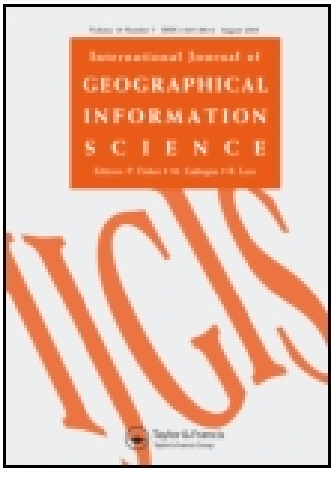

International Journal of Geographical Information Science

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tgis20

A visualization-oriented 3D method for efficient computation of urban solar radiation based on 3D-2D surface mapping

Jianming Liang[ab], Jianhua Gong[ac], Wenhang Li[ac] & Abdoul Nasser Ibrahim[ac]

[a] State Key Laboratory of Remote Sensing Science, Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing 100101, China

[b] University of Chinese Academy of Sciences, Beijing 100049, China

[c] Zhejiang-CAS Application Center for Geoinformatics, Zhejiang 314100, China

Published online: 31 Jan 2014.

To cite this article: Jianming Liang, Jianhua Gong, Wenhang Li & Abdoul Nasser Ibrahim (2014) A visualization-oriented 3D method for efficient computation of urban solar radiation based on 3D-2D surface mapping, International Journal of Geographical Information Science, 28:4, 780-798, DOI: 10.1080/13658816.2014.880168

To link to this article: http://dx.doi.org/10.1080/13658816.2014.880168

International Journal of Geographical Information Science, 2014
Vol. 28, No. 4, 780–798, http://dx.doi.org/10.1080/13658816.2014.880168

A visualization-oriented 3D method for efficient computation of urban solar radiation based on 3D–2D surface mapping

Jianming Liang[a,b], Jianhua Gong[a,c]*, Wenhang Li[a,c] and Abdoul Nasser Ibrahim[a,c]

[a]*State Key Laboratory of Remote Sensing Science, Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing 100101, China;* [b]*University of Chinese Academy of Sciences, Beijing 100049, China;* [c]*Zhejiang-CAS Application Center for Geoinformatics, Zhejiang 314100, China*

(*Received 19 November 2013; accepted 1 January 2014*)

The temporal and spatial distribution of solar energy in urban areas is highly variable because of the complex building structures present. Traditional GIS-based solar radiation models rely on two-dimensional (2D) digital elevation models to calculate insolation, without considering building facades and complicated three-dimensional (3D) shading effects. Inspired by the 'texture baking' technique used in computer graphics, we propose a full 3D method for computing and visualizing urban solar radiation based on image-space data representation. First, a surface mapping approach is employed to project each 3D triangular mesh onto a 2D raster surface whose cell size determines the calculation accuracy. Second, the positions and surface normal vectors of each 3D triangular mesh are rasterized onto the associated 2D raster using barycentric interpolation techniques. An efficient compute unified device architecture -accelerated shadow-casting algorithm is presented to accurately capture shading effects for large-scale 3D urban models. Solar radiation is calculated for each raster cell based on the input raster layers containing such information as slope, aspect, and shadow masks. Finally, a resulting insolation raster layer is produced for each triangular mesh and is represented as an RGB texture map using a color ramp. Because a virtual city can be composed of tens of thousands of triangular meshes and texture maps, a texture atlas technique is presented to merge thousands of small images into a single large image to batch draw calls and thereby efficiently render a large number of textured meshes on the graphics processing unit.

Keywords: solar radiation model; 3D triangular mesh; surface mapping; 2D raster

1. Introduction

Solar radiation models have been implemented in GIS frameworks to facilitate integrated spatial analysis and mapping. GIS-based solar radiation models provide rapid, cost-effective, and accurate estimations of insolation over large geographic areas and take into consideration surface inclination, aspect, and shadowing effects (Hofierka and Suri 2002). The solar analyst (SA) module of ESRI ArcGIS (Fu and Rich 2000) and the r.sun module of GRASS GIS (Hofierka and Suri 2002) are two of the most widely used tools for insolation modeling. They have been sufficiently tested and shown to be capable of accurately capturing both temporal and spatial variability. In GIS-based solar models, two-dimensional (2D) digital elevation models or digital surface models (DSM) are typically used to calculate shading effects and surface orientation, which are key factors

*Corresponding author. Email: jhgong@irsa.ac.cn

in determining local variability (Dubayah and Rich 1995, Kumar *et al*. 1997). Such models have been used to study solar energy distribution in various application contexts (Kryza *et al*. 2010, Nguyen and Pearce 2010).

A good solar radiation model should be able to account for four groups of factors, that is, the Sun–Earth position, topography, atmospheric characteristics, and overcast conditions, with the first two groups treated as the Sun–Earth geometric modeling in the GIS framework (Liu *et al*. 2012). Shading may result in significant differences in the received solar energy because of blocking of direct sunbeams or diffuse light (Liu *et al*. 2012).

LiDAR-based data acquisition technology has enabled the rapid reconstruction of DSM, which can be fit directly into existing 2D models to assess building insolation (Agugiaro *et al*. 2011, Niko and Borut 2013). However, such applications do not take into account building facades and complicated shadowing effects, which could have major implications on an urban scale.

Virtual cities can be reconstructed from LiDAR point clouds, extruded from 2D building footprints or manually constructed by artists. Improvements in data acquisition capacities and computational technologies have facilitated the widespread application of two-dimensional (3D) virtual cities for geospatial analyses (Yasumoto *et al*. 2012, Coutu *et al*. 2013), which require new GIS methods to study urban-scale solar energy patterns, considering both building roofs and facades. The combination of DSM and vectorial building shapes provides a better understanding of the solar energy distribution on vertical surfaces through slicing of a building into several stories for separate calculations (Carneiro *et al*. 2008). However, the insolation distribution on facades is not accurately captured because only a limited number of slices are created, due to the limits on computational and storage capacities. Ray-tracing packages, for example, Radiance and Heliodon, employ physically based light propagation models to compute solar radiation on a microscopic scale, as demonstrated in several previous studies (Mardaljevic and Rylatt 2003, Compagnon 2004, Merino *et al*. 2010). Nevertheless, ray-tracing techniques are computationally too costly to be utilized for large-scale urban models, and such micro-scale calculations may lead to many uncertainties that cannot be addressed in a GIS context.

Recently, a combined vector–voxel 3D solar radiation model (v.sun) was developed in the framework of GRASS GIS (Hofierka and Zlocha 2012). With the v.sun model, 3D vector objects are all segmented into smaller polygonal elements using a voxel-intersecting rule; thus, the accuracy of the calculation depends on the voxel resolution. The v.sun model has been shown to provide acceptable solutions to the problem of computing solar radiation in complex urban areas using 3D geometric models. Because voxel-based real-time rendering is still under study (Laine and Karras 2010), only triangle rasterization is universally supported on the graphics processing unit (GPU) rendering architecture. According to the v.sun model, the number of segmented polygons increases with the voxel resolution; thus, interactive visualization for large-scale 3D virtual cities requiring high-resolution solar energy results may be less efficient because a large number of triangle primitives need to be rendered on the GPU. Nevertheless, spatial resolution is extremely important in the study of urban-scale solar energy potential because coarse-scale calculations may significantly underestimate or completely miss shading losses because of shadowing effects (Nguyen and Pearce 2012). Therefore, the feasibility of developing a visualization-oriented 3D solar model for urban buildings warrants further study.

Inspired by the v.sun model (Hofierka and Zlocha 2012), we attempt to develop a new 3D method to compute solar radiation for large-scale urban models, taking into account both building roofs and facades and achieving efficient computation, storage, and interactive visualization.

