ICEIS 2014
16th International Conference on Enterprise Information Systems
Lisbon, Portugal · 27 - 30 April, 2014

An improved parallel algorithm using GPU for siting observers on terrain

Guilherme C. Pena
Marcus V. A. Andrade
Salles V. G. Magalhães
W. Randolph Franklin
Chaulio R. Ferreira

Universidade Federal de Viçosa (UFV)
Rensselaer Polytechnic Institute (RPI)

UFV

RPI

Introduction

- Visibility applications play an important role in Geographical Information Systems (GIS).

Introduction

- Visibility applications play an important role in Geographical Information Systems (GIS).

- The focus is to find the points on the terrain that are visible from a particular point (the *observer*).

Introduction

- These applications include telecommunications, security monitoring, observation paths, etc.

Introduction

- These applications include telecommunications, security monitoring, observation paths, etc.
- For example, an “observer” may be a **mobile phone tower**

Introduction

- These applications include telecommunications, security monitoring, observation paths, etc.
- For example, an “observer” may be a mobile phone tower **or an observation tower.**

Introduction

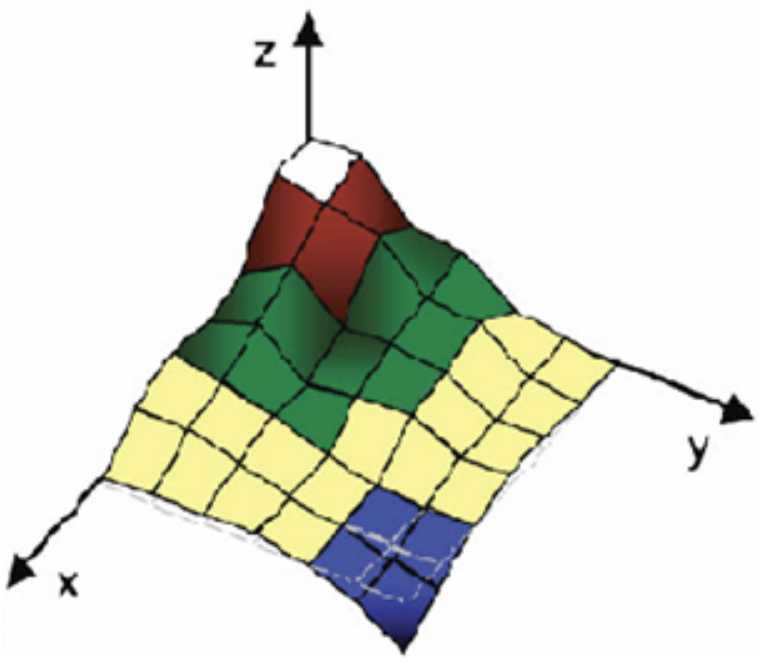
- An important problem is to site observers in order to obtain an optimal visual coverage of a terrain.

Introduction

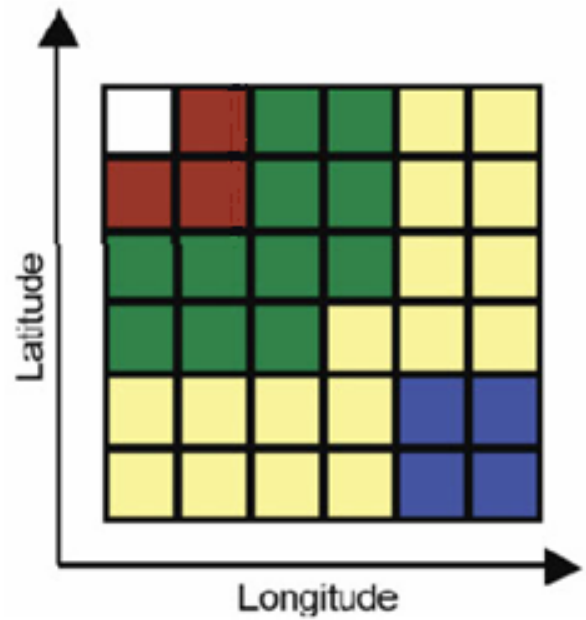
- An important problem is to site observers in order to obtain an optimal visual coverage of a terrain.
- For example, suppose that you want to cover 95% of a terrain.
- How many and where to site observers to achieve this coverage?

Introduction

- We will present a parallel method to solve a variation of the siting observers problem on terrains represented by a digital elevation matrix.



Terrain



Digital Elevation Matrix

Terrain visibility

- An *observer* is a point (in the space) from which we wish to see or communicate with other points, called *targets*.
- The *radius of interest*, *R*, of an observer means the distance that the observer can see.
- For example, for an observation tower, *R* is the maximum distance that a person on the tower can see.

