TU Delft

Delft University of Technology

A computationally cheap trick to determine shadow in a voxel model

Gorte, B. G.H.; Zhou, K.; Van Der Sande, C. J.; Valk, C.

DOI
10.5194/isprs-annals-IV-4-67-2018

Publication date
2018

Document Version
Final published version

Published in
ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences

Citation (APA)
Gorte, B. G. H., Zhou, K., Van Der Sande, C. J., & Valk, C. (2018). A computationally cheap trick to determine shadow in a voxel model. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*(4), 67-71. https://doi.org/10.5194/isprs-annals-IV-4-67-2018

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

A COMPUTATIONALLY CHEAP TRICK TO DETERMINE SHADOW IN A VOXEL MODEL

B.G.H. Gorte [a,1], K. Zhou [b], C.J. van der Sande [c], C. Valk [c]

[a] University of New South Wales, Faculty of the Built Environment, Sydney Australia, b.gorte@unsw.edu.au
[b] Delft University of Technology, Dept. of Geoscience and Remote Sensing, the Netherlands, k.zhou-1@tudelft.nl
[c] NEO BV., Amersfoort, the Netherlands ({corne.vandersande, cornelis.valk}@neo.nl

Commission IV, WG IV/10

KEY WORDS: 3D City models, Voxels, Shadow, Quantitative Modelling, Simulation

ABSTRACT:

Representation of scenes on the Earth surface by using voxels is gaining attention because of its suitability for integrating heterogeneous data sources in simulations and quantitative models. Computation of shadows in such models is needed, for example, to obtain crop suitability of agricultural fields in the presence of trees and buildings, or to analyze urban heat island causes and effects. We present an efficient algorithm to compute which of the voxels in a dataset receive direct sunlight, given the solar azimuth and elevation angles. The algorithm can work with multiple (sparse and dense) voxel storage strategies.

1. INTRODUCTION

The interaction of sunlight with the Earth surface is the driver of many fundamental Earth processes. When looking at a level of detail where individual three-dimensional objects, such as buildings and trees, play a role, the interaction is heavily influenced by the question whether the surface at a particular position, and at a given moment in time, is either 'in the sun' or 'in the shadow', i.e. whether it receives direct sunlight (coming in a straight line from the sun) or only indirect light (coming for elsewhere in the blue sky, or from reflecting surfaces in the surrounding). Clearly, the answer depends on the presence of other objects on the path between the sun at that moment in time and the position of the surface under consideration. In the absence of cloud cover, such objects blocking the sunlight, would be buildings, hills, trees etc., which are part of the scene.

Modelling these effects is of interest when studying, for example, urban microclimate, field-based crop suitability and yield prediction, or solar panel placement. It assumes that a sufficiently detailed 3D model of the area under consideration is available. Because the sun is at any given possible position only once (or twice) per year, the model needs to be run many times to get reliable estimates of sun and shadow durations over a long period, like a growing season. Therefore, computational efficiency is quite important.

