Real-time visualization of 3D terrains and subsurface geological structures

Alejandro Graciano[a,*], Antonio J. Rueda[a], Francisco R. Feito[a]

[a]*Department of Computer Science, University of Jaén, EPS Jaén, 23071, Spain*

Abstract

Geological structures, both at the surface and subsurface levels, are typically represented by means of voxel data. This model presents a major drawback: its large storage requirements. In this paper, we address this problem and propose the use of a stack-based representation for geological surface-subsurface structures. Although this representation has been mainly used for volumetric terrain visualization in previous works, it has been used as an auxiliary data structure. Therefore, our main contribution in this work is its use as a first-class representation for both processing and visualization of surface and subsurface information. The proposed solution provides real-time visualization of volumetric terrains and subsurface geological structures represented as stacks using a compact data representation in the GPU. Different GPU memory implementations of the stacks have been described, discussing the tradeoffs between performance and storage efficiency. We also introduce a novel algorithm for the calculation of the surface normal vectors using a hybrid object-image space strategy. Moreover, important features for geoscientific applications such as visualization of boreholes or geological cross sections, and selective attenuation of strata have also been implemented in a straightforward way.

Keywords: Terrain modeling, Volume rendering, GPU memory management, Stack-based representation of terrains

1. Introduction

Terrain modeling and visualization are fundamental aspects of many geoscientific applications in fields like Geomorphology or Stratigraphy, but are also important in non-scientific areas such as films or videogames. The representation of terrains is usually carried out by means of the modeling of its geometry (surface and optionally subsurface) and an enhancement of the appearance by including shading, different colors for different features (e.g., elevation, materials,

*Corresponding author

Email addresses: `graciano@ujaen.es` (Alejandro Graciano), `ajrueda@ujaen.es` (Antonio J. Rueda), `ffeito@ujaen.es` (Francisco R. Feito)

etc.), or aerial/satellite imagery. Traditionally, digital elevation models (DEM) have been the preferred method for terrain surface modeling. But terrain raster representation using this 2.5D model is limited to one single elevation value for each cell. As a result, it is unsuitable for modeling complex surface/subsurface features like natural overhangs or caves. Moreover, in recent years the improvement in data acquisition and generation methods [1] [2] has provided accurate subsurface data, such as statigraphic information or location of groundwater, cavities or fractures, which require more general models than a simple DEM. This kind of data is usually represented by means of voxel models [3]. Voxel models consist of a regular 3D space partition where each cubic element (called voxel, from volumetric element) represents a single value within the grid. Voxel models are ubiquitous in many fields such as medical imaging [4], scientific visualization and simulation [5], engineering applications [6] or gaming. It has also been widely used in many geoscientific works [7] and adopted by 3D GIS software such as GRASS [8] and Mapinfo Engage [9]. However, this volumetric representation raises the problem of a large memory consumption, which can be a relevant factor during the processing and visualization of high resolution models.

A more efficient representation is to extend DEMs to store in each cell a sequence of vertical intervals of the same material or attribute instead of a single elevation value. This is a straightforward way to compact stacks of voxels with a common attribute and the same X-Y coordinate. This is not a novel idea; indeed Benes and Forsbach already introduced the *Stack-Based Representation of Terrains* (SBRT) in the context of modeling terrain erosion [10]. A main strength of this representation is that it keeps the simplicity of DEMs, making it possible to implement raster operations in an easy way. Having a simple representation that serves both for implementing raster operations on the terrain and for efficient rendering is important for many geoscientific applications.

Based on this representation, we present a real-time framework for visualization of terrain and geological structures by adapting the volume rendering algorithm based on raycasting. Our visualization method allows performing cross sections on the terrain, attenuation of stratigraphic layers or selective visualization of boreholes, among other operations. In order to achieve interactive frame rates we have implemented the volume rendering in the GPU, encoding the stacks of materials in a very compact way as a set of textures and buffers in GPU memory. This avoids expensive data transfers between GPU and CPU that can be a bottleneck in a visualization system.

We want to highlight that the SBRT is not only a convenient representation for real-time visualization of surface and subsurface of the Earth. We are also working on the definition of different geomodel conversion methods (to and from voxel models, DEMs or MultiDEMs), spatial operators, resampling methods and advanced terrain analysis. Most of these operations can be defined by adapting the Map Algebra of Tomlin [11], widely used in geospatial applications, to the SBRT. For instance, resampling can be implemented as a particular case of the *local operations* defined in this algebra. But this is still a work in progress. The design and implementation of these operations will be discussed in subsequent papers.

Through this paper we distinguish between volumetric/3D terrain and geological/subsurface structures. The first term concerns the boundaries of the terrain, i.e., the surface and elements such as caves or overhangs, whilst the second term involves the different geological features of the subsurface such as materials or aquifers.

Thus, the novel contributions of this paper are:

- A real-time rendering method for the visualization of volumetric terrains and geological structures, using a stack-based representation of terrains as main data structure.
- A comparison among different implementations in GPU of the representation proposed.
- The implementation of useful visual features for geoscientific applications in order to validate the use of the representation.
- An efficient and simple method to calculate normal surface vectors in image space.

The remainder of this paper is organized as follows. Section 2 provides a review of the existing literature. In Section 3, the stack-based representation for geomodels is outlined. Section 4 is focused on the explanation of the pipeline followed in our rendering algorithm, as well as it depicts several features of our framework. An analysis of a set of memory layouts proposed is discussed in Section 5. Finally, we conclude our work in Section 6.