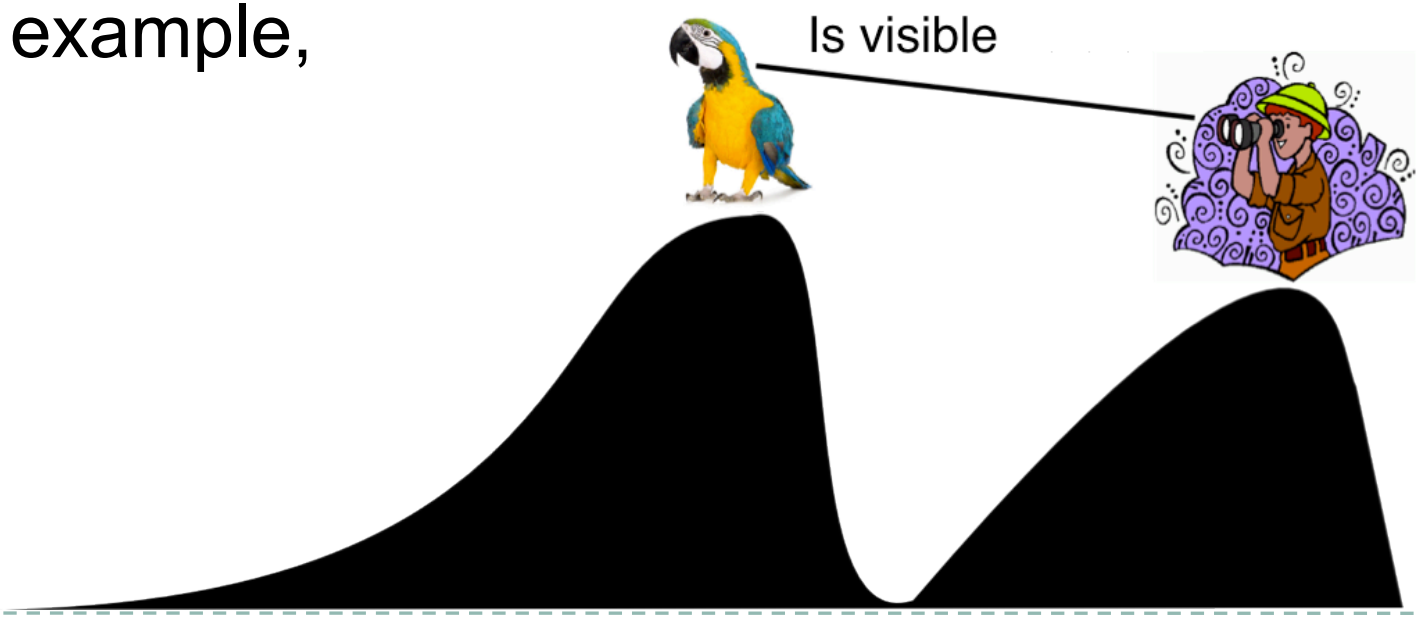
Terrain visibility

- A point is *visible* by the observer if its distance from the observer is, at most, *R,* and if there is no terrain point blocking the line segment connecting the point and the observer.

Terrain visibility

- A point is *visible* by the observer if its distance from the observer is, at most, *R,* and if there is no terrain point blocking the line segment connecting the point and the observer.

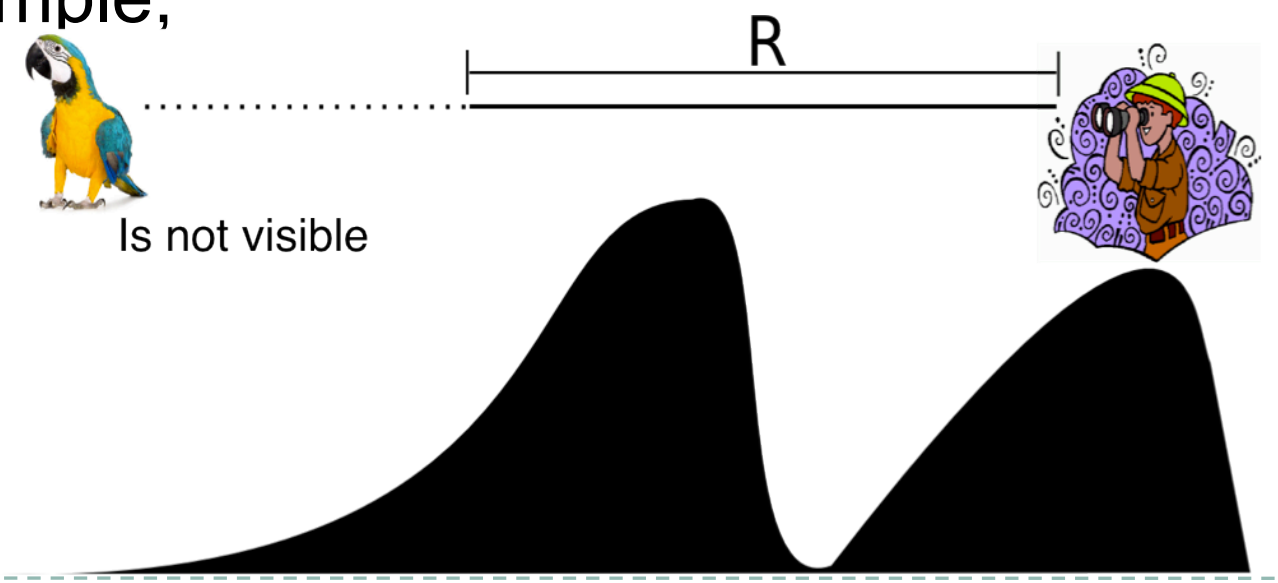
- For example,



Terrain visibility

- A point is *visible* by the observer if its distance from the observer is, at most, *R,* and if there is no terrain point blocking the line segment connecting the point and the observer.