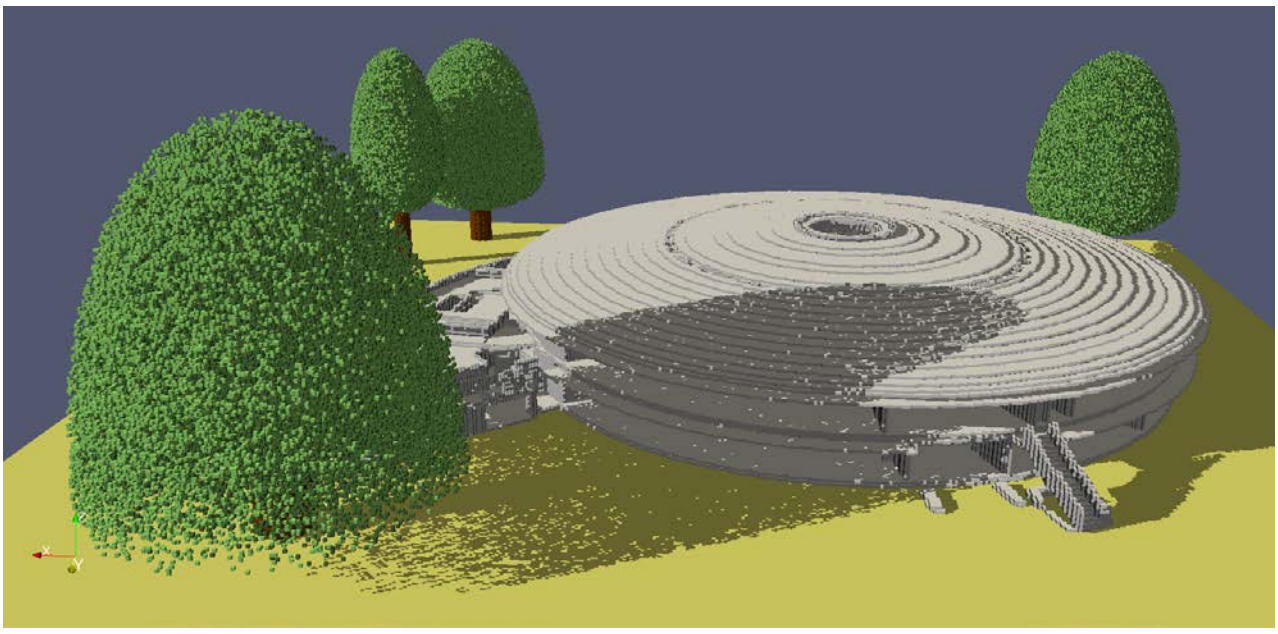


Fig. 1 Voxel model with shadows of the *Roundhouse* at UNSW Kensington campus, Sydney.

[1] Corresponding author

1.1 Voxels

The study in this paper was conducted within a larger effort to determine the usefulness of representing 3-dimensional geo-spatial information in large 3D grids, in which the raster elements are called *voxels*. The expectation is that this, in comparison to vector models, will lead to a more unifying approach on integrating many kinds of geo-information into computational simulation models, allowing to better exploit the spatial and temporal relationships between different objects, processes, themes, layers, etc, stored in a 3D GIS.

Here, we are considering a 3D model of an outdoor scene in sunny weather. Sunlight enters the scene as parallel rays from a direction that is defined by a pair of angles denoting sun azimuth and elevation. The goal of the paper is to compute which parts in the scene receive direct sunlight - the other parts being in the shadow. Fig. 1 shows an example.

The model describes a scene on the Earth surface, such as a part of a city. It is filled with different materials, forming objects, such as the terrain, buildings, bridges, trees etc. Air is present in the scene as well, and it is the only 'material' that is transparent to sunlight. Sunlight is blocked (i.e. either absorbed or reflected) by all other materials. Therefore, the objects cast shadows, which will prevent certain parts of surfaces of other objects from receiving direct sunlight - the effect on object surfaces of indirect light coming from the sky or being reflected by other objects is not the subject of this paper.

The scene has the extent of a rectangular block, subdivided into little cubes called voxels. The cubes are indexed by three-dimensional integer coordinates (x,y,z) ranging from (0,0,0) to $(N_x\text{-}1,N_y\text{-}1,N_z\text{-}1)$ and therefore the extent of the scene, measured in voxels, equals $N_x \times N_y \times N_z$. We will assume the axes are parallel to a relevant (perhaps local) terrestrial (X,Y,Z) coordinate system. The increment in X, Y or Z corresponding to an integer step in x, y or z is the resolution of the model - we could think of non-cubic voxels (with shapes likes bricks or pizza boxes), but it would not add much to the argument. The same holds for having a rotation between the two coordinate systems - except that it would require the azimuth (and/or elevation) angle to be rotated as well. Furthermore, we consider only shadows that are cast by objects inside the block. There is no shadow from objects surrounding the block. Finally, we will assume that the terrain surface is inside the block, and that all objects are lower than the top of the block. Therefore, the top layer(s) of the block consist/s of air, and the lowest ones of terrain.

