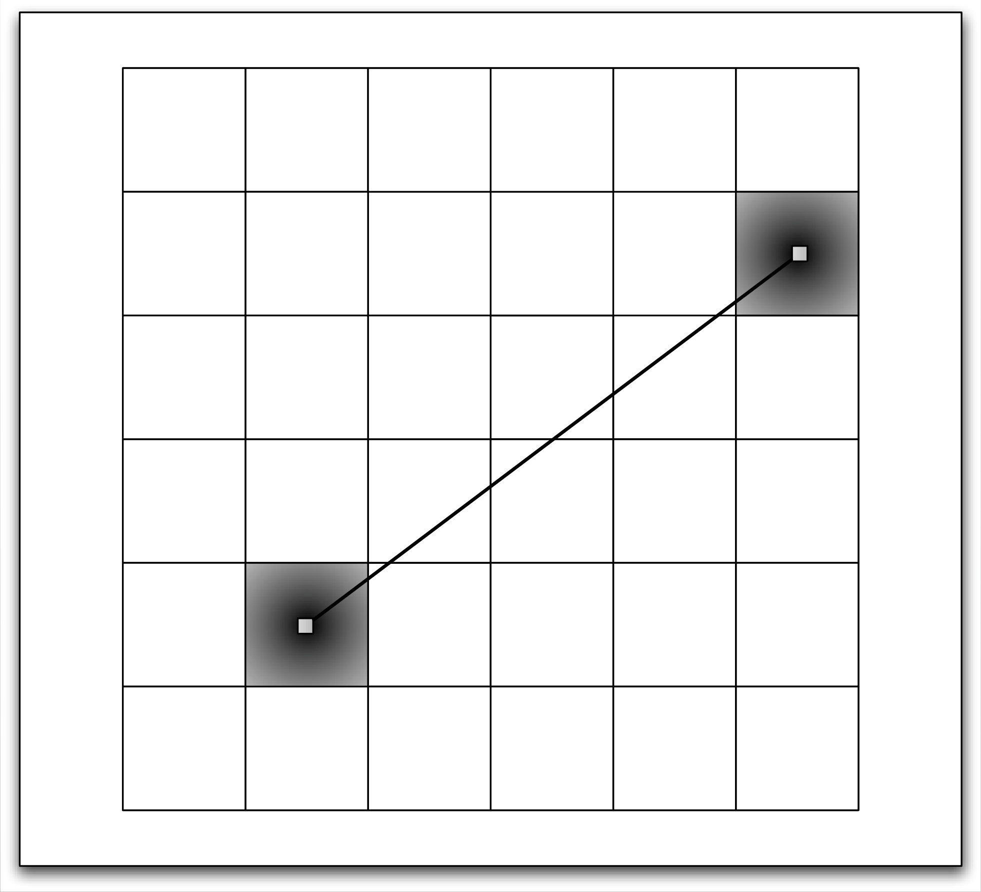
van Kreveld's algorithm



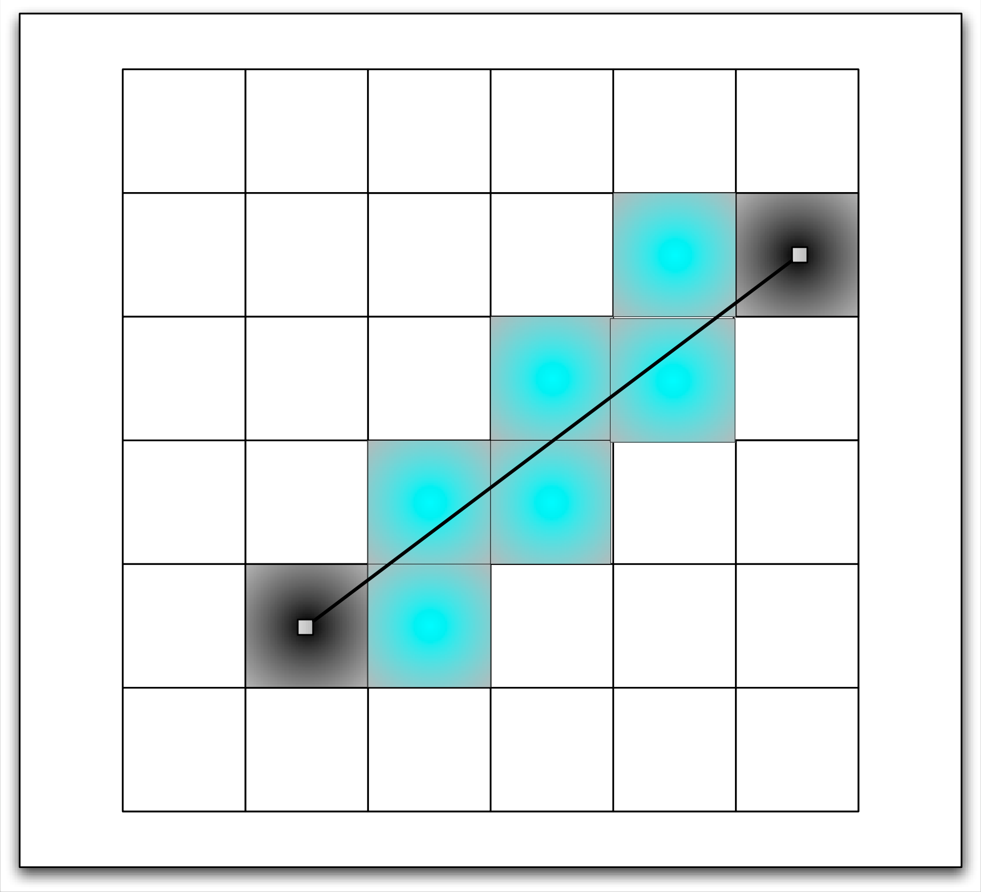
van Kreveld's algorithm



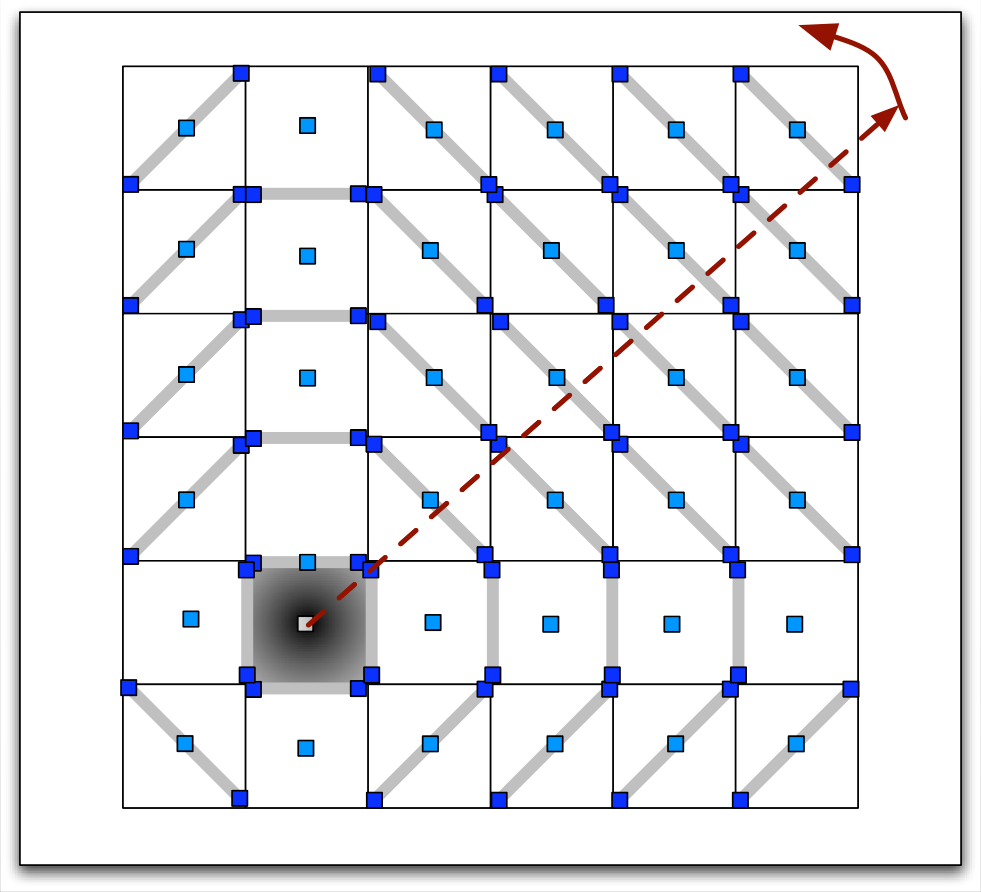
van Kreveld's algorithm



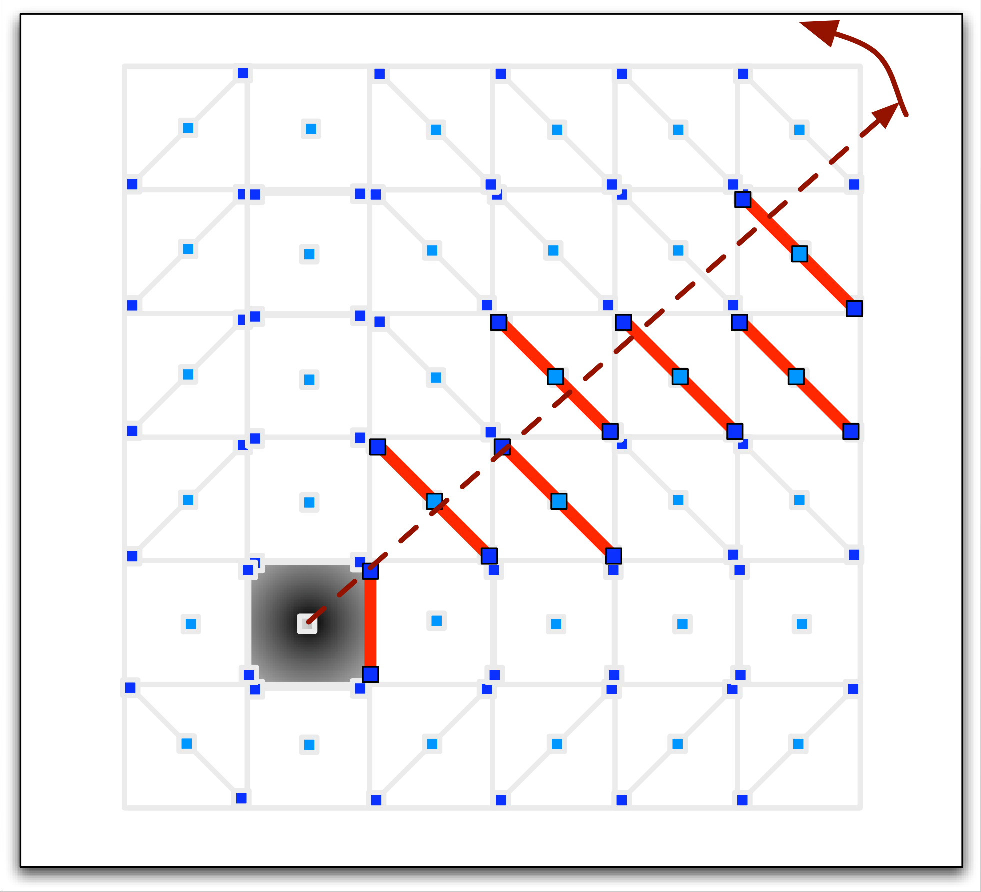
van Kreveld's algorithm



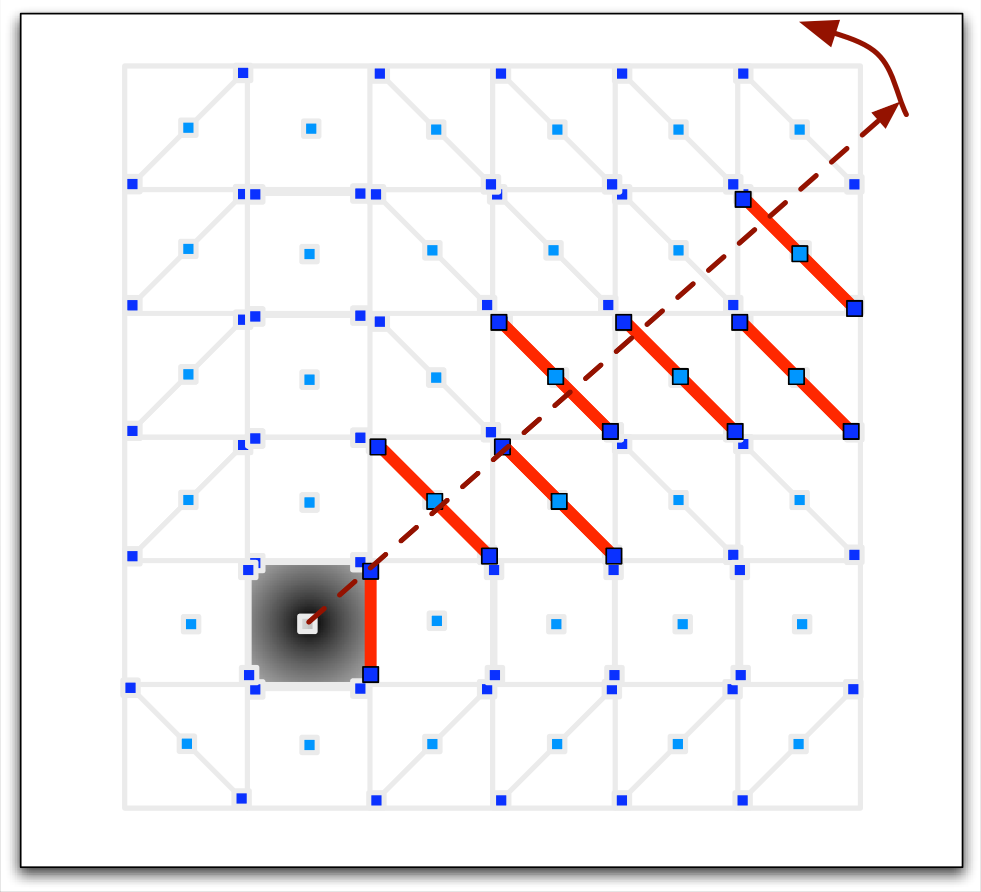
van Kreveld's algorithm

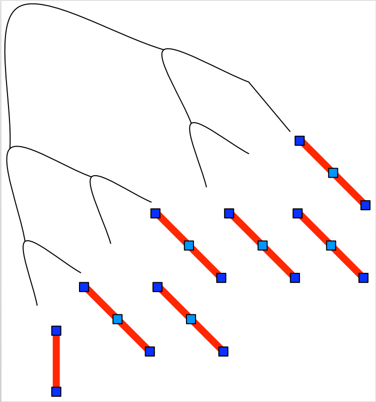


van Kreveld's algorithm