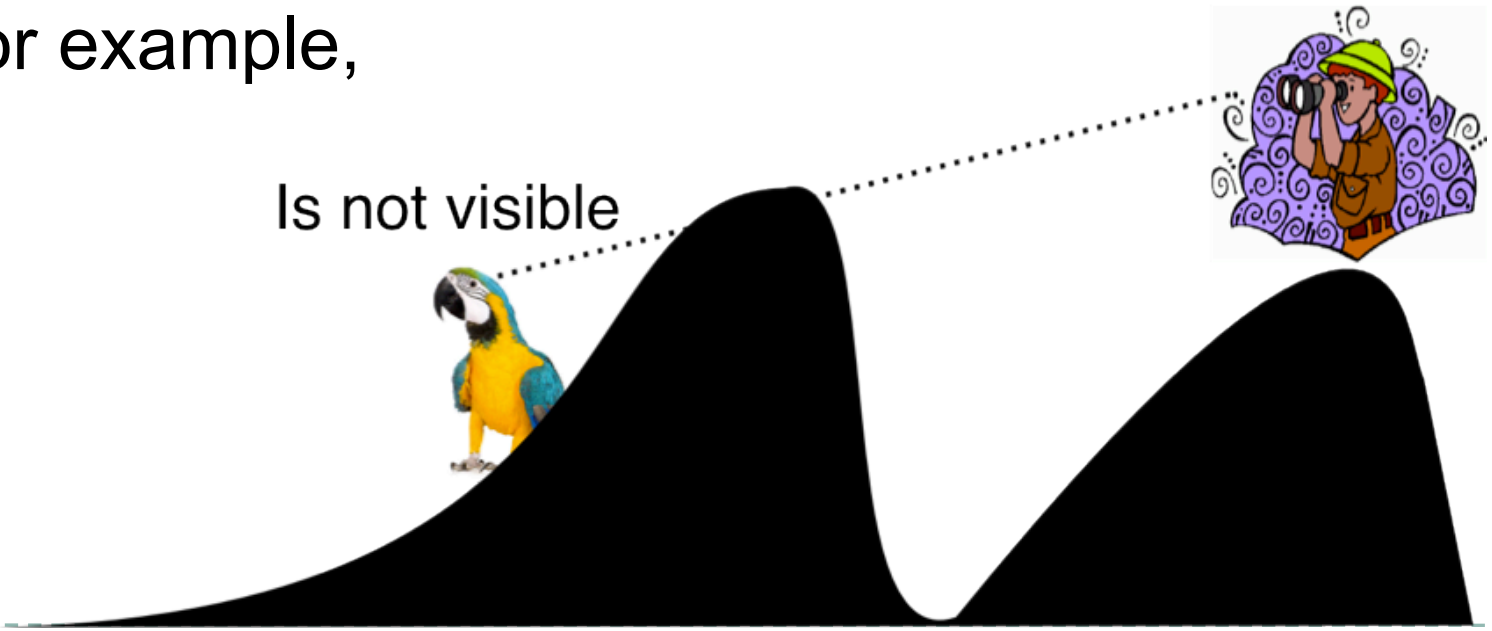
- For example,



Terrain visibility

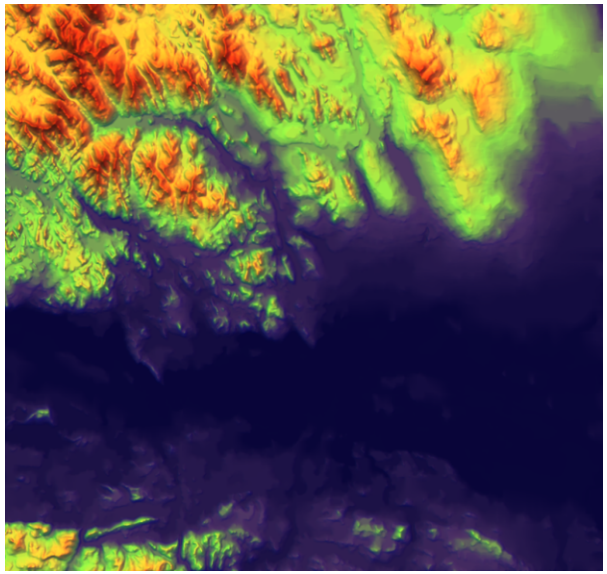
- A point is *visible* by the observer if its distance from the observer is, at most, *R,* and if there is no terrain point blocking the line segment connecting the point and the observer.

- For example,



Terrain visibility

- The *viewshed* of an observer is the set of terrain points whose corresponding targets are visible from it.
- The *visibility index* of an observer is the number of targets that are visible from it.

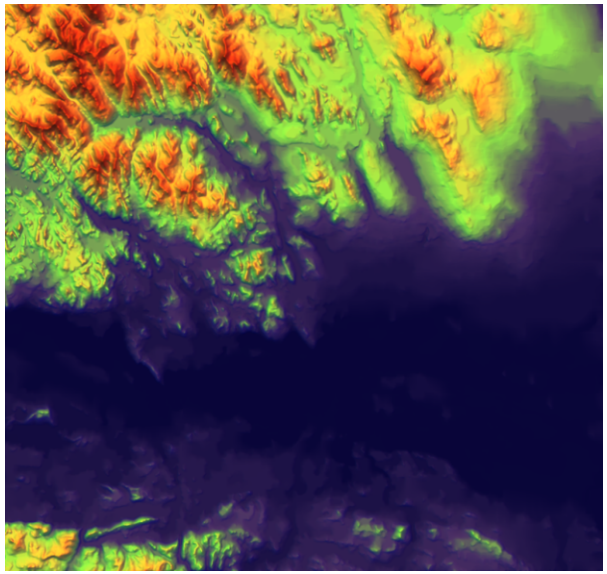


Terrain visualization

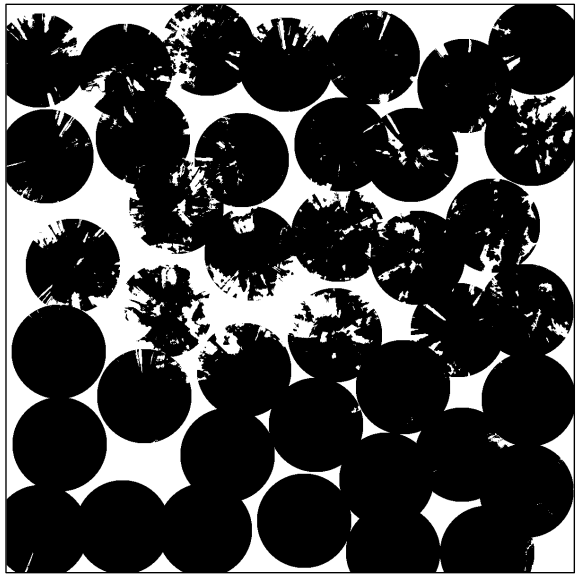
Viewshed

Terrain visibility

- The *joint viewshed* of a set of observers is the union of the individual viewsheds.
- The *joint visibility index* (*VIX*) of a set of observers is the number of targets that are visible from at least one observer in the set.