2. Data representation for 3D urban models

Urban features, such as buildings, are represented by 3D vector objects defined by a set of 3D surfaces (polygons, facets); thus, it can be quite straightforward to calculate the incoming solar radiation for each of these 3D polygons (Hofierka and Zlocha 2012). In comparison to voxels, triangular meshes provide far more compact storage for geometric information and are natively supported by the current GPU for real-time rendering (Laine and Karras 2010). However, polygons are not ideal for representing highly detailed surface properties because of the GPU's inefficiency in processing geometric complexity. In practice, triangular meshes are combined with 2D texture maps to simultaneously represent 3D geometric information and surface properties. For example, many 3D urban models are richly textured with surface property information obtained from photography or aerial imaging to make them more realistic (Tsai and Lin 2007). Voxels and 3D triangular meshes are two typical data models used to represent 3D geometric shapes and surface properties. They are characterized and compared in Table1 and Figure 1.

The comparisons made in Table 1 suggest that triangular meshes are more convenient for 3D geometric data representation in terms of GPU-based visualization under the present computational architecture. The surface properties of triangular meshes can also be compactly represented by 2D raster images (texture maps) to facilitate data storage, query, retrieval, and geovisualization (Lorenz and Döllner 2010). However, two types of

Table 1. Comparisons of voxels and triangular meshes for 3D model representation.

	Voxels	Triangular meshes
Representation	Unified geometry and surface properties	Mesh-based geometry and texture-based surface properties
Element	Voxel	Triangle and texel
Storage	Efficient for surface properties but not compact for geometric data because each voxel needs specific information	Efficient for surface properties with texture mapping and compact for geometric data because a facet can be described by a few vertices
Spatial organization	Regularly structured with natural spatial indexing, thus enabling rapid shadow-casting calculation	Irregularly structured and requiring specific spatial indexing; thus, a specific shadow-casting algorithm is needed
Visualization	Not supported by the conventional GPU rendering architecture	Natively supported by the GPU rendering architecture for triangle rasterization and texture mapping

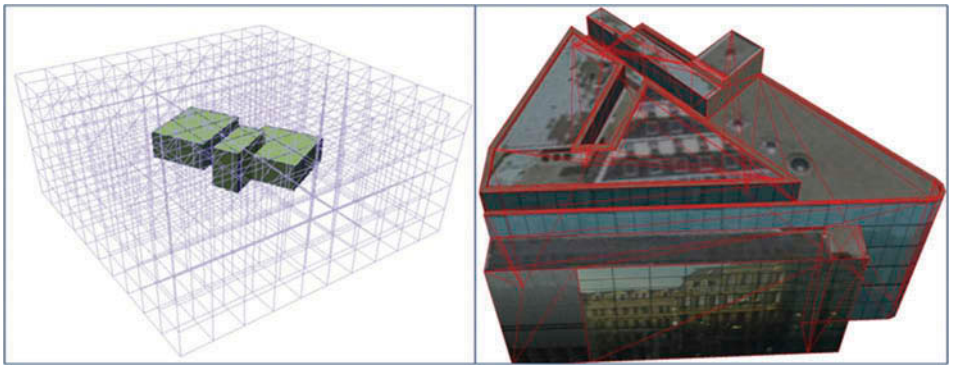

Figure 1. Comparisons of voxels and triangular meshes for 3D model representation.

difficulties must be addressed to implement a GIS-based solar model with such a data representation:

(1) GIS-based models necessitate the discretization of computational domains into regularly divided elements, for example, raster cells. Because triangles do not conveniently fit into such frameworks, certain data discretization techniques must be used to transform the triangular meshes into regularly partitioned elements.
(2) Shading calculations for 3D urban models are computationally intensive. The conventional triangle-based shadow-casting algorithm is prohibitively inefficient for such applications.

3. Test data

To demonstrate the applicability of the proposed method for 3D urban models, we have prepared a large-scale 3D virtual city (Figure 2) covering part of Boston. Boston is located at approximately 42°21′28″N and 71°03′42″W and is the largest city of the US State of Massachusetts.

The data set is a collection of 22,185 polygon shapes that describe building footprints and the number of stories, downloaded from the website of the open-source project osgEarth. This data set serves as the foundation for creating 3D urban models roughly corresponding to CityGML LoD1. Assuming a height of 3.5 m for a single story, each building is extruded and tessellated using a pair of 3D triangular meshes, that is, the roof and the walls. Consequently, the virtual scene is composed of 44,370 3D triangular meshes with 22,185 walls and 22,185 roofs. The total surface area for the building roofs and walls amounts to 31.39 million m^2, as determined by aggregating all the triangle primitives. The 3D scene is oriented so that the true north is aligned with the positive direction of the model-space Y axis (0, 1, 0) (Figure 2).

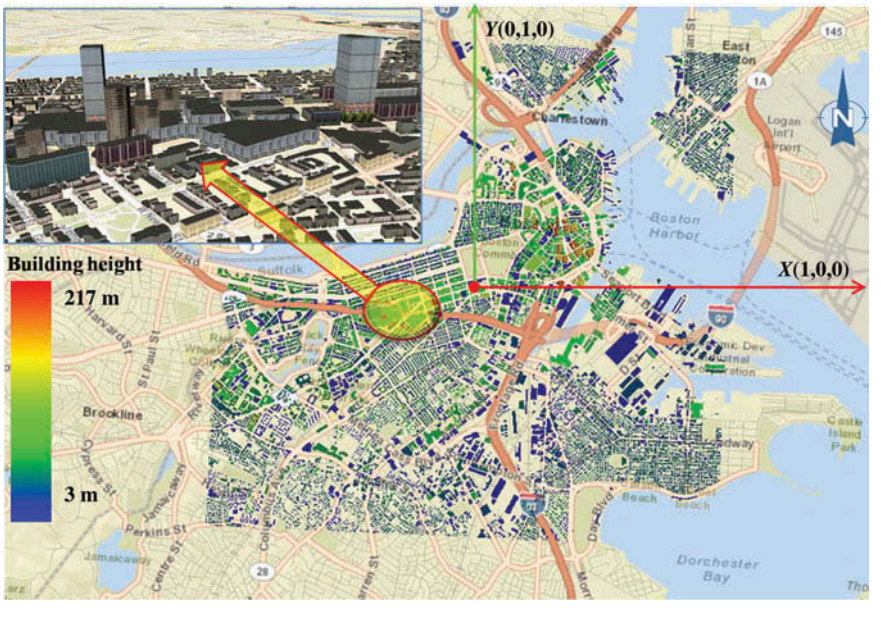

Figure 2. Boston urban models.

4. Mesh-raster-based 3D solar radiation model

We propose herein a mesh-raster-based method to compute and visualize urban insolation using 3D geometric models. The method has the following features:

(1) Incorporation of building facades in the calculation.
(2) Compact storage of calculation results.
(3) A uniform metric for calculation accuracy based on raster cell resolution.
(4) Accurate capture of shading-induced micro-scale variability.
(5) Efficient GPU-based interactive visualization.

Our approach exploits the 'texture baking' technique used in computer graphics. The basic assumption is that a 3D triangular mesh can be discretized into a set of 2D raster cells with certain 3D–2D surface mapping techniques, as shown in Figure 3.

The r.sun model is employed as the underlying GIS model to calculate the various components of the global solar radiation. Although multiple GIS solar models exist, the r.sun model is freely available with full source codes and has been tested in many studies with satisfactory results (Hofierka and Kanuk 2009, Nguyen and Pearce 2010, Agugiaro *et al*. 2011). Based on 2D raster representations of 3D surfaces (Figure 3), cell-by-cell calculations can be performed to extract the necessary input variables, for example, aspect, slope, and shadow mask, which are required by the r.sun model to determine the global solar radiation. The insolation results are produced as raster layers that can then be represented as RGB texture maps using a color ramp. Each texture map is bound to the corresponding triangular mesh and visualized using GPU-based real-time rendering techniques.

Shading is one of the most important factors in local spatial-temporal variability for areas of high geometric complexity. A ray-casting approach is used to calculate the shadowing effect for each raster cell by shooting a solar ray to perform ray-triangle intersection tests. The brute-force shadow-casting algorithm requires that a ray-triangle intersection test be performed with every triangle in the scene. Such a naïve implementation will certainly impose an overwhelming computational burden. Consequently, we seek to improve the efficiency of the brute-force shadow-casting algorithm by the following means:

(1) Employing suitable culling techniques to avoid as many ray-triangle intersection tests as possible. Such techniques should take into account the geographic characteristics of urban buildings, such as the shadow length.
(2) Leveraging the GPU to parallelize the raster-based calculations. Because an independent solar ray needs to be cast for each raster cell, parallelization can be expected to effectively accelerate such processes.