Voxels are considered homogeneous. Their content is denoted as a scalar value, representing a single material or material class. We will assume the voxel size (the resolution) to be such that an urban scene is represented at a scale that reflects building details like balconies, chimneys etc., but perhaps not smaller details like ornaments or door handles - a typical voxel size would be in the range between 0.1 and 1.0 m. If the indoor environment is to be considered for shadow determination as well, we must allow the sunlight to enter through windows. These will have to be modelled as holes in the walls of the building. Furthermore, if the sunlight is supposed to be only partially blocked by tree crowns, these will have to be represented as mixtures of 'air' and 'leaf' voxels.

Also, the question about objects being in the sun or shadow will be answered per entire voxel. This will mainly concern voxels being illuminated from above – or not. Depending on the incidence angle, however, two more faces except the top face of a voxel are candidates for being sunlit. This will be addressed below.

The result of shadow modelling in computational models addresses, for example, the amount of sunlight that is captured by objects of interest over time. This will boil down to counting sunlit voxels per object (and per epoch). This in contrast to many existing methods, including those in game engines, which mainly aim at producing fancier visualisations, and have been known for a long time [Crow, 1977]. Such models can also be used to radiometrically correct satellite imagery prior to automatic change detection [van der Sande et. al. 2008], in order to prevent shadow differences being recorded as detected changes.

The straightforward method to obtain the result described above would be to have a ray of light entering a voxel at the top of the block at the desired angle, and trace it down through the ‘air’ voxels layer by layer until it hits an other-material (object) voxel. This is then marked as ‘sunlit’ and the ray stops. By repeating this process, starting from every voxel in the top layer of the block, one will end up with a collection of sunlit-marked voxels; the remaining ones are ‘shadow’. To make it possible that each bottom voxel in the block could eventually be reached by a ray originating from the top, it will be necessary to extend the block sideways with ‘air’ voxels, depending on the azimuth and elevation angles, before starting to trace rays.

This method is computationally expensive, because it has to traverse almost the entire voxel space sequentially, all the time performing computations to get from one voxel to the next, while intersecting the oblique ray with (perhaps several) voxels at each layer. We present a simple method to get the same result very little computational effort.

2. VOXEL SHADOW ALGORITHM

We regard the block of voxels describing a scene as a stack of N_z horizontal layers, numbered from 0 to N_z - 1, each having a size of $N_x \times N_y$ voxels.

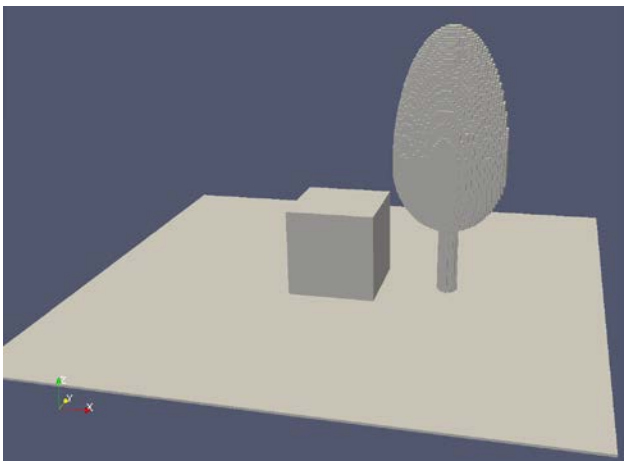


Fig. 2. Input model for voxel shadowing

2.1 The Basic Idea

The idea behind the algorithm is to shift each layer of the voxel model (Fig. 1) horizontally in the opposite direction of the sun azimuth angle, by an amount that depends on the sun elevation angle and the height of the layer. As a result (Fig. 3), points that

are geometrically located on the same sunbeam (if this beam were not blocked by the first non-air point it hits), will be exactly above one-another after the shifting took place.

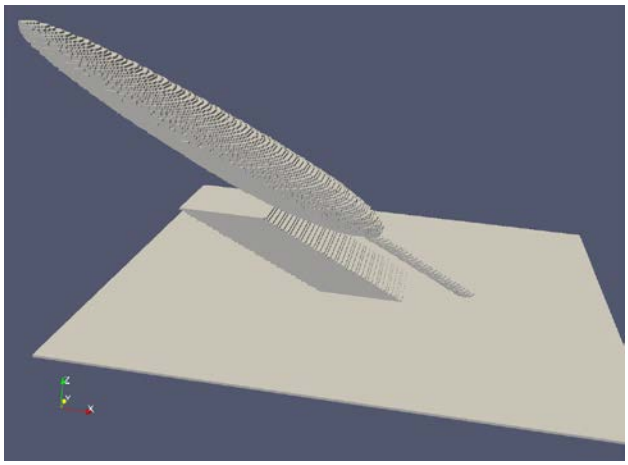


Fig. 3. Voxel model with shifted layers, according to sun angles.