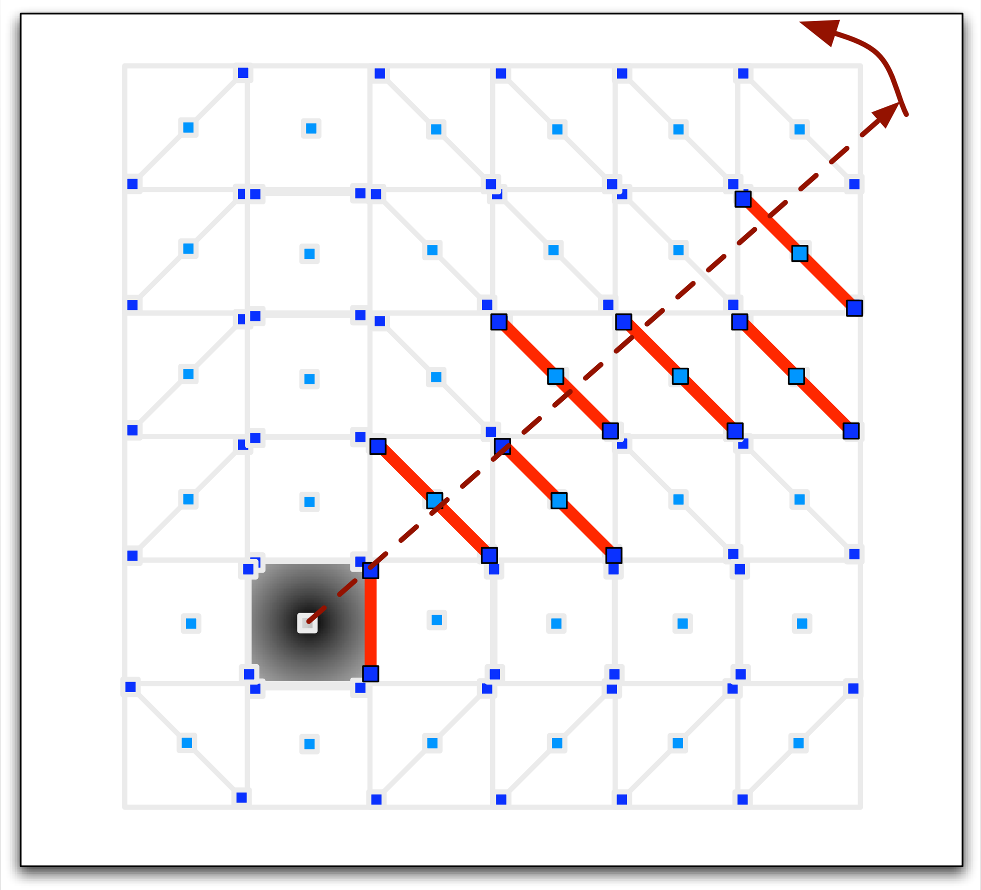


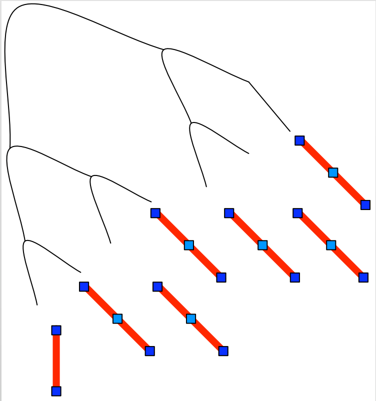
van Kreveld's algorithm





van Kreveld's algorithm





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

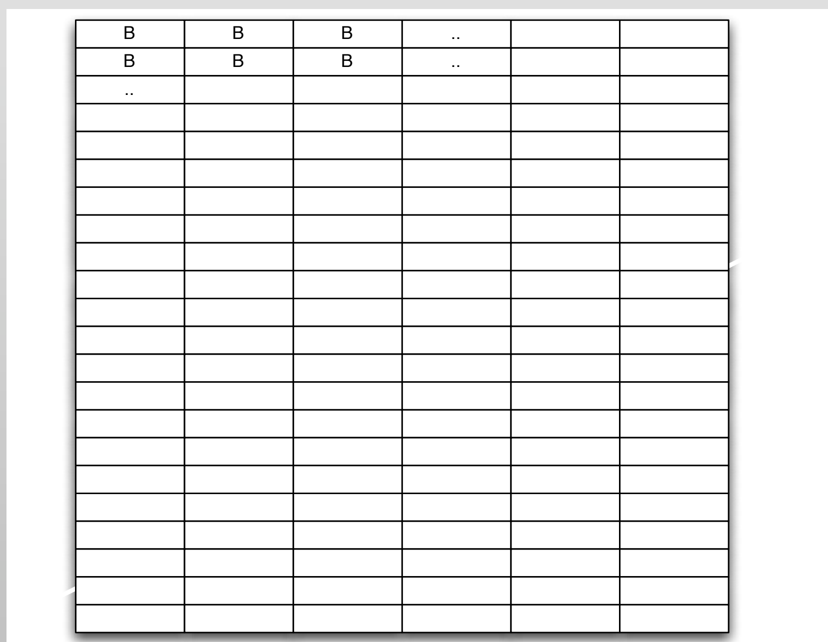
van Kreveld's algorithm

-in external memory-

- Requires 4 structures in memory
    - input elevation grid, output visibility grid
        - stored in row-major order, read in sweep order
    - event list