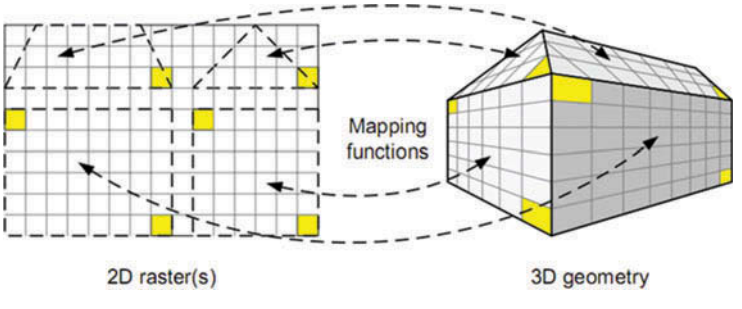

Figure 3. 3D–2D surface mapping (Lorenz and Döllner 2010).

4.1. Computation and visualization pipeline

In general, the computation and visualization framework consists of six major stages: 3D–2D surface mapping, geometry rasterization, model input preparation, insolation calculation, symbolization, and visualization (Figure 4). For each triangular mesh, the following actions are performed:

(1) A unique 2D raster collection object is created using surface mapping techniques. The raster collection can contain various attributes of the triangular mesh, for example, the normal vector, aspect, and shadow mask.
(2) The positions and normal vectors are rasterized and stored in the related raster collection as separate layers.
(3) The slopes, aspects, and shadow masks are calculated and stored as raster layers for model input.
(4) The input layers are fetched from the raster collection object and forwarded to the incorporated r.sun model to obtain the insolation values as a raster layer.
(5) The output insolation layer is symbolized using a color ramp to generate an RGB-colored texture map showing the insolation distribution on the 3D surface.
(6) The texture map is bound to the triangle vertices via texture coordinates for interactive visualization using the GPU.

4.2. The underlying GIS solar radiation model

The r.sun model in GRASS GIS was developed on the basis of the European Solar Radiation Atlas (ESRA) (Rigollier *et al.* 2000). To derive the global radiation, three components are taken into account in the model (Hofierka and Suri 2002), namely, the beam radiation, the diffuse radiation, and the reflective radiation. The beam radiation is the portion of the solar radiation that penetrates the atmosphere and passes directly to the

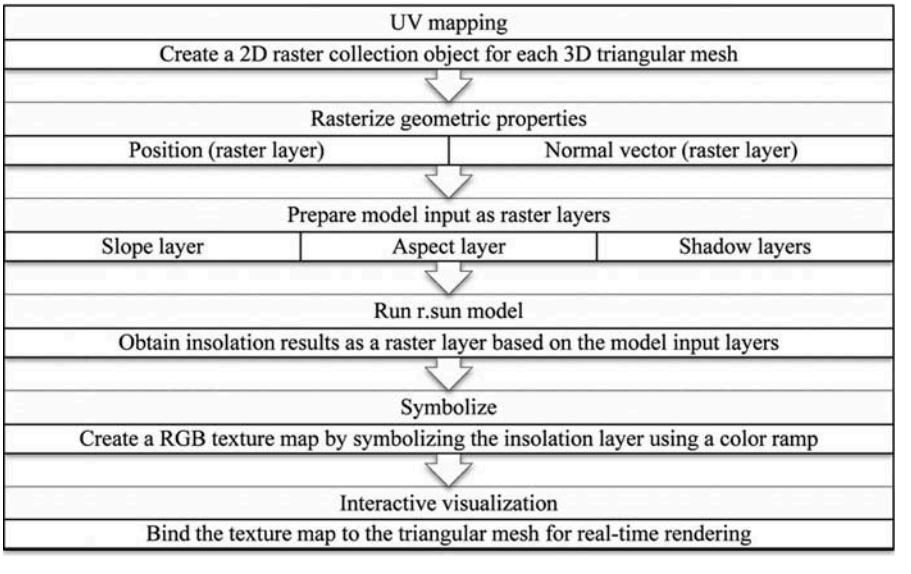

Figure 4. Computational and visualization pipeline.

receiving surface and is the primary source of the received solar energy. The diffuse radiation is the portion that is scattered from the atmosphere toward the receiving surface and is the second most important component. In the r.sun model, the reflective component is the portion that is reflected back from the ground and is the least significant component. In urban environments, multiple reflections between building surfaces also contribute to the reflective component. However, because of the lack of data on building surface reflectance, we do not consider multiple reflections in the calculations of the reflective component.

Information on the following variables are required by the r.sun model, and some of this information has to be derived at the model input preparation stage:

(1) Geographic coordinates (longitude, latitude, and elevation). Because a city usually covers a relatively small geographic area, we designated a shared geographic coordinate for calculation.
(2) Slope and aspect. The surface gradient is already implicitly contained in the normal vector for 3D triangular planes. Considering the model-space orientation of the virtual city (Figure 2), the slope of a triangular plane is defined by the following formula:

$$\text{slope} = \arccos[\text{dot}(V_n(X, Y, Z), V(0, 0, 1))] \tag{1}$$

where V_n refers to the normal vector, dot is the dot product. In GRASS GIS, aspect represents the number of degrees counterclockwise from east; thus, it can be calculated with the following formula:

$$\text{aspect} = \arccos[\text{dot}(V_n(X, Y), V(1, 0))] \tag{2}$$

(3) Shadow mask. For each time step, a Boolean shadow mask has to be calculated for each raster cell. The r.sun solar model describes the solar position by a zenith angle and an azimuth angle using the horizontal coordinate system. However, the urban models adopt a 3D Cartesian coordinate system that describe a 3D position or vector using a triple set (*X*, *Y*, *Z*). Thus, a 3D Cartesian vector needs to be derived from the zenith and azimuth angle to construct the solar vector. Because the true north of the 3D model scenes correspond to the *Y*-axis (0,1,0) (Figure 2), the model-space solar vector can be obtained by rotating the 3D vector (0,1,0) about the vector (0,0,–1) by an amount equal to the azimuth angle and then rotating the resulting vector about the vector (1,0,0) by an amount equal to the zenith angle.

4.3. 3D–2D surface mapping

A 2D raster herein takes the form of a two-dimensional image. The process of projecting a 3D polygon mesh onto a 2D image is known as UV mapping or UV unwrapping. UV mapping is used in computer-aided design (CAD) to unwrap 3D polygon meshes onto 2D texture maps, which is an essential step in 'texture baking'. The least squares conformal mapping (LSCM) method (Levy *et al*. 2002) is one of the classic UV mapping methods and has been incorporated into many 3D modeling software packages for 'baking textures'. Software toolkits such as Blender, 3ds Max, and Maya require many user interactions to unwrap even a single triangular mesh. To automatically process thousands of triangular meshes, three approaches can be considered:

(1) Incorporating the open-source library OpenNL that contains an LSCM implementation.
(2) Writing a script in Blender, 3ds Max, or Maya to batch UV unwrapping.
(3) Developing a case-specific UV unwrapping algorithm that accommodates only certain particular geometric shapes.

In this study, we developed a simple UV unwrapping algorithm to maximize texture space usage under the assumption that a building can be represented by two triangular meshes describing its facades and its roof, respectively. The algorithm projects the facades onto a vertically aligned plane (perpendicular to the ground) and projects the roof onto a horizontally aligned plane (parallel to the ground) according to the following steps (Figure 5):

(1) Determine the spatial extent (width and height in meters) of the 2D raster. For the roof, the extent can be the axis-aligned geographic bounding box or an arbitrarily aligned minimum bounding box. For the walls, the width is the circumference of the original 2D building footprint, and the height is the extrusion height.
(2) Determine a reasonable cell resolution to balance accuracy with efficiency.
(3) Create two separate raster surfaces for the roofs and the walls based on the specified spatial extent and cell resolution.