In the next step of the process, the shifted voxel block is examined from top to bottom, one vertical column at a time, and the highest non-air voxel in each column is marked as ‘sunlit’. All voxels that are located lower in the column remain unmarked, meaning ‘shadow’ (fig. 4).

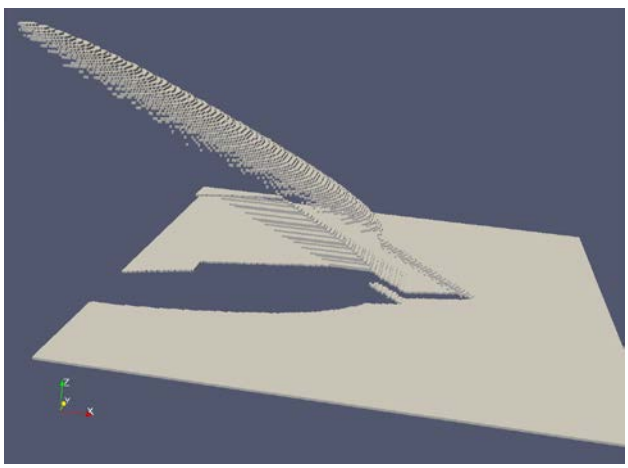


Fig. 4. Upper voxels in shifted model of Fig. 3

Finally, the sunlit voxels are shifted back to their original positions, layer by layer, by the amount that belongs to that layer (Fig. 5). They become the sunlit voxels in the original voxel space; the remaining voxels are in the shadow (Fig. 6).

2.2 Implementations in different voxel storage structures

We distinguish two storage structures for matrices, which we will name *dense* and *sparse* - both can be applied to represent voxel spaces in RAM or on disk. Very generally speaking, dense matrices are quicker during processing, but sparse ones require less memory. However, the ‘sparser’ a dataset really is, the faster its processing will tend to be, and, depending on the operation, there may exist a break-even point where ‘sparse’ gets quicker than ‘dense’.

Generally, computer memory is linear; data are stored in words (of e.g. 64 bits) at addressable locations, where the range of addresses is a consecutive linear list.

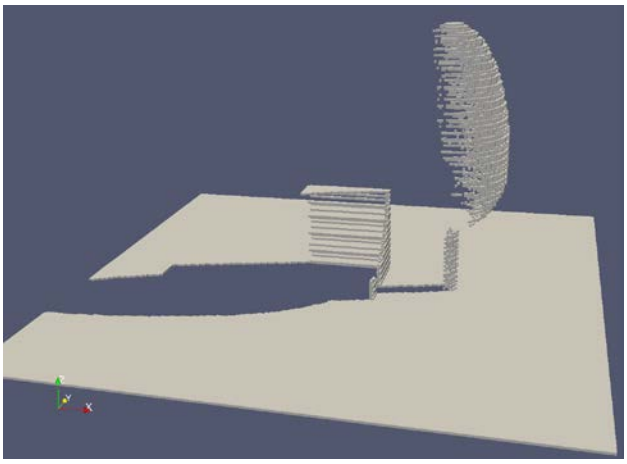


Fig. 5. Top voxels from Fig. 4 shifted back to their original positions.