    - status structure

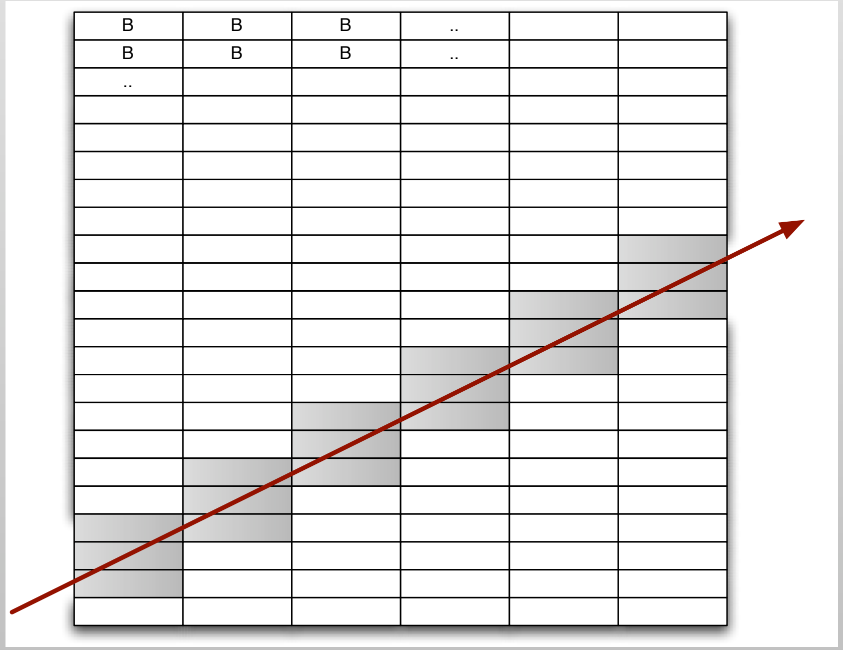
van Kreveld's algorithm
-in external memory-

- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, read in sweep order
  - event list
  - status structure



van Kreveld's algorithm
-in external memory-

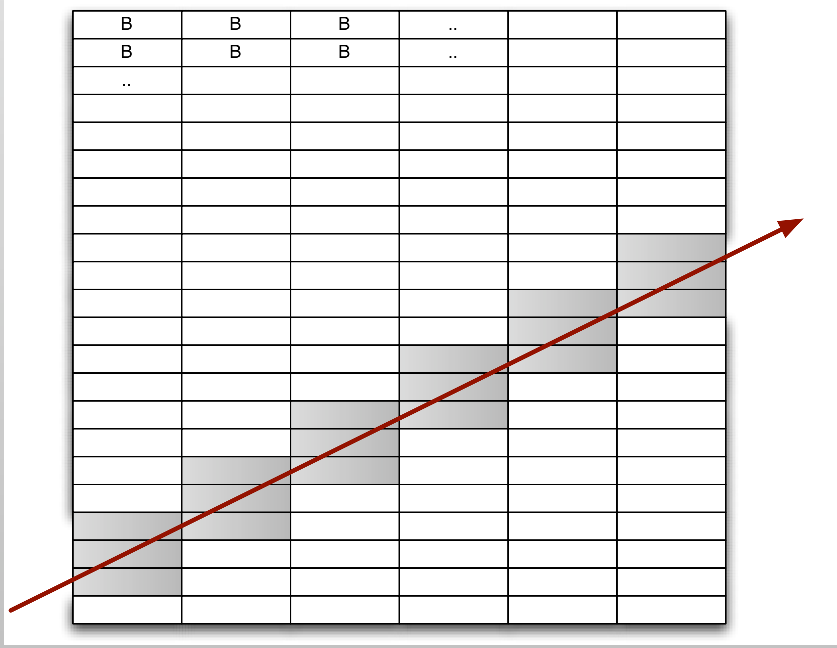
- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, read in sweep order
  - event list
  - status structure



