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
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](image1.png)  
![Digital Elevation Matrix](image2.png)

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 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 visibility

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

- 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 observers.
  - 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 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.

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 $V_1, V_2, \ldots, V_5$ and let $S=\{V_1, V_2, V_3\}$ be a partial solution. Thus, the neighbors of $S$ are:

- $S'_{1} = \{V_1, V_2, V_4\}$
- $S'_{2} = \{V_2, V_3, V_5\}$
- $S'_{3} = \{V_1, V_3, V_4\}$
- $S'_{4} = \{V_1, V_3, V_5\}$
- $S'_{5} = \{V_1, V_2, V_5\}$
- $S'_{6} = \{V_2, V_3, V_4\}$
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 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.

In each iteration, it is necessary to check all candidate observers to select the one that will most increase the joint visibility index.
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 method requires a lot of processing 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 $i=1,\ldots,k$ and $j=1,\ldots,n$ 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 $k \times n$ 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:

  \[
  \begin{align*}
  &\text{for } i \leftarrow 1 \text{ to } k \text{ do} \\
  &\quad \text{for } m \leftarrow 1 \text{ to } k \text{ do} \\
  &\quad\quad \text{if } m \neq i \text{ then} \\
  &\quad\quad\quad \text{// overlap } B[i] \text{ with } S[m] \\
  &\quad\quad\quad B[i] \leftarrow B[i] \oplus S[m]
  \end{align*}
  \]
Local search: an efficient implementation

A straightforward implementation of step 1 is:

\[
\text{for } i \leftarrow 1 \text{ to } k \text{ do}
\]
\[
\quad \text{for } m \leftarrow 1 \text{ to } k \text{ do}
\]
\[
\quad \quad \text{if } m \neq i \text{ then}
\]
\[
\quad \quad \quad \text{// overlap } B[i] \text{ with } S[m]
\]
\[
\quad \quad B[i] \leftarrow B[i] \oplus S[m]
\]

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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B[1] =</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[2] =</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[3] =</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[4] =</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[5] =</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GPU parallel siting algorithm
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:


The matrix with all $B$’s can be split in the following way.
Local search: an efficient implementation

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

|------|------|------|------|------|

= 

The diagram shows the computation of the matrix B as a combination of S matrices. The result is a matrix that combines the values from each S matrix in the correct order.
Local search: an efficient implementation

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

|-------|------|------|------|------|

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 $i = 2, \ldots k-1$,

\[ B_i = S_1 \oplus \cdots \oplus S_{i-1} \oplus S_{i+1} \oplus \cdots \oplus S_k \]
Local search: an efficient implementation

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

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

\[
L_i \oplus R_i
\]
Local search: an efficient implementation

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

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

\[ L_i \]

\[ R_i \]

\[ B_i = L_i \oplus R_i \]

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

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

\[ R_k = \Phi \text{ and } R_i = S_{i+1} \oplus R_{i+1} \text{ for } i=k-1, \ldots, 1 \]
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 $i=1,\ldots,k$ and $j=1,\ldots,n$, is the number of 1 bits in the overlapping of $B[i]$ with $P[j]$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>$4R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ B \]

\[ P \]

GPU parallel siting algorithm
Local search: an efficient implementation

- A straightforward implementation of step 2 is:

  for $i \leftarrow 1$ to $k$ do
    for $j \leftarrow 1$ to $n$ do
      if $j \neq i$ then
        // count the number of 1 bits in $B[i] \oplus P[j]$
        for $w \leftarrow 1$ to $4R^2$ do
          $V[i,j] \leftarrow V[i,j] + (B[i,w] \text{ or } P[j,w])$
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:

\[
\begin{align*}
\text{for } i &\leftarrow 1 \text{ to } k \text{ do} \\
&\quad \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
&\quad \quad \text{if } j \neq i \text{ then} \\
&\quad \quad \quad \text{// count the number of 1 bits in } B[i] \oplus P[j] \\
&\quad \quad \text{for } w \leftarrow 1 \text{ to } 4R^2 \text{ do} \\
&\quad \quad \quad V[i,j] \leftarrow V[i,j] + (B[i,w] \text{ or } P^T[w,j])
\end{align*}
\]
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