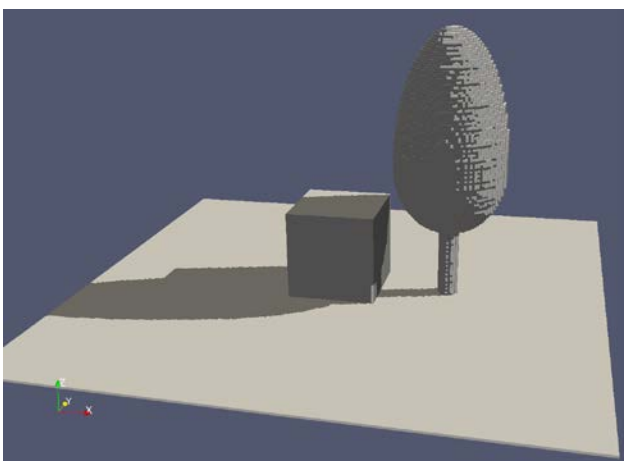


Fig. 6. The voxels of Fig. 5 are ‘sun’, the remaining voxels of the input model (Fig. 2) are ‘shadow’

A dense matrix occupies a consecutive piece of memory, in which only the values of the elements (the voxels) of the matrix are stored. The size of the block, which equals $N_x \times N_y \times N_z$., together with the data type of the voxel values (byte, floating point, etc.) determines the required amount of memory. In our method it matters in which order the voxels are stored. A block can be thought of as either a vertical stack of (horizontal) layers, or as a horizontal ‘field’ of (vertical) columns. In the first case, each layer occupies a consecutive piece of memory. In the second case, the vertical columns are stored as consecutive pieces. The choice is made during the declaration (or the creation) of the matrix in your programming language of choice – in a vast majority of languages, a formulation like M[N_x][N_y][N_z.] will yield a set of columns, whereas M[N_z,][N_x,][N_y.] gives a layer-by-layer representation. *Generalized* (or multi-dimensional) *transpose* is the name of the operation that changes between the two. In any case, the address of each voxel (w.r.t. the beginning of the block) can be easily computed based on its index.

The advantage of the proposed layer shift algorithm, as compared to the ‘straightforward’ approach, is firstly that the amount of shift is constant within each layer, and needs to be computed only once. But more importantly, performing those shifts means to move relatively large chunks of data around in memory, and this can be done quite quickly, as far as these are stored compactly, i.e. as layers. If this is not the case, it may be advantageous to perform a generalized transpose first. In the shifted dataset, a search for the highest non-air voxel is performed, which is again a very simple operation (which may benefit from a pillar-wise storage, however).

The sparse matrix representation, on the other hand, attempts to take advantage of the fact that usually a majority of the voxels in a scene will share a single value or belong to a single class. In outdoor, above-ground, scenes this value or class will be the one denoting ‘air’. In a sparse matrix, those voxels are not stored and do not occupy any space. At the downside, the positions of (other) voxels in the data structure cannot be easily computed on the basis of their position in the 3D space. Instead, the indices of the (non-air) voxels are stored explicitly. Therefore, the indices require space as well, which comes in addition to the space for the (non-air) values. Retrieving a voxel implies performing a search for the wanted index. Moreover, finding out that a voxel at a certain position is ‘air’ means to discover that it is not present in the data, which in a first approximation might involve checking the entire dataset. Fortunately, techniques exist, such as hashing and spatial indexing, to greatly speed up those searches, but on the other hand hash tables and indices require memory space as well.

The ‘straightforward’ shadow algorithm is expensive with sparse matrices, since tracing rays diagonally through the space requires continuous searching for the next (perhaps non-existing) voxel. Layer shifting, on the other hand, is extremely efficient. It has to be done for the non-air voxels only, coordinate by coordinate, where (x,y) have to be changed by an amount only depending on z – this amount can be taken from a lookup table. Next, the shifted non-air voxels have to be re-grouped based on their new (x,y) indices, and the one with the highest z has to be identified in each group. Now the efficiency depends heavily on the applied hashing (or indexing) technique, but again there are only non-air voxels involved. The top ones obtain a modified value (‘sun’) and are shifted back to their original positions, replacing the original values.

3. REFINEMENTS

3.1 Thin structures with low sun positions

The results of the above-described algorithm are correct, provided the sun elevation angle is larger than 45 degrees. At a smaller angle, it occurs that a layer a height z needs to be shifted more than one voxel further than the layer a height z-1. Structures of only one voxel thickness, in such a case, may start to show ‘air’ between subsequent layers – hopefully Fig. 7 illustrates the effect sufficiently clearly.



Fig. 7. Thin wall and low sun give a broken shadow.

The cause is shown in Fig. 8: Differences in shifts between subsequent layers are more than the wall thickness.