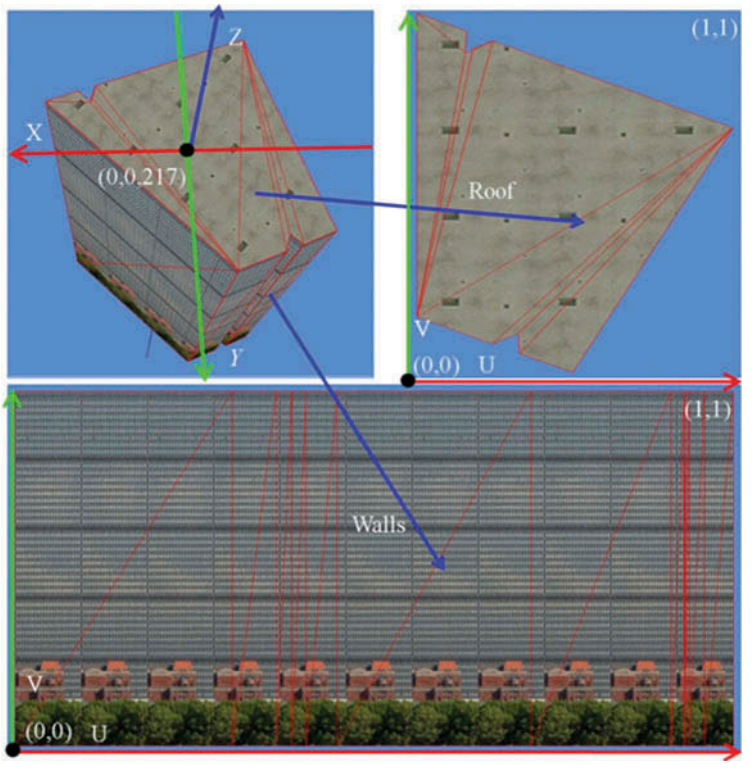

Figure 5. Transformation from a 3D model to an image space.

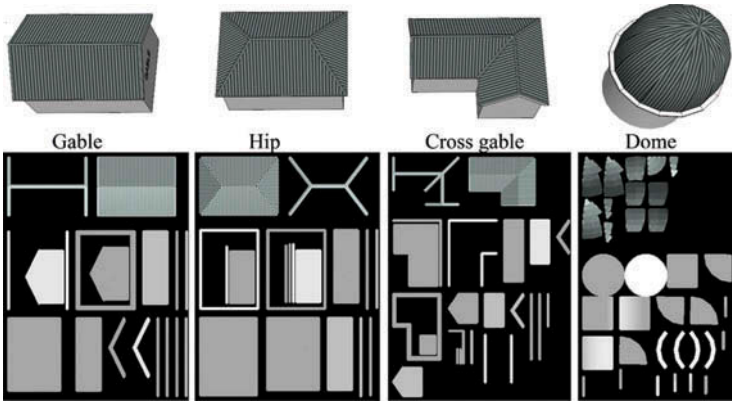

Figure 6. UV mapping for building models with various types of roofs.

(4) Generate texture coordinates for the triangle vertices of the roof and the walls. First, the four corners of the raster extent are assigned the texture coordinates (0,0), (0,1), (1,0), and (1,1), respectively. The texture coordinates for each triangle vertex are then calculated by linear interpolation using the corner points.

The UV mapping algorithm, presented in this study, has many limitations and does not consider arbitrary 3D triangular meshes. For example, a nearly vertical roof facet will be poorly represented in the 2D raster because a roof is treated as a 2.5D mesh. However, we illustrate in Figure 6 how the buildings with typical roofs can be unwrapped using a CAD software that implements a generalized UV mapping algorithm.

4.4. *Geometry rasterization*

Geometry rasterization is the process of discretizing a 3D triangular mesh into a set of raster cells (Figure 3) based on predetermined per-vertex texture coordinates. At this stage, each triangle vertex of the 3D triangular meshes is assigned a unique texture coordinate on the associated 2D raster surface. A triangle is geometrically described by three vertex positions and a normal vector. The barycentric coordinate system (Bradley 2007) can be set up to obtain the weights to linearly interpolate the geometric attributes of a raster cell using the three triangle vertices (Figure 7).

The left half of Figure 7 shows that the given triangle is located on the 3D triangular mesh by its three vertex positions, $P_1(X,Y,Z)$, $P_2(X,Y,Z)$, and $P_3(X,Y,Z)$. The right half of Figure 7 shows that the triangle's 2D counterpart on the raster surface is represented by the three texture coordinates, $T_1(U,V)$, $T_2(U,V)$ and $T_3(U,V)$, which are used to construct the barycentric coordinate triple $Ba(X,Y,Z)$(Bradley 2007). Given a raster cell within the triangle with the texture coordinate $T(U,V)$, the position and normal vector can be interpolated using the following formula:

$$\begin{cases} P(X,Y,Z) = Ba(X) \times P_1 + Ba(Y) \times P_2 + Ba(Z) \times P_3 & (3) \\ N(X,Y,Z) = Ba(X) \times N_1 + Ba(Y) \times N_2 + Ba(Z) \times N_3 & (4) \end{cases}$$

where $P(X,Y,Z)$ and $N(X,Y,Z)$ refer to the interpolated position and normal vector of the raster cell, respectively.

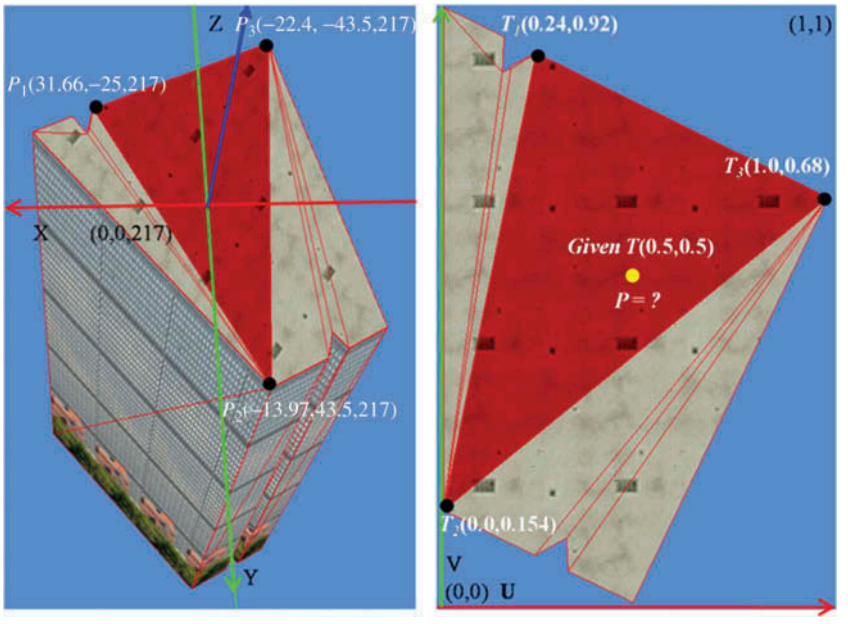

Figure 7. Triangle rasterization based on weighted interpolation.

4.5. *Efficient shadow-casting algorithm*

The basic shadow-casting algorithm is simple: a ray is constructed from the solar direction to find out whether the path from the sun to the destination is obstructed or not; to exclude the possibility of the ray being occluded, we have to traverse all the triangles in the scene to perform intersection tests. Because of the sheer volume of the geometric data in a 3D virtual city, brute-force shadow-casting can be an extremely computationally intensive and time-consuming process. For example, if a 3D virtual city consists of 1 million triangles, 1 million ray-triangle intersection tests need to be performed to determine the shadowing effect for even a single raster cell in the worst-case scenario.