<table>
<thead>
<tr>
<th>Ter.</th>
<th>R</th>
<th>( \Omega )</th>
<th>#Obs.</th>
<th>( \text{Site}_{GSM} )</th>
<th>( \text{Site}_{GPU} )</th>
<th>( \text{Site}^+ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1201</td>
<td>100</td>
<td>75%</td>
<td>162</td>
<td>12</td>
<td>180 (15.0)</td>
<td>11010 (917.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85%</td>
<td>299</td>
<td>33</td>
<td>545 (16.5)</td>
<td>( \infty ) (-)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95%</td>
<td>*</td>
<td>*</td>
<td>* (-)</td>
<td>* (-)</td>
</tr>
<tr>
<td>1201</td>
<td>200</td>
<td>75%</td>
<td>55</td>
<td>3</td>
<td>35 (11.7)</td>
<td>1304 (434.7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85%</td>
<td>97</td>
<td>6</td>
<td>104 (17.3)</td>
<td>4020 (670.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95%</td>
<td>323</td>
<td>48</td>
<td>888 (18.5)</td>
<td>( \infty ) (-)</td>
</tr>
<tr>
<td>1201</td>
<td>300</td>
<td>75%</td>
<td>34</td>
<td>2</td>
<td>19 (9.5)</td>
<td>479 (239.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85%</td>
<td>62</td>
<td>4</td>
<td>70 (17.5)</td>
<td>1826 (456.5)</td>
</tr>
<tr>
<td>1201</td>
<td>400</td>
<td>75%</td>
<td>20</td>
<td>14</td>
<td>81 (5.8)</td>
<td>14867 (1061.9)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>85%</td>
<td>24</td>
<td>19</td>
<td>124 (6.5)</td>
<td>22869 (1203.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>95%</td>
<td>30</td>
<td>30</td>
<td>270 (9.0)</td>
<td>( \infty ) (-)</td>
</tr>
</tbody>
</table>
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

## Table: Performance Comparison of GPU Parallel Siting Algorithm

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Number of Candidates</th>
<th># Obs.</th>
<th>Compute B matrix</th>
<th>Compute V matrix</th>
<th>Total</th>
<th>Conv.</th>
<th>Proposed</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Conv.</td>
<td>DP</td>
<td>CPU</td>
<td>GPU</td>
<td>Conv.</td>
<td>Proposed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>0.1</td>
<td>0.1</td>
<td>17.4</td>
<td>0.1</td>
<td>17.6</td>
<td>0.9</td>
</tr>
<tr>
<td>1201</td>
<td>500</td>
<td>32</td>
<td>1.3</td>
<td>0.1</td>
<td>98.7</td>
<td>0.5</td>
<td>100</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>9.2</td>
<td>0.3</td>
<td>351</td>
<td>1.5</td>
<td>360</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>1.1</td>
<td>0.1</td>
<td>175</td>
<td>0.7</td>
<td>177</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>10.7</td>
<td>0.4</td>
<td>829</td>
<td>3.1</td>
<td>839</td>
<td>5.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>94</td>
<td>1.6</td>
<td>3363</td>
<td>12</td>
<td>3457</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>640</td>
<td>5.5</td>
<td>11129</td>
<td>39.1</td>
<td>11769</td>
<td>56.8</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>16</td>
<td>0.4</td>
<td>0.1</td>
<td>52.3</td>
<td>0.4</td>
<td>53</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>2.6</td>
<td>0.1</td>
<td>183</td>
<td>0.9</td>
<td>186</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>18</td>
<td>0.6</td>
<td>635</td>
<td>2.8</td>
<td>654</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>2.2</td>
<td>0.2</td>
<td>314</td>
<td>1.3</td>
<td>317</td>
<td>5.1</td>
</tr>
<tr>
<td>3601</td>
<td>500</td>
<td>64</td>
<td>23.9</td>
<td>0.8</td>
<td>1689</td>
<td>6.2</td>
<td>1713</td>
<td>12.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>175</td>
<td>3</td>
<td>6083</td>
<td>21.7</td>
<td>6259</td>
<td>35.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>1375</td>
<td>12</td>
<td>24114</td>
<td>83.7</td>
<td>25489</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>32</td>
<td>2.2</td>
<td>0.2</td>
<td>636</td>
<td>2.3</td>
<td>639</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>13.4</td>
<td>0.5</td>
<td>1880</td>
<td>6.7</td>
<td>1895</td>
<td>14.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>192</td>
<td>3.3</td>
<td>13381</td>
<td>46.9</td>
<td>13575</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256</td>
<td>1320</td>
<td>11.5</td>
<td>46008</td>
<td>159</td>
<td>47329</td>
<td>203</td>
</tr>
</tbody>
</table>

GPU parallel siting algorithm
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

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

Grant IIS-1117277

GPU parallel siting algorithm