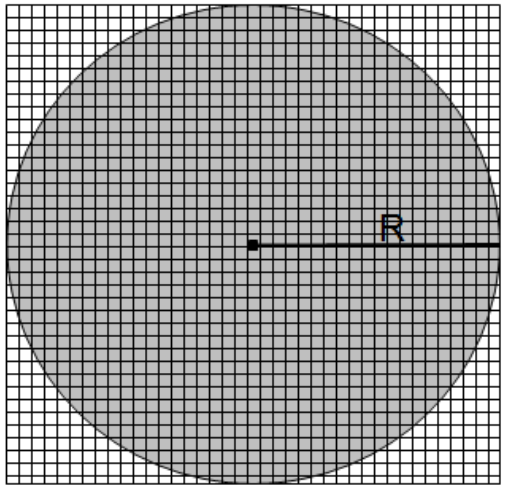
Terrain visualization



Joint viewshed

Terrain visibility

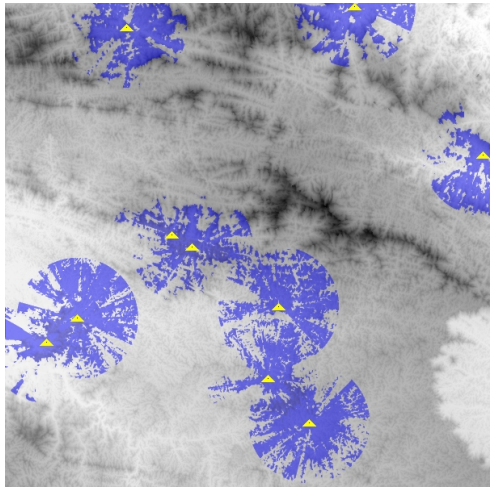
- The viewshed and the joint viewshed are (usually) represented by a square bit matrix of size *2R* x *2R.*

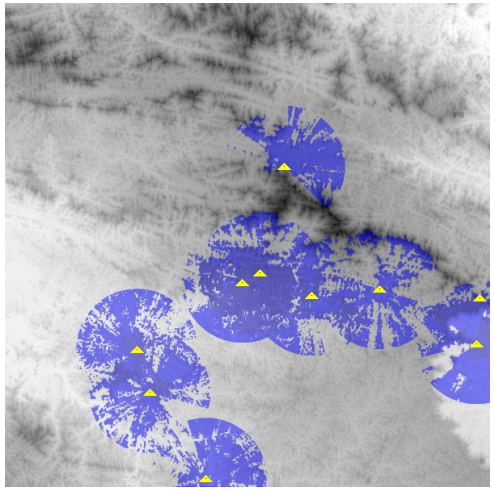


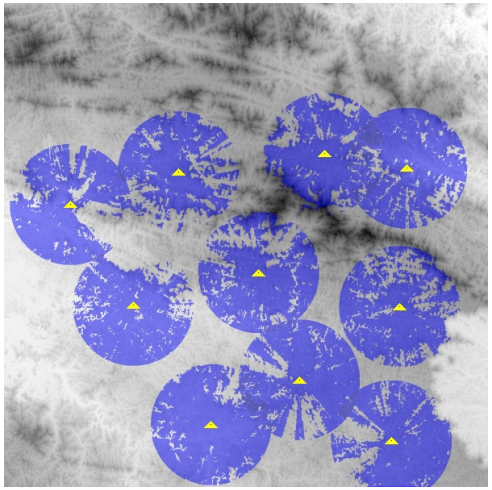
- In this matrix, 1 indicates that the corresponding target is visible and 0 is not.
- Thus, the (joint) visibility index is the number of 1 bits in the matrix.

Observer siting

- *The Multiple Observer Siting Problem*: given a set *P* of (candidate) observers, select *N* observers in *P* such that the joint visibility index of this subset is maximized.
- Example: selecting 10 observers







Observer siting

- This problem is NP-Hard.
- It is (generally) solved using a heuristic.
- We propose an efficient local search strategy to improve the solution obtained by a greedy method.

Multiple observer siting

- A greedy solution: *Site* method (Franklin 2002)
  - Given a terrain, let *P* be a set with the "best" candidate observers;

Multiple observer siting

- A greedy solution: *Site* method (Franklin 2002)
  - Given a terrain, let *P* be a set with the "best" candidate observe

Those observers with the highest visibility index

Multiple observer siting

- A greedy solution: *Site* method (Franklin 2002)
  - Given a terrain, let *P* be a set with the "best" candidate observers;
  - Initialize the solution *S* as empty;

Multiple observer siting

- A greedy solution: *Site* method (Franklin 2002)
  - Given a terrain, let *P* be a set with the "best" candidate observers;
  - Initialize the solution *S* as empty;
  - Then, iteratively, select the observer (in *P*) that will most increase the current joint visibility index of *S* and insert this observer in *S;*

Multiple observer siting

- A greedy solution: *Site* method (Franklin 2002)
  - Given a terrain, let *P* be a set with the "best" candidate observers;
  - Initialize the solution *S* as empty;
  - Then, iteratively, select the observer (in *P*) that will most increase the current joint visibility index of *S* and insert this observer in *S;*
  - Repeat the last operation until a termination condition is satisfied.