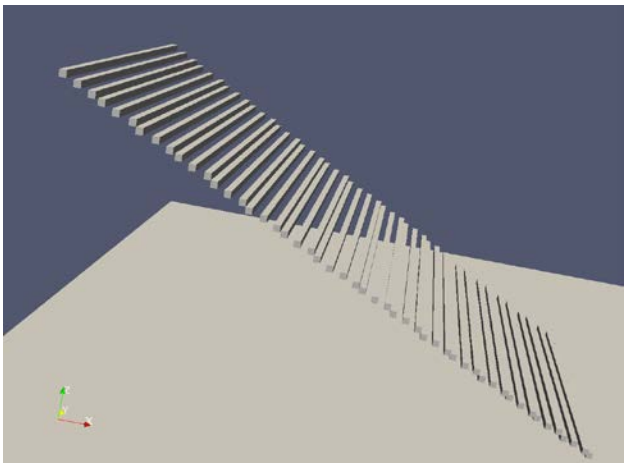


Fig. 8. Broken thin wall after shift

The solution is to perform the shifting bottom up, keep track of the shifts between layers, and fill the holes as the (might) occur (Fig. 9).

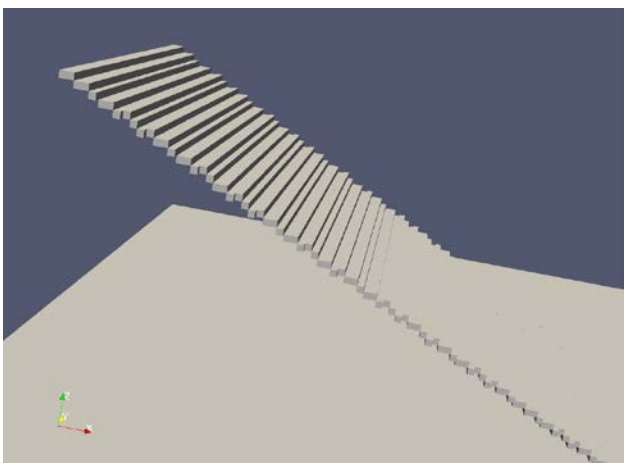


Fig. 9. Corrected shifted thin wall

After that, the entire space under the shifted wall will be ‘filled’ with uninterrupted shadow, and remain so after shifting back (Fig. 10).

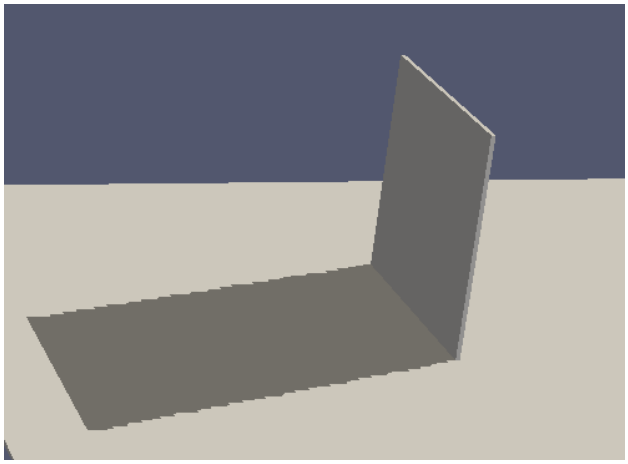


Fig 10. Corrected shadow (compared to Fig. 7) of thin wall at low sun position

3.2 Side views of sunlit walls

The proposed algorithm is only concerned with how sun is shining on objects from above. It gives correct results, for example, at the terrain and at the roofs of buildings. In addition, all other voxels in the shadow, which are part of a wall, for example, will never show up as ‘sunlit’. Walls that are sunlit, however, may get strangely striped (Fig 11).