van Kreveld's algorithm

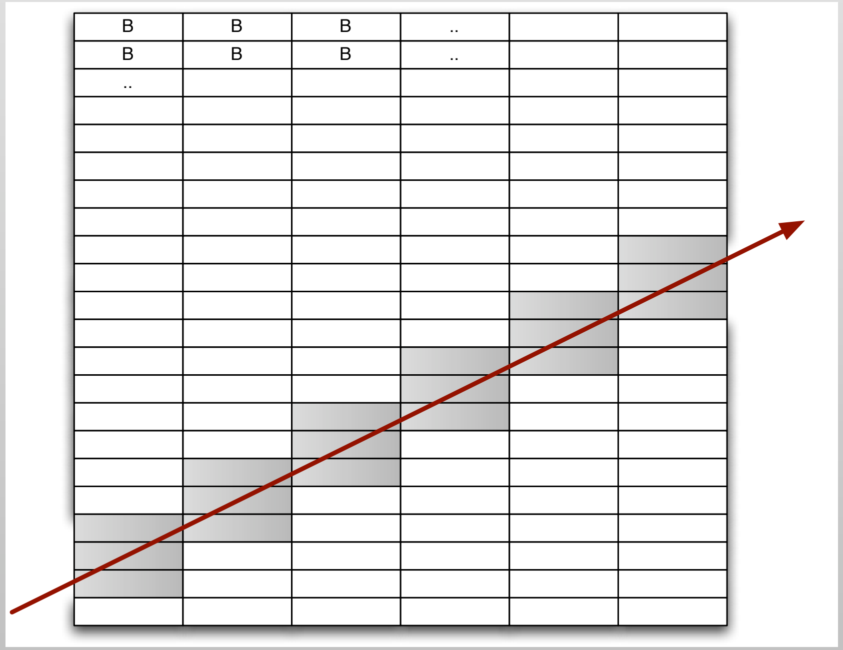
-in external memory-

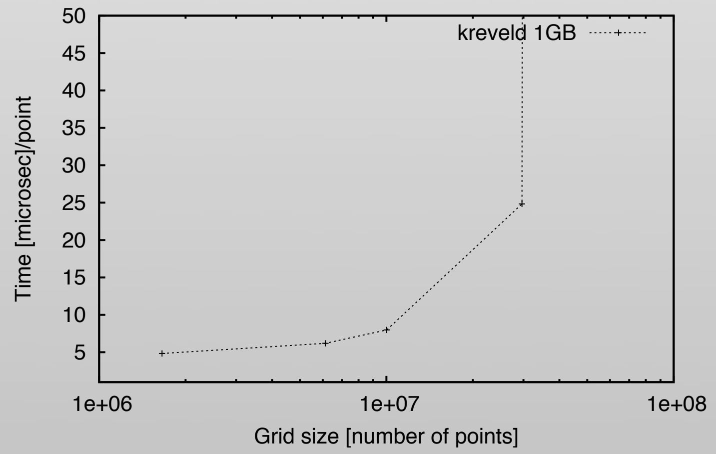
- Requires 4 structures in memory
  - input elevation grid, output visibility grid
    - stored in row-major order, read in sweep order
  - event list
  - status structure
- If n > M: O(1) I/O per element, O(n) I/Os total



van Kreveld's algorithm
-in external memory-

- Requires 4 structures in memory
    - input elevation grid, output visibility grid
        - stored in row-major order, read in sweep order
    - event list
    - status structure
- If n > M: O(1) I/O per element, O(n) I/Os total





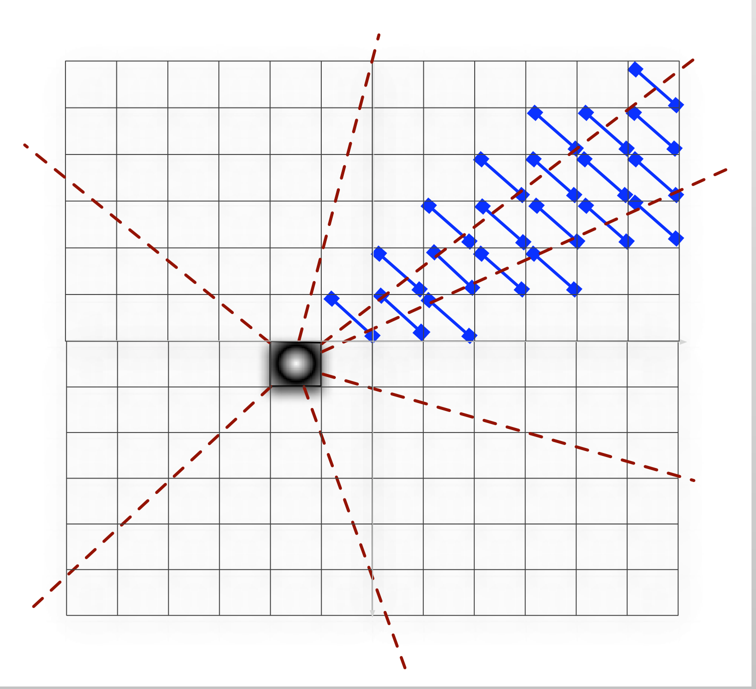
Our results

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

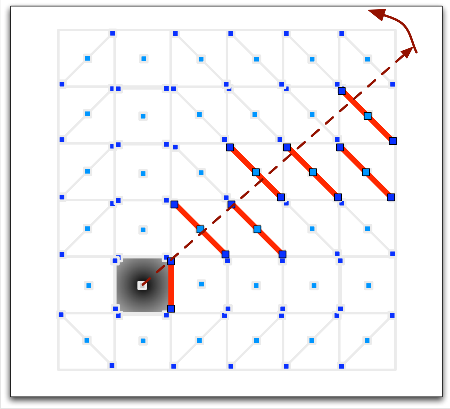
- The visibility grid of an arbitrary viewpoint on a grid of size n can be computed with O(n) space and O(sort(n)) I/Os
- Experimental evaluation
  - ioviewshed
  - standard algorithm (Kreveld)
  - visibility algorithm in GRASS GIS

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

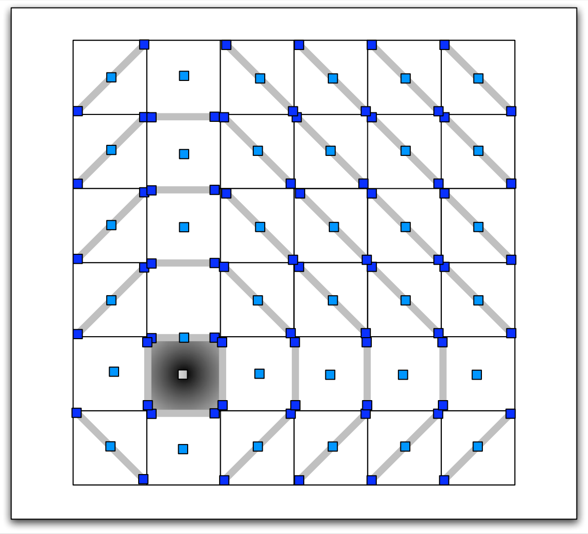


- Usually, stop recursion when n < M
- Our idea: stop when status structure fits in memory

- 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

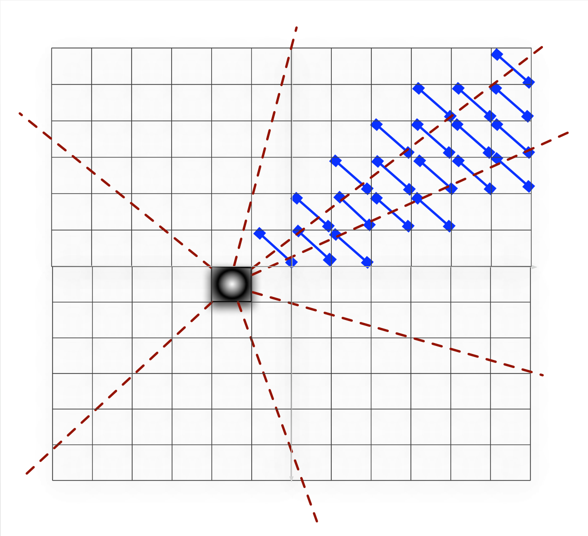
- Total: O(sort(n)) I/Os

The recursion



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

The recursion

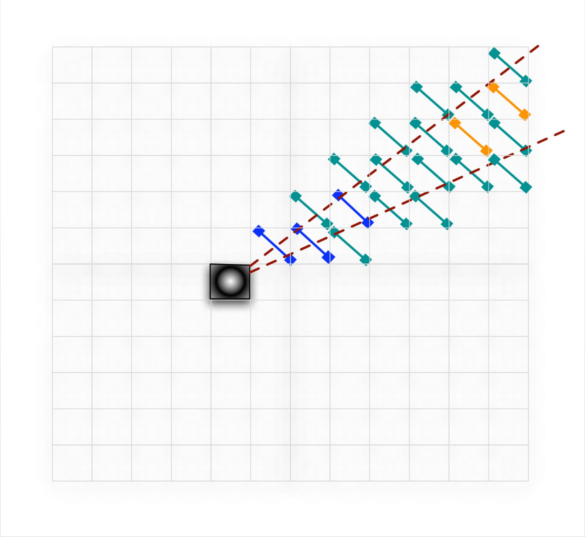


- Divide events into O(M/B) sectors of equal size
- $O(\log_{M/B} n)$ recursion levels