Multiple observer siting

- A greedy solution: *Site* method (Franklin 2002)
  - Given a terrain, let *P* be a set with the "best" candidate observers;
  - Initialize t
  - Then, iter at will most ty index of S
  - Repeat the last operation until a termination condition is satisfied.

Typically, until a minimum visual coverage has been achieved or a maximum number of observers has been selected.

Multiple observer siting

- The solution obtained by the greedy method is (mostly) not optimal.
- We propose a strategy (to try) to increase the terrain coverage preserving the number of observers selected.

Multiple observer siting

- The solution obtained by the greedy method is (mostly) not optimal.
- We propose a strategy (to try) to increase the terrain coverage preserving the number of observers selected.
- This may reduce the number of observers required to achieve the desired coverage.
- It may represent an important improvement since an "observer" can be an expensive facility, for example, a communication tower.

Our propose

- Extend the greedy method including an *improvement step* to try to increase the joint visibility index of each current partial solution.

- This improvement step checks if the joint visibility index (of a partial solution) can be increased replacing an observer in the solution with another one did not select yet.

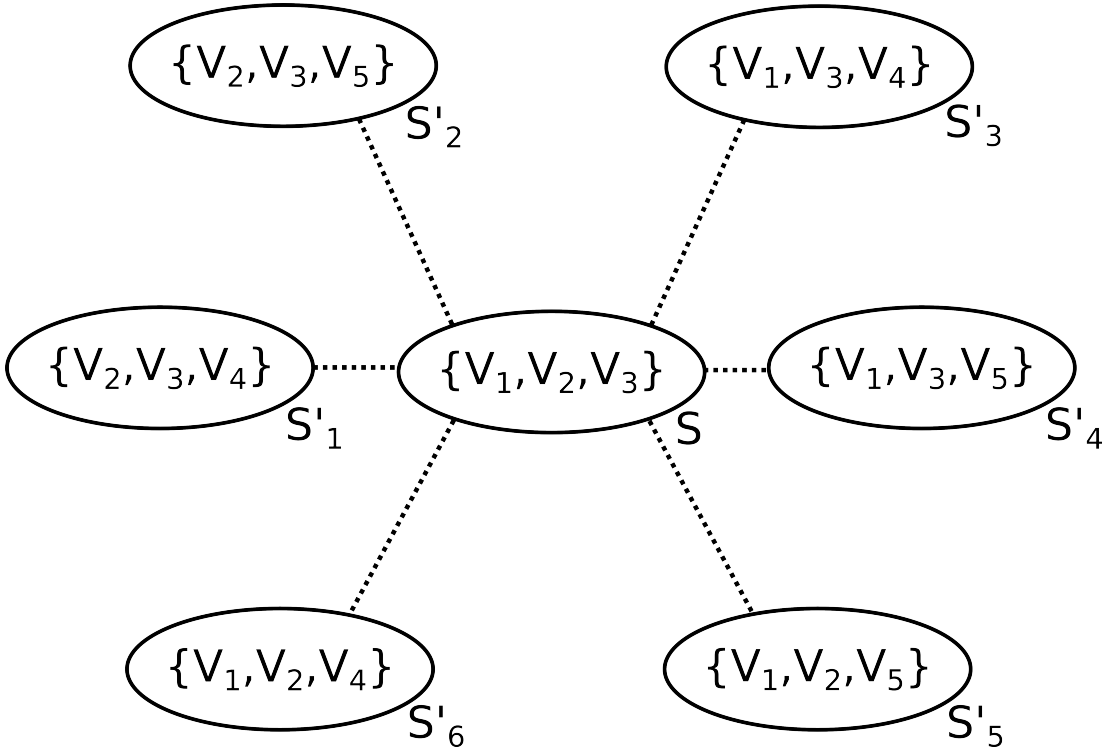
Our propose

- This checking step performs a *local search* whose goal is to select the best *neighbor solution.*

- A neighbor solution of a solution *S* is a solution *S'* where an observer in *S* is replaced with another observer not in *S.*

Our propose – local search

- **For example**: Suppose *P* with 5 observers whose viewsheds are V_1, V_2, …, V_5 and let $S=\{V_1, V_2, V_3\}$ be a partial solution. Thus, the neighbors of *S* are



Our propose – local search

- In each iteration of the greedy method, the local search is repeated until to obtain a solution having no better neighbor (a local optimal).

Our propose – local search

- In each iteration of the greedy method, the local search is repeated until to obtain a solution having no better neighbor (a local optimal).

- This approach is very time consuming.

Our propose – local search

- In each iteration of the greedy method, the local search is repeated until to obtain a solution having no better neighbor (a local optimal).
- This approach is very time consuming.
- The greedy method requires a lot of processing time.

Our propose – local search

- In each iteration of the greedy method, the local search is rep
having no bet

In each iteration, it is necessary to check all candidate observers to select the one that will most increase the joint visibility index

- This approac

- The greedy method requires a lot of processing time.

Our propose – local search

- In each iteration of the greedy method, the local search is repeated until to obtain a solution having no better neighbor (a local optimal).
- This approach is very time consuming.
- The greedy method requires a lot of processing time.
- The local search is still worse: it has to evaluate all neighbors of each partial solution.