We present an efficient shadow-casting algorithm by employing two strategies: first, we reduce the number of triangles that need to be traversed for intersection tests, and second, we use the GPU to parallelize the shadow-casting calculation on a cell-by-cell basis. The shadow-casting algorithm is implemented as follows:

(1) Bounding box calculation. As a preprocessing step, an axis-aligned 3D bounding box is calculated for each triangular mesh of the virtual city.
(2) Shadow-radius-based culling. Because the horizontal scale of a city is significantly greater than its vertical scale, we assume that the shadow projected by most shadow-casters in the scene may not be sufficiently long to cover the targeted shadow-receiving triangular mesh. Assuming that the highest point of a shadow-caster is defined by a 3D coordinate $P_h(X, Y, Z)$ and that the solar vector is given by a normalized 3D vector $V_{light}(X, Y, Z)$, the maximum shadow length L_{shadow} can be estimated using the following formula (Figure 8):

$$L_{shadow} = P_h(Z)/\arcsin\left(V_{light}(Z)\right) \tag{5}$$

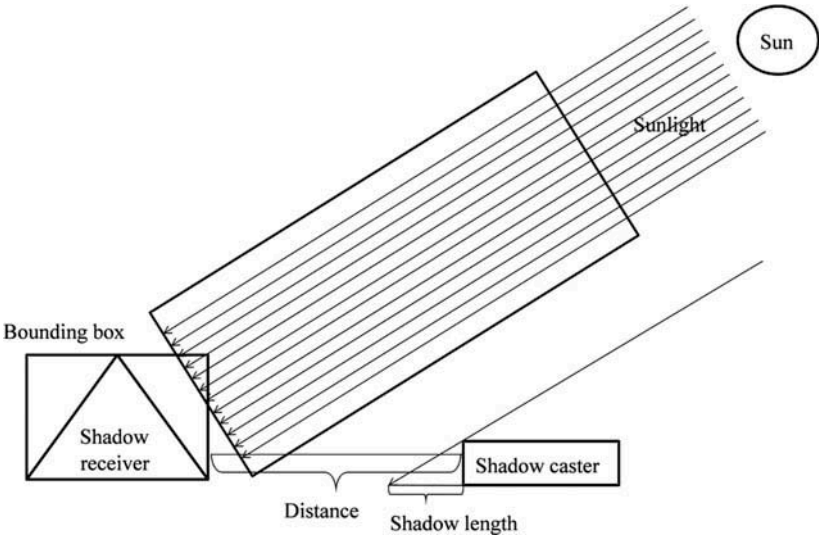

Figure 8. Shadow-radius-based culling.

(3) View-frustum-based culling. For a targeted shadow-receiving triangular mesh, an orthographic view frustum is set up based on the solar vector and the bounding box of the mesh. The bounding box of each triangular mesh in the scene is transformed into the solar view space for the frustum-box intersection test. Only the intersected meshes are selected from the scene for accurate ray-triangle intersection testing in a further step (Figure 9).
(4) Ray-triangle intersection. A ray is constructed starting from the cell position and extending in the direction of the solar vector to perform an accurate ray-triangle intersection test with the remaining triangle meshes. The shadow mask for a raster cell is marked true if the ray actually hits any triangle.

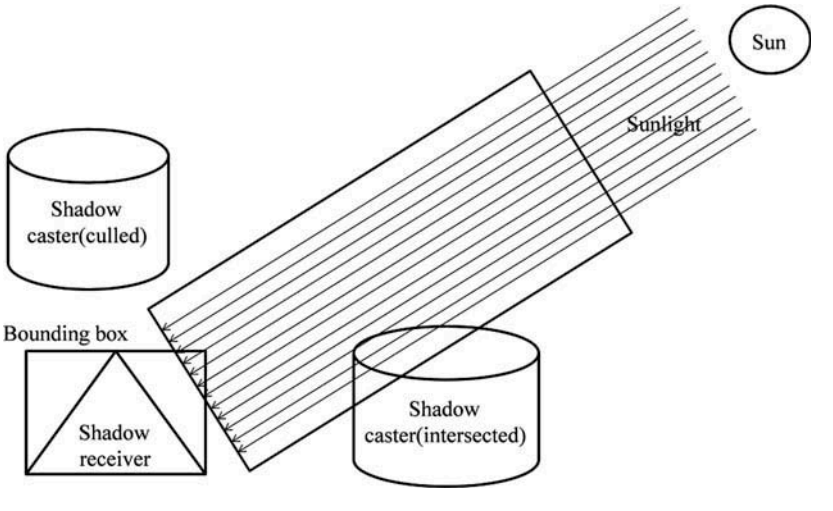

Figure 9. View-frustrum-based culling.

Table 2. Performance of the culling techniques.

Azimuth	Zenith	Number of meshes remaining (out of 44,370)	Number of triangles remaining (out of 818,377)
180	20	20	441
198	20	20	441
216	20	21	460
234	20	20	447
252	20	20	444
270	20	20	444
288	20	20	450
306	20	20	449
324	20	21	460
342	20	20	447
360	20	20	446

An experiment was performed using the Boston scene to determine how the culling techniques can contribute to a reduction in the number of intersection tests. Initially, we chose 11 azimuth angles ranging from 180° to 360° in increments of 18° and a zenith angle of 45°. The combination of the 11 azimuth angles and the zenith angle created 11 instances of model-space solar vectors. One in every 10 triangular meshes in the scene was selected to act as a shadow-receiver. The number of remaining triangles for each solar vector after culling was averaged over all the shadow-receivers. The results are presented in Table 2.

The statistics shown in Table 2 indicate that the majority of the triangular meshes were excluded from the actual intersection tests, with an average of 450 triangles or 20 meshes remaining, equivalent to 10 buildings. This means that only 450 ray-triangle intersection tests on average were performed for each raster cell for a single time step. Figure 10 illustrates how the techniques actually work.

In conclusion, a combination of shadow-radius- and view-frustum-based culling can significantly reduce the number of intersection tests required in the shadow-casting process.

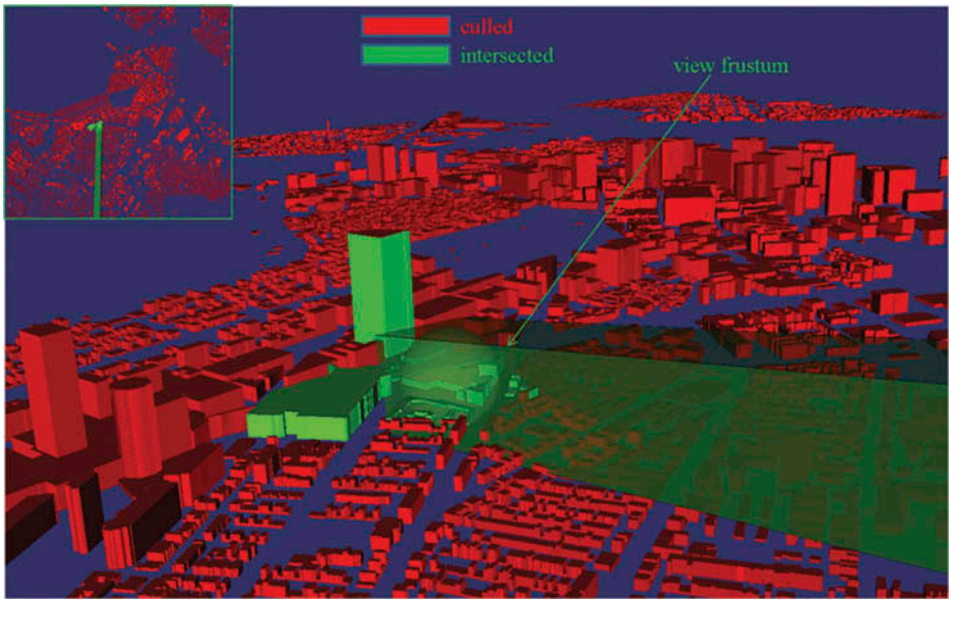

Figure 10. Illustration of the culling techniques.