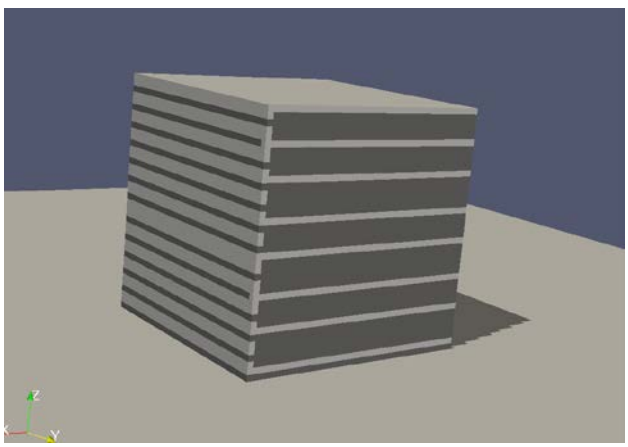


Fig. 11. Striped sunlit walls

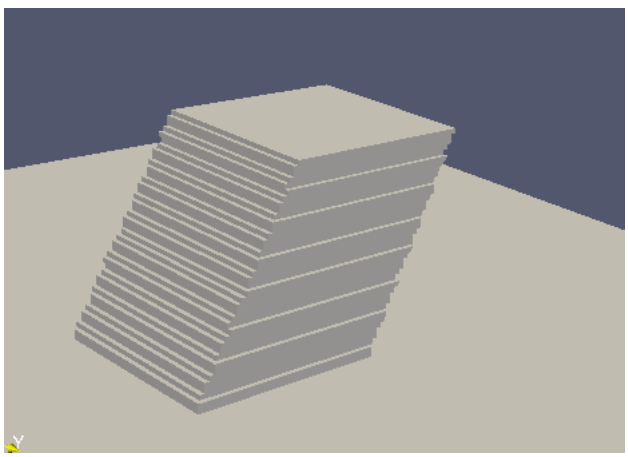


Fig. 12. Shifted block showing the cause of striped walls

The cause of this gets immediately clear when inspecting the shifted block (Fig 12). Indeed there are sunlit (top) voxels along the walls. As a solution, one might choose to allow only voxels that have ‘air’ above them n the original models as candidates for being ‘sunlit’. They can be selected with the same algorithms as the sunlit voxels in the shifted model. The selection can be made before or after shifting, and the result is shown in Fig. 12. The result visually more appealing, but not necessarily more useful from a modelling point of view, where one might quantize the interaction between sunlight and objects by counting sunlit voxels – then the result in Fig. 11 is actually not bad.

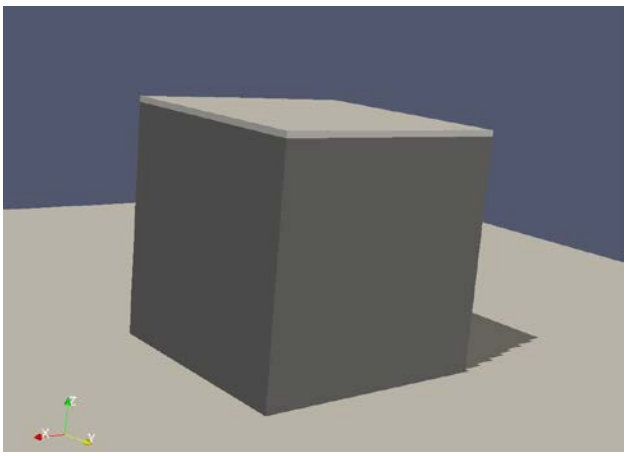


Fig. 13. Shadow model where only top voxels are sunlit

4. EVALUATION AND CONCLUSION

We have shown the feasibility of computing shadowed vs. sunlit voxels using a little algorithm that directly runs in the voxel domain. Its main attraction is simplicity. Apart from delivering a useful result, for which several applications have been identified, it shows the potential of voxel-based modelling of Earth processes.

REFERENCES

References

Crow, F.C., 1977. Shadow algorithms for computer graphics. *Computer Graphics*, 11(3), 242-8, (Proc. SIGGRAPH ‘77).

Van der Sande, C, Zanoni, M and Gorte, B, 2008, Improving 2D change detection by using available 3D data, *IAPRS* Vol. 37 b.7, Beijing 2008.

Blinn, James (1988), "Me and my (fake) shadow", *IEEE Computer Graphics and Applications*, January 1988.

Maren, Gert van and Jinwu Ma (2012) , 3D Analyst – Feature & Volumetric Analysis, *ESRI International User Conference*, San Diego, July 2012.