Our propose – local search

- In each iteration of the greedy method, the local search is repeated until to obtain a solution having no better neighbor (a local optimal).
- This approach is very time consuming.
- The greedy m ng time.
- The local search is still worse: it has to evaluate all neighbors of each partial solution.

Each observer (in the partial solution) is replaced with all the other observers non selected yet.

Local search: an efficient implementation

- The local search bottleneck is the computation of the visibility index of all neighbor solutions.

Local search: an efficient implementation

- The local search bottleneck is the computation of the visibility index of all neighbor solutions.

- Let $P = \{p_1,\ldots,p_n\}$ be the candidate set and $S = \{s_1,\ldots, s_k\}$ be a partial solution.

Local search: an efficient implementation

- The local search bottleneck is the computation of the visibility index of all neighbor solutions.

- Let $P = \{p_1,\ldots,p_n\}$ be the candidate set and $S = \{s_1,\ldots, s_k\}$ be a partial solution.

- The neighbors of S are

$$S'_{ij} = S \setminus \{s_i\} \cup \{p_j\}$$

for all $i=1,..,k$ and $j=1,..,n$ with $i \neq j$ *and* $p_j \notin S$

Local search: an efficient implementation

- The visibility indices computation can be subdivided in two steps:

  ① Create an array B of size k and for $i=1,\ldots,k$, store in $B[i]$ the joint viewshed of $S \setminus \{s_i\}$;

  ② Create a matrix V of size k x n and for each $i=1,\ldots,k$ and $j=1,\ldots,n$, with $j \neq i$, store in $V[i,j]$ the visibility index of the joint viewshed obtained overlapping $B[i]$ with the viewshed of the observer p_j.

Local search: an efficient implementation

- A straightforward implementation of step 1 is:

```
for i ← 1 to k do
   for m ← 1 to k do
      if m ≠ i then
         // overlap B[i] with S[m]
         B[i] ← B[i] ⊕ S[m]
```

Local search: an efficient implementation

- A straightforward implementation of step 1 is:

```
for i ← 1 to k do
   for m ← 1 to k do
      if m ≠ i then
         // overlap B[i] with S[m]
         B[i] ← B[i] ⊕ S[m]
```

Overlapping two matrices: the joint viewshed B_i and the viewshed of the observer p_m

Local search: an efficient implementation

- A straightforward implementation of step 1 is:

```
for i ← 1 to k do
   for m ← 1 to k do
      if m ≠ i then
         // overlap B[i] with S[m]
         B[i] ← B[i] ⊕ S[m]
```

- This code performs $\Theta(k^2)$ overlapping operations;
- We can make much better using dynamic programming.

Local search: an efficient implementation

- Suppose the partial solution *S* has 5 observers, that is, $S = \{S_1, \ldots, S_5\}$.

- Then, the computation of *B* would require the overlapping of the following viewsheds:

B[1] =	S[2]	S[3]	S[4]	S[5]
B[2] =	S[1]	S[3]	S[4]	S[5]
B[3] =	S[1]	S[2]	S[4]	S[5]
B[4] =	S[1]	S[2]	S[3]	S[5]
B[5] =	S[1]	S[2]	S[3]	S[4]

Local search: an efficient implementation

- Suppose the partial solution S has 5 observers, that is, $S = \{S_1, \ldots, S_5\}$.
- Then, the computation of [illegible] overlapping of the following v[illegible]eds:

The matrix with all B's can be split in the following way

B[1] =	S[2]	S[3]	S[	S[5]
B[2] =	S[1]	S[3]	[illegible][4]	S[5]
B[3] =	S[1]	S[2]	S[4]	S[5]
B[4] =	S[1]	S[2]	S[3]	S[5]
B[5] =	S[1]	S[2]	S[3]	S[4]

Local search: an efficient implementation

- The computation of the matrix storing all B's can be rewritten as following:

B[1]=	S[2]	S[3]	S[4]	S[5]
B[2]=	S[1]	S[3]	S[4]	S[5]
B[3]=	S[1]	S[2]	S[4]	S[5]
B[4]=	S[1]	S[2]	S[3]	S[5]
B[5]=	S[1]	S[2]	S[3]	S[4]

=

S[1]			
S[1]	S[2]		
S[1]	S[2]	S[3]	
S[1]	S[2]	S[3]	S[4]

+

S[2]	S[3]	S[4]	S[5]
	S[3]	S[4]	S[5]
		S[4]	S[5]
			S[5]

Local search: an efficient implementation

- The computation of the matrix storing all B's can be rewritten as following:

B[1]=	S[2]	S[3]	S[4]	S[5]
B[2]=	S[1]	S[3]	S[4]	S[5]
B[3]=	S[1]	S[2]	S[4]	S[5]
B[4]=	S[1]	S[2]	S[3]	S[5]
B[5]=	S[1]	S[2]	S[3]	S[4]

=

S[1]			
S[1]	S[2]		
S[1]	S[2]	S[3]	
S[1]	S[2]	S[3]	S[4]

+

S[2]	S[3]	S[4]	S[5]
	S[3]	S[4]	S[5]
		S[4]	S[5]
			S[5]

- Let *L* be the left (blue) matrix and *R* be the right (orange) matrix.
- These two matrices can be computed separately using an efficient iteration.