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

The recursion:

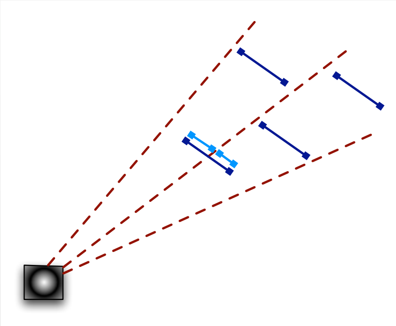
Distributing events to sectors



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

The recursion:

Distributing events to sectors

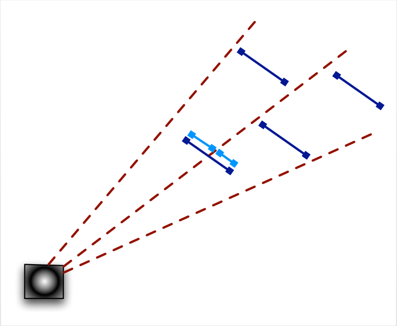


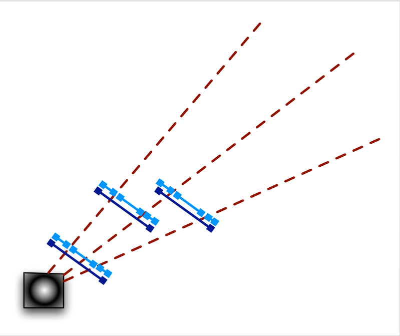
- narrow cells
  - cut and insert in both sectors

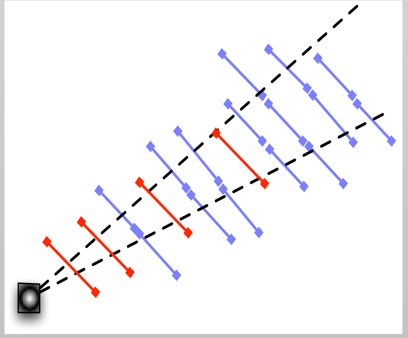
The recursion:

Distributing events to sectors

- narrow cells
  - cut and insert in both sectors
- wide cells
  - cannot insert cell in each sector spanned (space blow-up)
  - the visibility of a cell is determined by
    - the highest of all wide cells that span the sector and are closer to the viewpoint
    - all narrow cells in the sector that are closer to the viewpoint
  - for each sector, process wide cells spanning the sector interleaved with query points and narrow cells in the sector, in increasing order of their distance form viewpoint

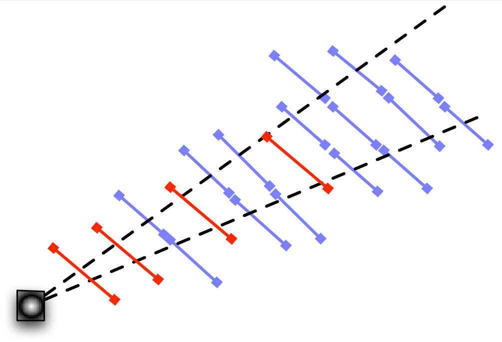






The recursion

- Input: event list in concentric order E_c and in radial order E_r
- Radial sweep: scan E_r
  - find sector boundaries
  - compute a list E_r of events in each sector
- Concentric sweep: scan E_c
  - for each sector
    - keep a block of events in memory
    - maintain the currently highest wide cell spanning the sector, $High_S$
  - if next event in E_c is
    - wide cell: for each sector spanned, update $High_S$ for that sector.
    - narrow cell: if it is not occluded by $High_S$, insert in the buffer of sector. Otherwise skip it.
    - 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

- 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
  - GRASS segment library for virtual memory management
  - bypass the VMS, manage data allocation and de-allocation in segments on disk
  - program will always run (no malloc() fails) but ... slow

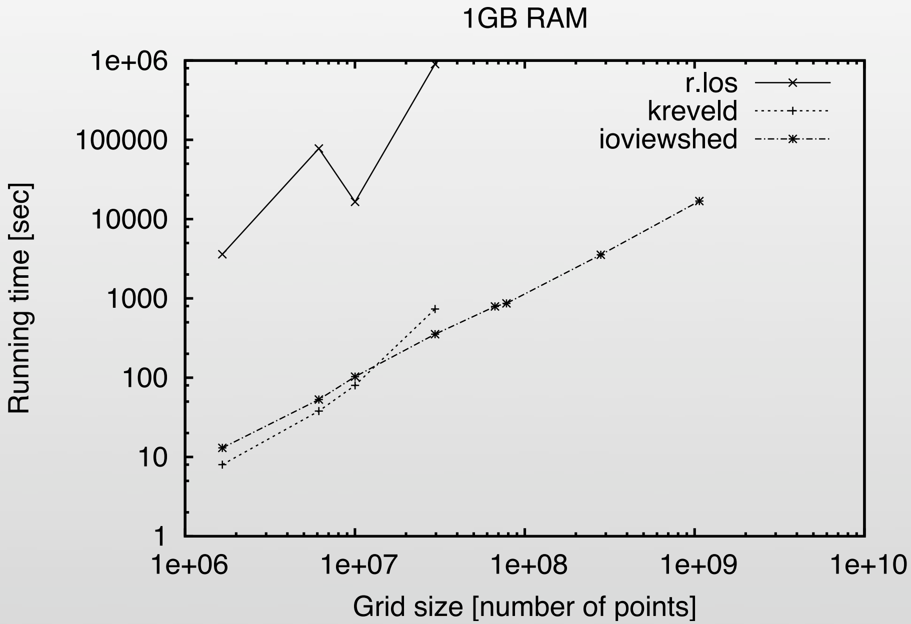
Experimental results

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

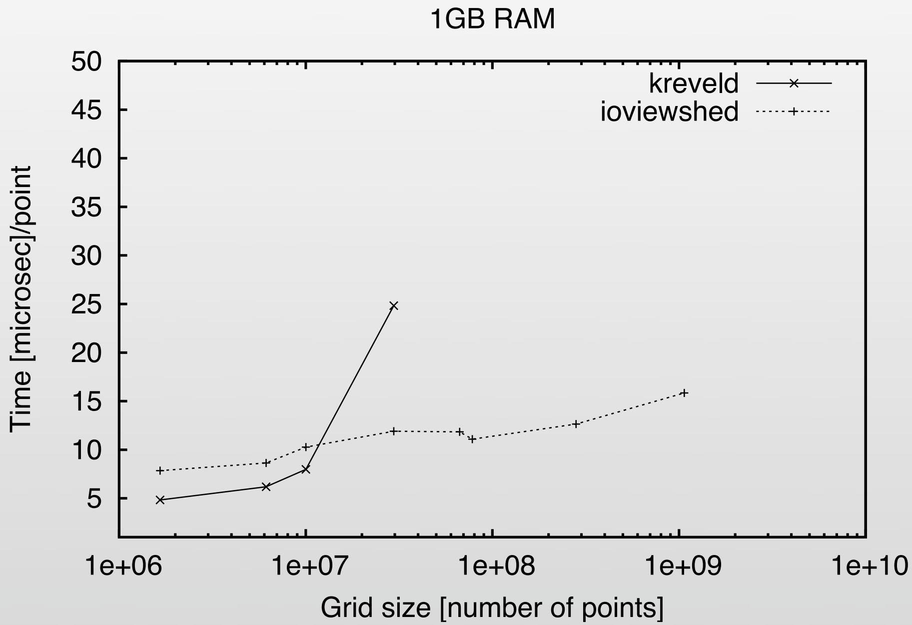
Dataset	Grid Size (million elements)	MB (Grid Only)	Valid size
Kaweah	1.6	6	56%
Puerto Rico	5.9	24	19%
Sierra Nevada	9.5	38	96%
Hawaii	28.2	112	7%
Cumberlands	67	268	27%
Lower New England	77.8	312	36%
Midwest USA	280	1100	86%
Washington	1066	4264	95%