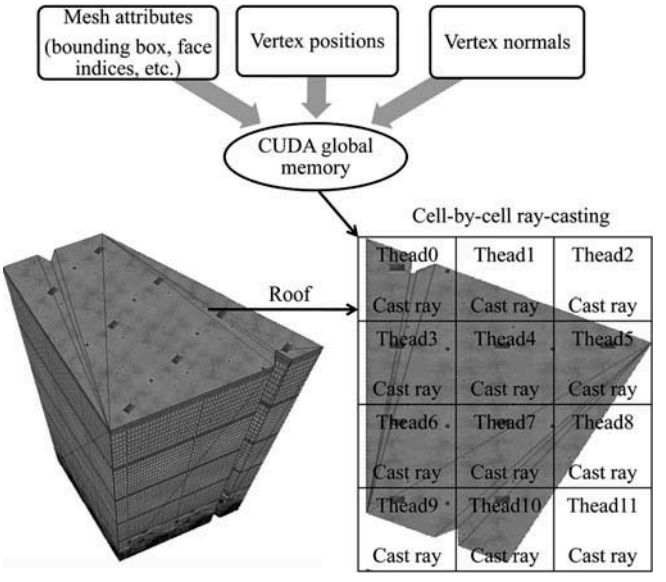

Figure 11. CUDA-based ray-casting implementation.

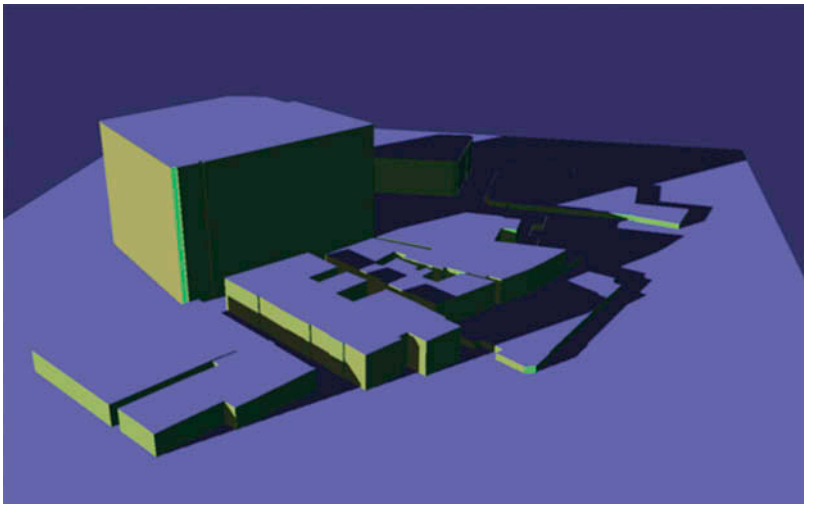

Figure 12. Accurate shadow-casting results.

However, we found that it was still overly time-consuming to calculate a shadow mask for each raster cell for all the triangular meshes in the virtual city. Therefore, we decided to parallelize the shadow-casting algorithm using the NVIDIA Compute Unified Device Architecture (CUDA). The geometric information for each triangular mesh, such as the spatial bounding box, the vertex positions, and the normal vectors, is stored in the CUDA global memory. For each cell in a raster surface, a CUDA thread is issued to perform the ray-casting calculation (Figure 11), during which the global memory is accessed to fetch the geometric data concerning the potential shadow-casters.

The GPU implementation presented herein was compared with the CPU implementation of the same shadow-casting algorithm. A small scene consisting of 11 meshes and 827 triangles was used to test the algorithm. The 11 texture maps bound to the meshes amounted to 3.67 million pixels, suggesting that 3.67 million rays needed to be cast for a single solar vector.

At first, the culling techniques were applied for both the GPU and the CPU implementations. The GPU implementation reported a computation time of less than 1 second, while the CPU counterpart reported a computation time of 8 seconds, suggesting a speedup factor of at least 8. However, the observed GPU running time increased to 1.5 seconds when culling was disabled in a later test, suggesting the significance of the culling techniques even for such a small amount of geometric data. The tests generated very accurate, view-independent, and aliasing-free shadow maps, as shown in Figure 12.

5. Discussion and conclusions

The tests described above and in this section were run on a machine with a 2.90-GHz quad-core Intel Core i52310 CPU with 4 GB of RAM. The graphics card is an NVIDIA GeForce GTS 450 with 1.0 GB RAM. The proposed computational framework was implemented using *C++*, CUDA, and OpenSceneGraph.

The date selected for calculation was January 1. The proposed method and computational framework were employed to assess the potential solar irradiance for the 22,185 buildings for time intervals of 1 hour and a raster resolution of 1 m. The time required to calculate the irradiance values was approximately 36 minutes, and another 5 minutes of computation time were required for symbolizing the raster cells in texture maps. The computation time encompassed the full time elapsed from file input through result output. Because each building was split into two parts, that is, the roof and the walls, 44,370 texture maps were generated for the 22,185 buildings, amounting to a total surface area of 31.39 million m^2.

5.1. Interactive visualization techniques

The output raster layers were symbolized using custom color ramps and rendered to RGB texture maps, which were bound to the associated triangular meshes for texture mapping in GPU-based real-time rendering.

At first, a brute-force approach was employed to load the 44,370 triangular meshes and texture maps directly into the GPU for real-time rendering. The application came to a standstill as soon as the rendering loop began and crashed before any images were generated. We then performed GPU–CPU runtime profiling to determine the possible causes of the program failure. In principle, GPU-based real-time rendering can be constrained by many potential bottlenecks, such as geometric complexity, video memory consumption, and frequent API draw calls (NVIDIA 2008). We identified frequent API draw calls as the major bottleneck, after geometric complexity and video memory issues were ruled out. Because an individual draw call is required by each texture map, 44,370 API draw calls need to be issued to generate a single frame. Because each API draw call requires a certain amount of GPU and CPU overhead, the application could be extremely inefficient in executing actual rendering tasks.

Having determined the GPU bottleneck, we decided to employ a texture atlas to improve the rendering performance by batching the draw calls (NVIDIA 2004). A texture

atlas is a large image containing a group of sub-images, each of which corresponds to the texture of a 3D triangular mesh, that is, the roof or walls of a building. In this study, a texture atlas was generated as follows:

(1) Sorting the original texture maps in ascending order by image height.
(2) Allocating an atlas image with a given width, for example, 2048 pixels, and an initial height that may be adjusted later.
(3) Drawing the sorted original images onto the atlas image row by row, keeping track of the position offset. A new atlas image is created if the current position exceeds a given image height limit, for example, 2048 pixels.
(4) Remapping the atlas images onto the original 3D triangular meshes by transforming the texture coordinates on the basis of the relative positioning of the sub-images on the corresponding atlas image.

In Figure 13, a collection of 84 images belonging to 42 buildings were merged into a single texture atlas, which was remapped onto the original building meshes for rendering with a single draw call.

In summary, the 44,370 textures were combined into 23 atlas images, each of which ranged from 1024-by-1024 to 2048-by-2048 pixels in size. The improvement in the rendering performance was significant, as suggested by a frame rate of over 200, indicating the benefit of the batched draw calls.