Local search: an efficient implementation

- Generalizing, for any *i* = *2,…k-1,*

$$B_i = S_1 \oplus \cdots \oplus S_{i-1} \oplus S_{i+1} \oplus \cdots \oplus S_k$$

Local search: an efficient implementation

- Generalizing, for any *i* = *2,…k-1,*

$$B_i = \underbrace{S_1 \oplus \cdots \oplus S_{i-1}}_{L_i} \oplus \underbrace{S_{i+1} \oplus \cdots \oplus S_k}_{R_i}$$

$$B_i = L_i \oplus R_i$$

Local search: an efficient implementation

- Generalizing, for any $i = 2,\ldots k\text{-}1$,

$$B_i = \underbrace{S_1 \oplus \cdots \oplus S_{i-1}}_{L_i} \oplus \underbrace{S_{i+1} \oplus \cdots \oplus S_k}_{R_i}$$

$$B_i = L_i \oplus R_i$$

- And the values of L and R can be computed by the following recurrences:

$$L_1 = \Phi \text{ and } L_i = L_{i-1} \oplus S_{i-1} \quad \text{for } i=2,\ldots,k$$

$$R_k = \Phi \text{ and } R_i = S_{i+1} \oplus R_{i+1} \quad \text{for } i=k\text{-}1,\ldots,1$$

Local search: an efficient implementation

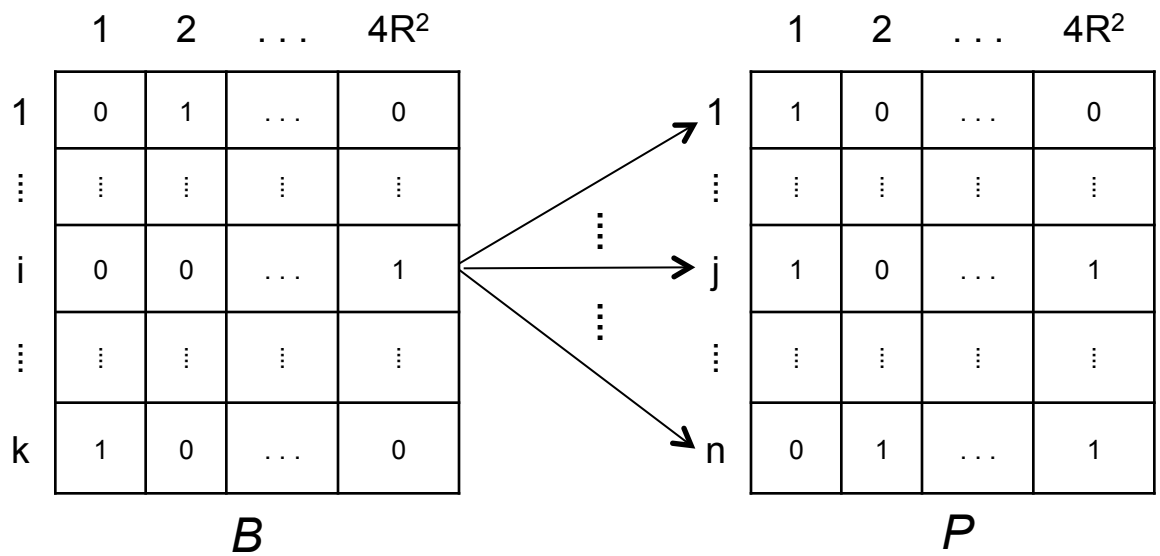
- Thus, the step 1 can be computed performing $\Theta(k)$ overlapping operations:
  - k to compute L;
  - k to compute R;
  - k to overlap L and R

Local search: an efficient implementation

- In step 2, to compute the matrix *V*:
  - each joint viewshed stored in *B* is overlapped with the viewshed of each candidate observer did not include in the solution yet;
  - the number of 1 bits in the resulting joint viewshed is counted.

Matrix *V* computation

- Supposing the viewsheds are linearized and stored in a matrix *P;*
- Each *V*[*i,j*], for *i=1,..,k* and *j=1,…,n,* is the number of 1 bits in the overlapping of *B*[*i*] with *P*[*j*]



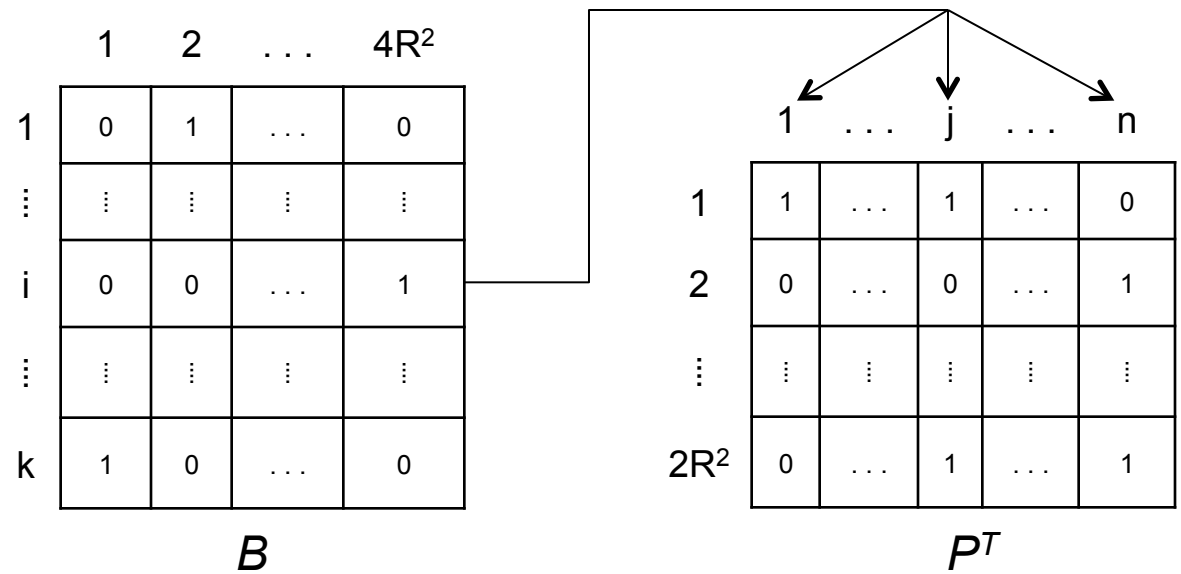
Local search: an efficient implementation