1GB RAM

total time (seconds)

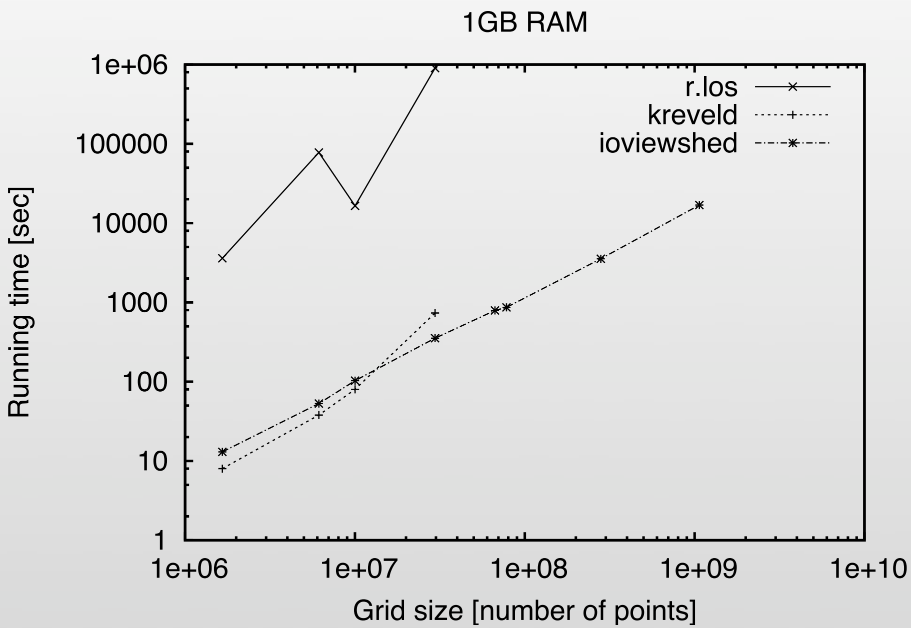


microseconds per grid point

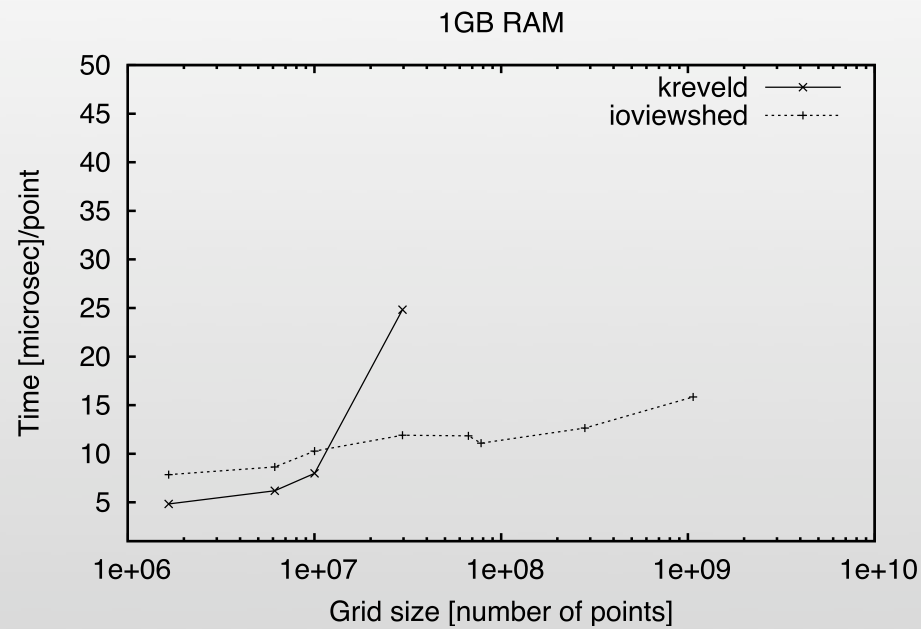


1GB RAM

total time (seconds)



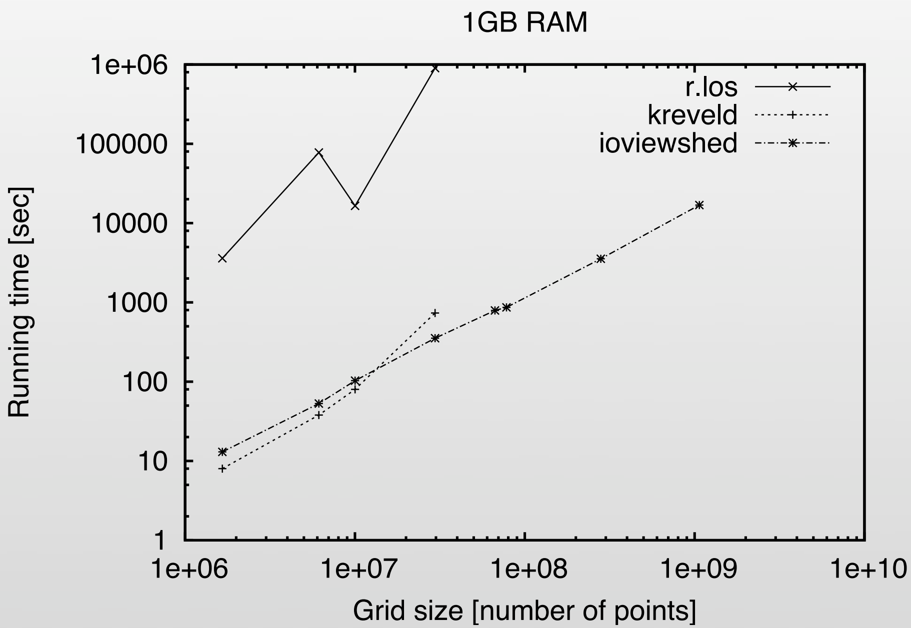
microseconds per grid point



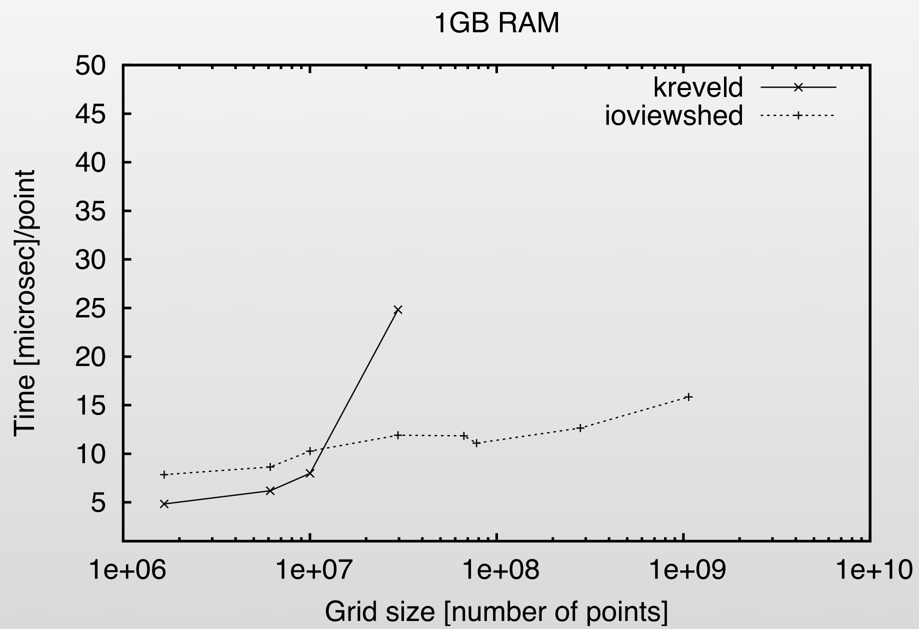
- GRASS
  - program always runs (no malloc() failures) but is very slow
- kreveld
  - starts thrashing on Hawaii (39% CPU, 739 seconds)
  - malloc() fails on Cumberlands
- ioviewshed
  - finishes Washington in 4.5 hours
  - in practice status structure fits in memory, never enters recursion