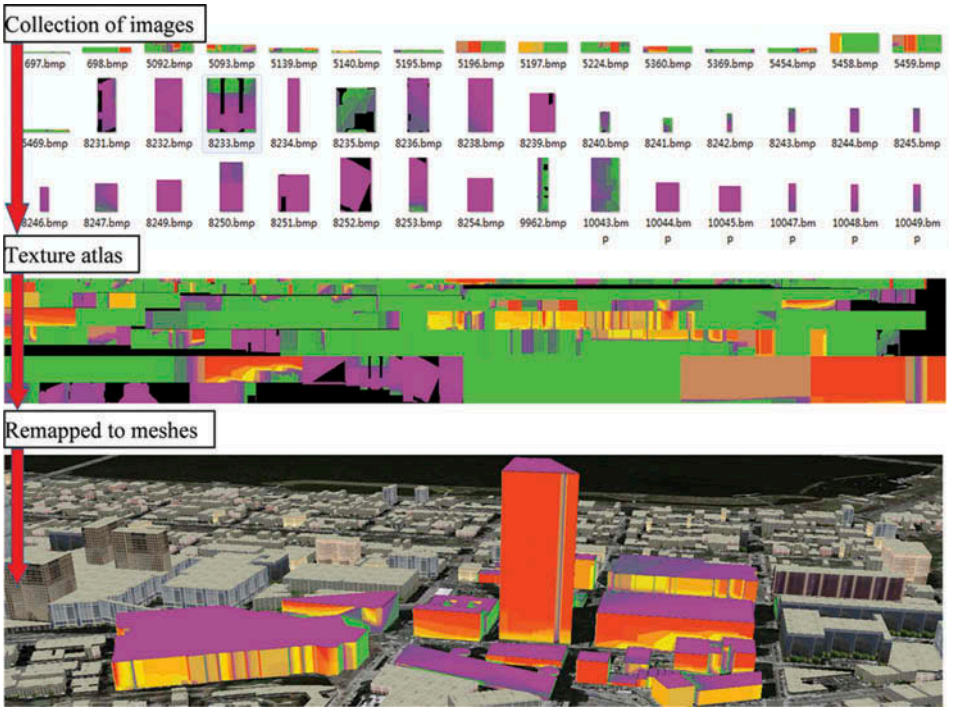

Figure 13. Texture atlas creation and visualization.

5.2. General applicability of the proposed method

The calculation results indicate that the distribution of urban irradiance is mainly affected by two factors: surface orientation and shading. Because the sun is to the south most of the time during the winter, south-facing walls are more favorable to insolation because an equal amount of solar energy falls on a much smaller area. The top view (Figure 14) suggests lower irradiance levels for the building roofs, especially for those that are shaded by the surrounding buildings. The roofs display a relatively uniform irradiance pattern due to the lack of roof orientation information. The south view (Figure 15) reveals many high-

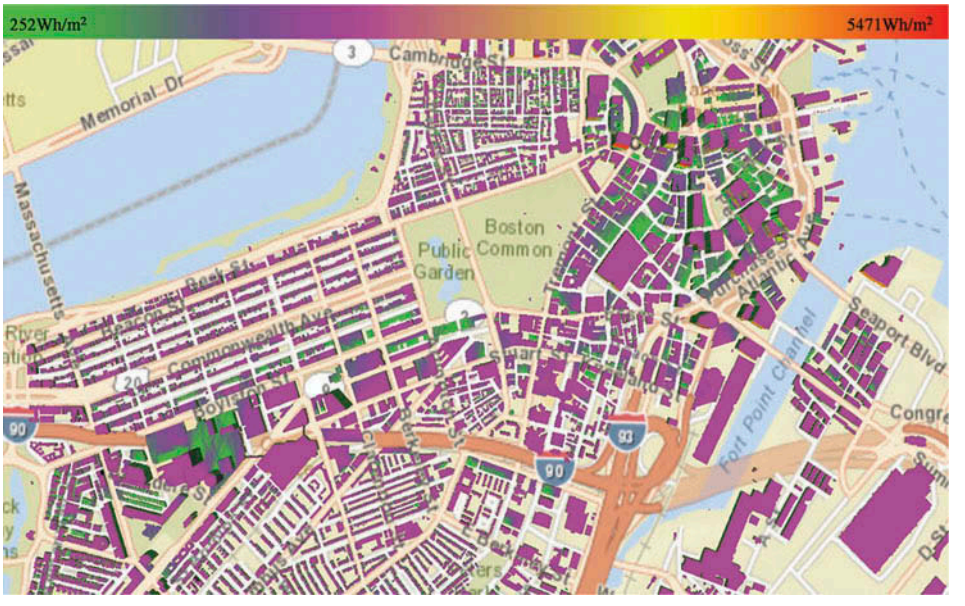

Figure 14. Top view of the solar irradiance distribution in Boston.

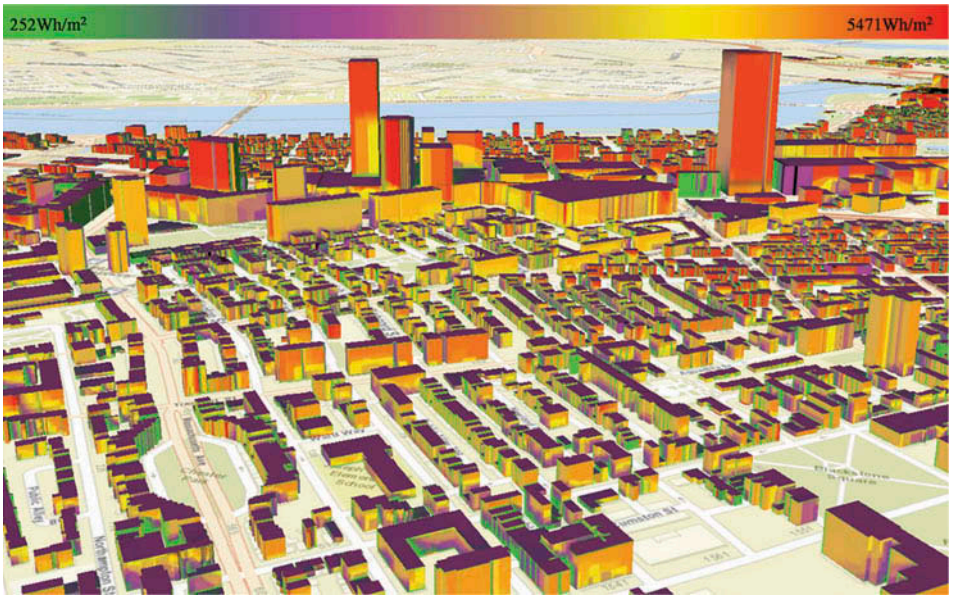

Figure 15. South view of the solar irradiance distribution in Boston.

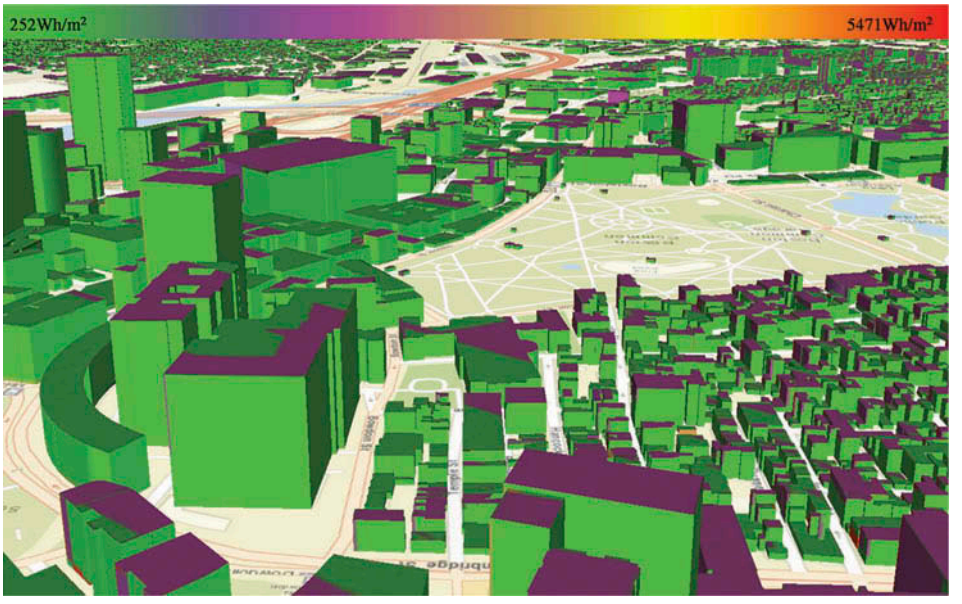

Figure 16. North view of the solar irradiance distribution in Boston.

insolation facades marked in yellow to red tones. In contrast, the north-facing facades (Figure 16) exhibit very low irradiance levels in green tones. The north-facing facades are also mostly shaded because the sun is always to the south in the winter. The buildings that lie north of taller buildings are likely to receive far less solar radiation, especially during the winter, when buildings cast longer shadows. These complicated shading effects can be clearly observed in the south-view map (Figure 15).