- A straightforward implementation of step 2 is:

```
for i ← 1 to k do
    for j ← 1 to n do
        if j ≠ i then
            // count the number of 1 bits in B[i] ⊕ P[j]
            for w ← 1 to 4R² do
                V[i,j] ← V[i,j]+(B[i,w] or P[j,w])
```

Matrix *V* computation

- But, considering the transpose of *P*



- The computation of *V* is very similar to the matrix multiplication (replacing the multiplication operator with a bitwise-or)

Local search: an efficient implementation

- Thus, the code for step 2 is:

```
for i ← 1 to k do
    for j ← 1 to n do
        if j ≠ i then
            // count the number of 1 bits in B[i] ⊕ P[j]
            for w ← 1 to 4R² do
                V[i,j] ← V[i,j]+(B[i,w] or P^T[w,j])
```

Local search: an efficient implementation

- The step 2 can be efficiently computed adapting a very fast GPU matrix multiplication algorithm.

- We adapted the algorithm developed (implemented) by Nvidia in 2013:
  - the multiplication operation was replaced with bitwise-or operation;
  - as the viewsheds are, usually, very sparse matrices, we included code to avoid loading and processing matrix blocks where all elements are 0;

Results

- Our algorithm *SiteGSM* was compared against two other versions (implementations): *Site*+ and *SiteGPU.*
- Both are also based on the greedy strategy and use local search, but
  - *Site*+ uses a sequential (CPU) implementation;
  - *SiteGPU* implements some operations using GPU but it uses only the GPU global memory and does not include dynamic programming.

Results

- The tests were executed on a computer with dual Intel Xeon E5-2687 3.1GHz, 128GiB of memory, GPU NVIDIA Tesla Kepler K20x with 2688 cores running Ubuntu 12.04 LTS.

- We used terrains with 1201 x 1201 points and 3601 x 3601 points (obtained from NASA STRM)

Results

Ter.	R	Ω	#Obs.	Processing Time (in sec.)				
				SiteGSM	*SiteGPU*		*Site+*	
1201 × 1201	100	75%	162	12	180	(15.0)	11010	(917.5)
		85%	299	33	545	(16.5)	∞	(-)
		95%	*	*	*	(-)	*	(-)
	200	75%	55	3	35	(11.7)	1304	(434.7)
		85%	97	6	104	(17.3)	4020	(670.0)
		95%	323	48	888	(18.5)	∞	(-)
	300	75%	34	2	19	(9.5)	479	(239.5)
		85%	62	4	70	(17.5)	1826	(456.5)
		95%	216	28	566	(20.2)	19408	(693.1)
3601 × 3601	200	75%	81	76	422	(5.6)	∞	(-)
		85%	97	110	708	(6.4)	∞	(-)
		95%	125	183	1341	(7.3)	∞	(-)
	300	75%	36	28	160	(5.7)	∞	(-)
		85%	42	41	276	(6.7)	∞	(-)
		95%	54	66	523	(7.9)	∞	(-)
	400	75%	20	14	81	(5.8)	14867	(1061.9)
		85%	24	19	124	(6.5)	22869	(1203.6)
		95%	30	30	270	(9.0)	∞	(-)

Results

- As an additional test, we compared the execution time of the local search using a conventional approach against our proposed strategy that includes:
  - dynamic programming
  - “matrix multiplication” using GPU

Results

Terrain	Number of Candidates	# Obs.	Time (in seconds) Compute B matrix Conv.	Compute B matrix DP	Compute V matrix CPU	Compute V matrix GPU	Total Conv.	Total Proposed	Speedup
1201 X 1201	500	16	0.1	0.1	17.4	0.1	17.6	0.9	20x
		32	1.3	0.1	98.7	0.5	100	1.6	63x
		64	9.2	0.3	351	1.5	360	3.4	106x
	1000	32	1.1	0.1	175	0.7	177	1.9	93x
		64	10.7	0.4	829	3.1	839	5.2	161x
		128	94	1.6	3363	12	3457	18	192x
		256	640	5.5	11129	39.1	11769	56.8	207x
3601 X 3601	500	16	0.4	0.1	52.3	0.4	53	2.6	20x
		32	2.6	0.1	183	0.9	186	3.6	52x
		64	18	0.6	635	2.8	654	6.9	95x
	1000	32	2.2	0.2	314	1.3	317	5.1	62x
		64	23.9	0.8	1689	6.2	1713	12.8	134x
		128	175	3	6083	21.7	6259	35.2	178x
		256	1375	12	24114	83.7	25489	126	202x
	2000	32	2.2	0.2	636	2.3	639	8.7	73x
		64	13.4	0.5	1880	6.7	1895	14.2	133x
		128	192	3.3	13381	46.9	13575	64	212x
		256	1320	11.5	46008	159	47329	203	234x

Conclusion

- We presented a very fast implementation of a method to site observers on terrains.
- This implementation is based on a greedy strategy combined with a local search where we used dynamic programming and GPU parallel implementation.
- This local search strategy can be used to improve other heuristics that solves other optimization problems.

Future work

- Develop parallel implementation using GPU to:
  - compute the viewshed of each observer;
  - replace the greedy strategy.

Thank you

Any questions or suggestions?

Acknowledgements

UFV

FAPEMIG

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

NSF
Grant IIS-1117277

Contacts:
marcus@ufv.br
salles@ufv.br