1GB RAM

total time (seconds)



microseconds per grid point

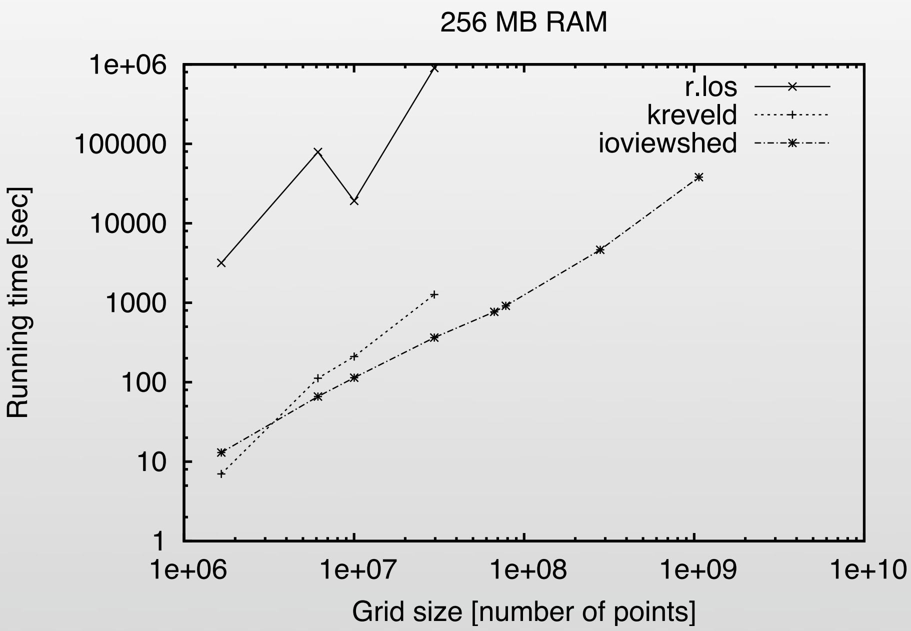


Data set	`r.los`	`kreveld`	`ioviewshed`
Kaweah	2 928	8 (100 %)	13 (84 %)
Puerto Rico	78 778	38 (100 %)	53 (78 %)
Sierra Nevada	16 493	80 (95 %)	102 (67 %)
Hawaii	>1 200 000	736 (39 %)	353 (63 %)
Cumberlands		`malloc` fails	791 (63 %)
LowerNE			865 (64 %)
Midwest USA			3 546 (64 %)
Washington			16 895 (68 %)

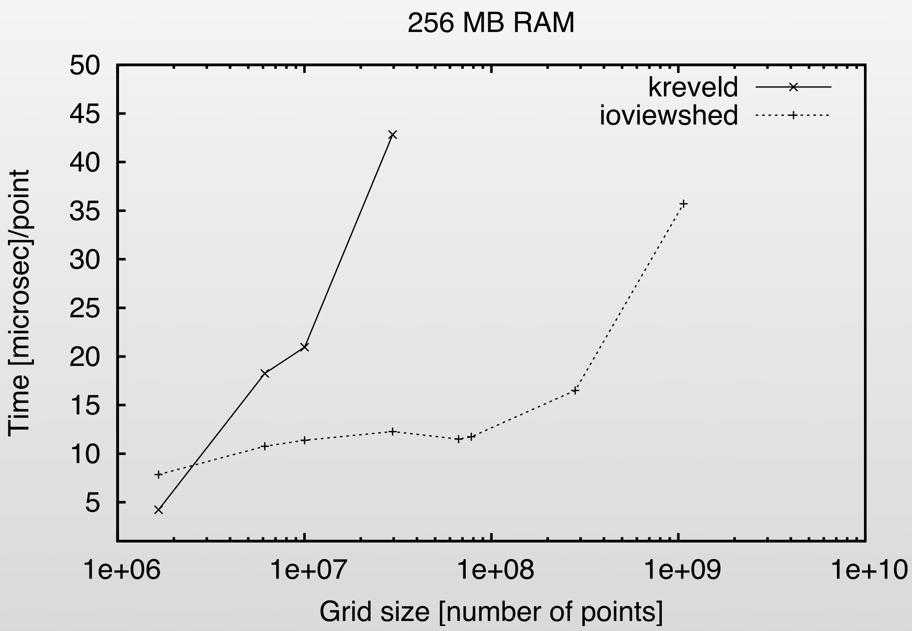
Table 3: Running times (seconds) and CPU-utilization (in parentheses) at 1 GB RAM.

256MB RAM

total time (seconds)



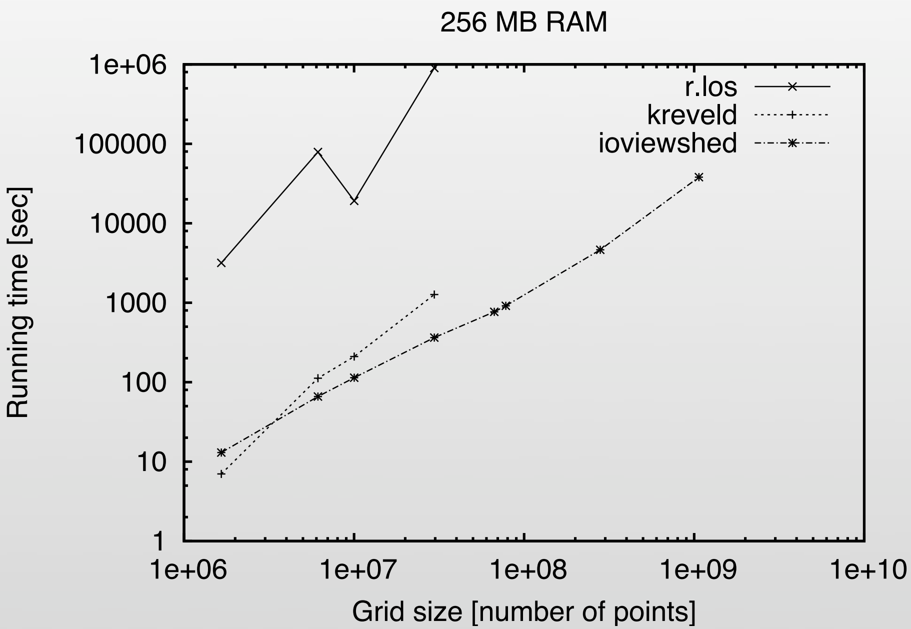
microseconds per grid point



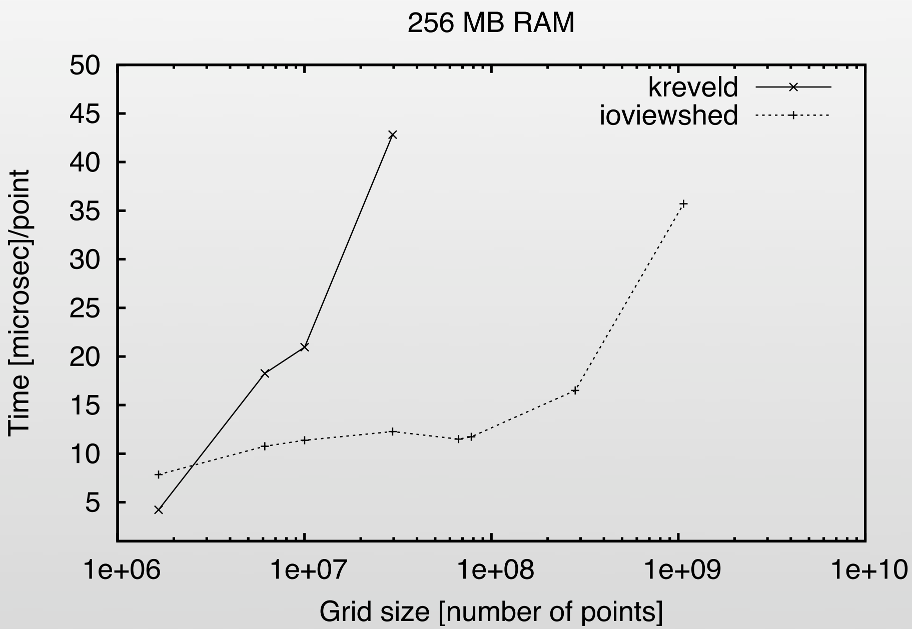
- kreveld starts thrashing earlier (Puerto Rico, 38% CPU)
- ioviewshed slowdown on Washington dataset
  - due 90% to sorting
  - can be improved using customized I/O sorting [TPIE, STXXL]