5.3. *Efficiency and further improvement*

The improved efficiency of the proposed method is primarily reflected in two aspects, namely, visualization and computation.

The GPU cannot normally support real-time rendering of over 5000 textured buildings without the texture atlas technique herein presented, which enables batched drawing. If the amount of surface properties grows too large to fit in video memory, the images can be compressed using the DXT1 algorithm with a fixed compression ratio of 8:1. Further optimization is possible if a view-dependent data streaming strategy is implemented with cached image pyramids.

The computation efficiency mainly depends on the shadow-casting algorithm and the r.sun model. The CPU-based implementation of both the shadow-casting algorithm and the r.sun model revealed that most of the computation time was spent on shadow-casting calculations owing to the geometric complexity. To address this issue, we resorted to a GPU-based shadow-casting implementation, which accelerated the computation by a factor of at least 8. Minimizing the number of ray-triangle intersection tests is key to improving the shadow-casting algorithm. It should be noted that about 450 intersection tests are needed for each ray using the algorithm presented in this study. Thus, there is still ample room for improvement. For instance, the computation efficiency can be expected to

increase tenfold if the number of intersection tests for each ray is to be reduced to 45. Potential techniques for further improvement include the following:

(1) Resorting to hierarchical spatial partitioning structures, for example, kdtree and octree (Crassin 2011), to minimize the number of ray-triangle intersection tests. Implementation of such an irregular spatial partitioning structure on the GPU can be more challenging.
(2) Using real-time shadowing algorithms, including shadow mapping and shadow volume (Scherzer *et al*. 2011, Kolivand and Sunar 2013), which probably require much less computational cost than the approach developed in the present study. However, both algorithms are dedicated to view-dependent interactive applications and thus further investigation will be needed to validate their accuracy.

Although the proposed method achieves efficient data representation and interactive visualization, voxel-based data representation still offers many advantages that cannot be emulated by the proposed method. For example, the shadow-casting implemented in a voxel-based data model is insensitive to geometric complexity. However, a regularly partitioned voxel model is also inefficient at storing 3D geometric properties and calculation results because there are many void spaces inside the 3D building structures. The sparse voxel octree (Laine and Karras 2010) may be more efficient at representing 3D geometric surfaces, given its highly compact storage. Nevertheless, many problems remain to be solved if the sparse voxel octree is to be incorporated in a GIS-based framework in a manner suitable to practical applications. Finally, if building surface reflectance can be retrieved using certain remote sensing techniques, it can be very interesting to study building inter-reflections and their effects on the solar radiation regimes in urban environments.

Funding

This research was supported and funded by the Key Knowledge Innovative Project of the Chinese Academy of Sciences (KZCX2 EW 318), the National Key Technology R&D Program of China (2014ZX10003002), and the National Natural Science Foundation of China (41371387).

References

Agugiaro, G., *et al.*, 2011. Estimation of solar radiation on building roofs in mountainous areas. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38 (3/W22), 155–160.

Bradley, C., 2007. *The algebra of geometry: cartesian, areal and projective co-ordinates*. Bath: Highperception.

Carneiro, C., *et al.*, 2008. Solar radiation over the urban texture: LIDAR data and image processing techniques for environmental analysis at city scale. *In*: J. Lee and S. Zlatanova, eds. *3D geo-information sciences part II*. Heidelberg: Springer, 319–340.

Compagnon, R., 2004. Solar and daylight availability in the urban fabric. *Energy and Buildings*, 36, 321–328.

Coutu, S., *et al.*, 2013. Modelling wind-driven rain on buildings in urbanized area using 3-D GIS and LiDAR datasets. *Building and Environment*, 53, 528–525.

Crassin, C., 2011. *GigaVoxels: a voxel-based rendering pipeline for efficient exploration of large and detailed scenes*. Thesis (PhD). Université de Grenoble.

Dubayah, R. and Rich, P., 1995. Topographic solar radiation models for GIS. *International Journal of Geographical Information Systems*, 9, 405–419.

Fu, P. and Rich, P., 2000. *The Solar Analyst 1.0 user manual* [online]. Helios Environmental Modeling Institute. Available from: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=063D78F145F3244277FE555E1BBF4AC9?doi=10.1.1.98.1271&rep=rep1&type=pdf [Accessed 20 October 2013].

Hofierka, J. and Kanuk, J., 2009. Assessment of photovoltaic potential in urban areas using open-source solar radiation tools. *Renewable Energy*, 34 (10), 2206–2214.

Hofierka, J. and Suri, M. 2002. The solar radiation model for Open source GIS: implementation and applications. *In: Proceedings of the open source GIS–GRASS users conference 2002*, 11–13 September 2002. Trento, Italy.

Hofierka, J. and Zlocha, M., 2012. A New 3-D solar radiation model for 3-D city models. *Transactions in GIS*, 16 (5), 681–690.

Kolivand, H. and Sunar, M., 2013. A survey of shadow volume algorithms in computer graphics. *IETE Technical Review*, 30, 38–46.

Kryza, M., *et al.*, 2010. Spatial information on total solar radiation: application and evaluation of the r.sun model for the Wedel Jarlsberg Land, Svalbard. *Polish Polar Research*, 31 (1), 17–32.

Kumar, L., *et al.*, 1997. Modelling topographic variation insolar radiation in a GIS environment. *International Journal of Geographical Information Science*, 11 (5), 475–497.

Laine, S. and Karras, T., 2010. Efficient sparse voxel octrees. *In*: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. New York, NY: ACM, 55–63.

Levy, B., *et al.*, 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21 (3), 362–371.

Liu, M., *et al.*, 2012. GIS-based modelling of topography-induced solar radiation variability in complex terrain for data sparse region. *International Journal of Geographical Information Science*, 26 (7), 1281–1308.

Lorenz, H. and Döllner, J., 2010. 3D feature surface properties and their application in geovisualization. *Computers, Environment and Urban Systems*, 34 (6), 476–483.

Mardaljevic, J. and Rylatt, M., 2003. Irradiation mapping of urban environments: an image-based approach. *Energy and Buildings*, 35, 27–35.

Merino, L., *et al.*, 2010. Solar energy inputs estimation for urban scales applications. *In: 8th international conference on system simulation in building*, 13–15 December 2010, Liège, Belgium.

Nguyen, H. and Pearce, J., 2010. Estimating potential photovoltaic yield with r.sun and the open source Geographical Resources Analysis Support System. *Solar Energy*, 84 (5), 831–843.

Nguyen, H. and Pearce, J., 2012. Incorporating shading losses in solar photovoltaic potential assessment at the municipal scale. *Solar Energy*, 86 (5), 1245–1260.

Niko, L. and Borut, Z., 2013. GPU-based roofs' solar potential estimation using LiDAR data. *Computers & Geosciences*, 52, 34–41.

NVIDIA, 2004. *Improve batching using texture atlases* [online]. NVIDIA Corporation. Available from: http://developer.download.nvidia.com/SDK/9.5/Samples/DEMOS/Direct3D9/src/BatchingViaTextureAtlases/AtlasCreationTool/Docs/Batching_Via_Texture_Atlases.pdf [Accessed 20 October 2013].

NVIDIA, 2008. *GeForce 8 and 9 Series GPU programming guide* [online]. NVIDIA Corporation. Available from: http://developer.download.nvidia.com/GPU_Programming_Guide/GPU_Programming_Guide_G80.pdf [Accessed 20 October 2013].

Rigollier, C., *et al.*, 2000. On the clear sky model of the ESRA, European solar radiation atlas, with respect to the Heliosat method. *Solar Energy*, 68, 33–48.

Scherzer, D., *et al.*, 2011. A survey of real-time hard shadow mapping methods. *Computer Graphics Forum*, 30 (1), 169–186.

Tsai, F. and Lin, H., 2007. Polygon-based texture mapping for cyber city 3D building models. *International Journal of Geographical Information Science*, 21 (9), 965–981.

Yasumoto, S., *et al.*, 2012. Virtual city models for assessing environmental equity of access to sunlight: a case study of Kyoto, Japan. *International Journal of Geographical Information Science*, 26 (1), 1–13.