256MB RAM

total time (seconds)



microseconds per grid point



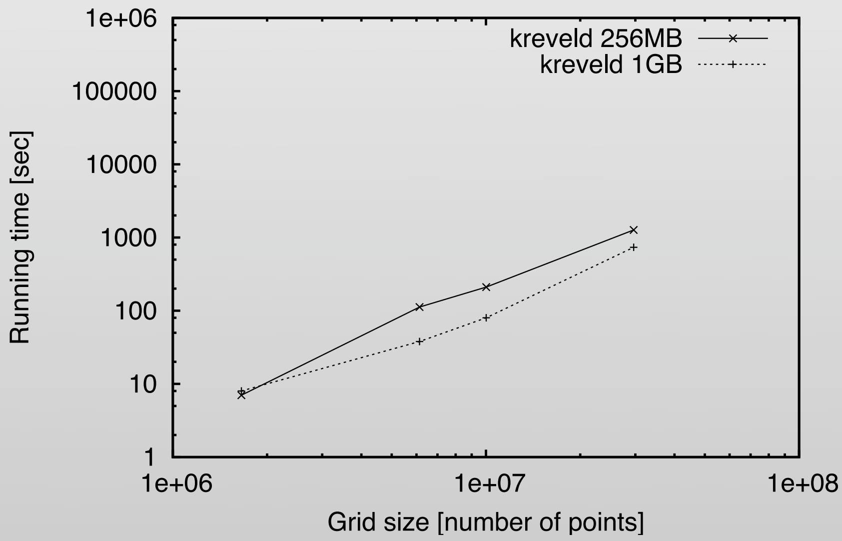
Data set	r.los	kreveld	ioviewshed
Kaweah	2 984	7 (100 %)	13 (77 %)
Puerto Rico	78 941	112 (38 %)	66 (60 %)
Sierra Nevada	19 140	211 (29 %)	115 (57 %)
Hawaii	>1 200 000	1270 (27 %)	364 (63 %)
Cumberlands		malloc fails	768 (62 %)
LowerNE			916 (62 %)
Midwest USA			4 631 (52 %)
Washington			40 734 (30 %)

Table 4: Running times (seconds) and CPU-utilization (in parentheses) at 256 MB RAM.

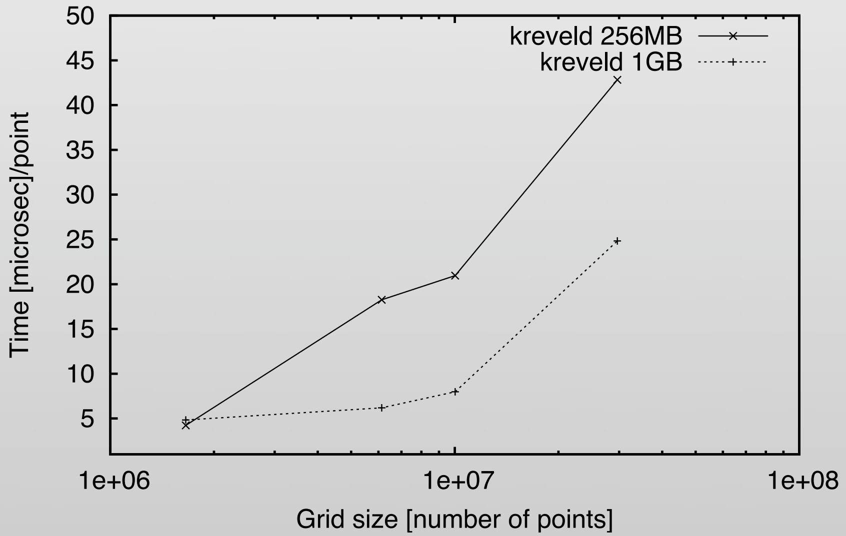
Kaweah	1.6	6	56%
Puerto Rico	5.9	24	19%
Sierra Nevada	9.5	38	96%
Hawaii	28.2	112	7%
Cumberlands	67	268	27%
Lower New England	77.8	312	36%
Midwest	280	1100	86%
Washington	1066	4264	95%

1GB vs. 256MB RAM kreveld

total time (seconds)



microseconds per grid point

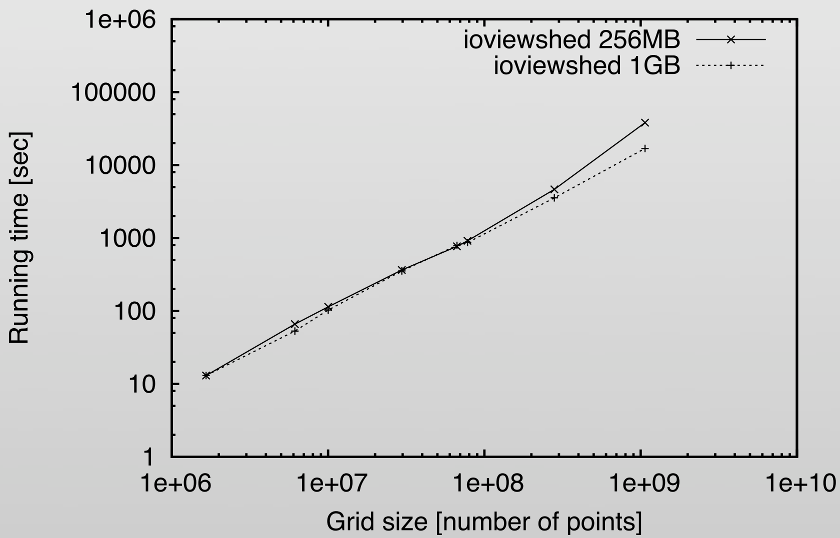


- starts thrashing earlier
  - 1GB: Hawai, 39% CPU
  - 256MB: Puerto Rico 38% CPU

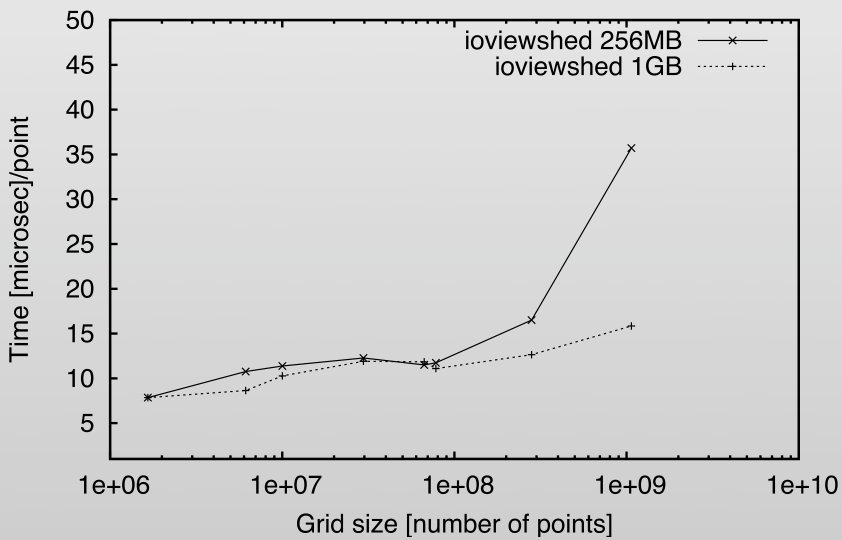
1GB vs. 256MB RAM ioviewshed

Kaweah	1.6	6	56%
Puerto Rico	5.9	24	19%
Sierra Nevada	9.5	38	96%
Hawaii	28.2	112	7%
Cumberlands	67	268	27%
Lower New England	77.8	312	36%
Midwest	280	1100	86%
Washington	1066	4264	95%

total time (seconds)



microseconds per grid point



- slowdown on Washington dataset
- due 90% to sorting
- can be improved using a customized I/O sorting [TPIE, STXXL]

Conclusion

- I/O-efficient visibility computation
    - Theoretically worst-case optimal algorithm
    - In practice status structure fits in memory
        - with extended base case it never enters recursion
    - Scalable
        - Can process grids that are out of scope with traditional algorithm

Thank you.