2. Related work

Traditionally, spatial data in GIS are represented by vector and raster data types. 3D GIS commonly takes boundary representation models (B-rep) besides 3D lines and points, and voxel models as the natural 3D extension of 2D vector and raster data types respectively [12].

B-rep models have been represented by several structures. Irregular Networks are based on the use of the simplest geometric structures for each dimension, called simplexes. Therefore, Triangular Irregular Networks (TINs) are made up of 2D simplexes (i.e. triangles) and Tetrahedronised Irregular Networks (TENs) consist of a set of 3D simplexes (i.e. tetrahedrons). TINs have been used for the representation of geological structures in the form of surfaces. For instance, Lemon and Jones [13] used triangular surfaces in order to delimit the horizons between the strata identified from boreholes data. TINs have also been used for modeling free-form stratigraphic layers [14]

Regarding TEN representation, it was used by Caumon *et al.* for integrating geological formation data, provided by remote sensing images (stratigraphic horizons), and digital elevation models [15]. Another work presented a methodology for mesh generation in subsurface simulation modeling using finite elements [16]. An important drawback of these data structures is that checking and ensuring their topology consistency is a non-trivial task [17]. Data structures based on hierarchical decomposition such as generalized maps [18], or focusing on the representation of boreholes such as generalized tri-prism [19] have been proposed to avoid this problem.

In reference to terrain representation using voxels, Jones *et al.*, estimated the curvature in rock weathering using voxel grids [20], whereas in [21] a voxel model was constructed from geological data acquired from using Airbone Electromagnetics (AEM). More recently, a tool for generating voxel models obtained from parametrized data provided by a series of geological surfaces has been presented [22].

Usually, the visualization of these models has been done by converting them to triangle meshes as a first step [23] [24] [25]. This approach has a couple of significant drawbacks in terrain visualization applications: (1) the large amount of geometry that has to be processed by the GPU and (2) the difficulty of rendering internal elements. For these reasons, some researchers have focused on the visualization through *Direct Volume Rendering* (DVR) techniques [26]. In this field, we can find works performing raycasting [27] [28], representing volumes as multiscale vectors [29], by diffusion surfaces [30] or from implicit representations [31]. There are also works focusing on the surface of the terrain, without rendering volumetric features [32] [33] [34].

For a more comprehensive review of volumetric terrain and subsurface modeling state of art, we refer to the survey of Natali *et al.* [35].

So far the stack-based terrain representation and its variations have been used in a few scientific works, mostly focusing on the visualization at the ground level. Benes and Forsbach, precursors of this model, used it to simulate thermal erosion in a terrain. In this work, they visualized the surface of the terrain by generating a height map [10]. Peytavie *et al.* [36] got a more realistic visualization by proposing a hybrid model in which a stack-based representation (referred to as *material layer stacks representation*) serves as support for generating an implicit surface for rendering. Also, some sculpting and erosion terrain tools were added. This work was extended by Loffler *et al.* [37] by creating a pipeline for the acceleration of this surface generation. Their system achieves real-time frame rates in the rendering of high resolution models. Natali *et al.* [28] were the first to take advantage of this structure for modeling subsurface geological structures. In their work, they present a system for sketching and visualization of geomodels to help geologists to teach geological concepts. Their system works as follows: an expert sketches a series of material layers and geological elements (e.g. rivers, lakes, etc.), which are converted into multiple height maps for rendering. Unfortunately, the use of height maps only allows elements that can be represented in 2.5D, excluding features like overhangs, caves, aquifers or petroleum reservoirs.

In the previous approaches, the stack-based model plays a secondary role, to visualize complex terrain structures or erosion processes at the ground level or as an intermediate representation generated from a logical model. In this paper we propose to use the stack-based model as a primary representation for geological structures both at the surface and subsurface levels, describing

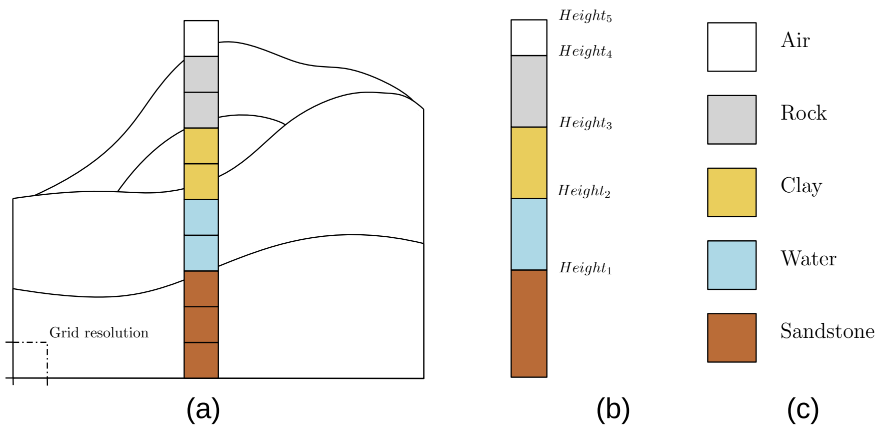


Figure 1: Stack-based representation scheme. Voxels belonging to a specific X-Y position (a), the stack resulting (b) and the set of materials (c)

a direct real time rendering algorithm in GPU with a good visual quality for scientific applications.

3. Stack-based terrain representation

This terrain representation can be considered as a generalization of common height maps. As described above, whereas height maps contain a single value for each X-Y coordinate position, stack-based representation stores a “stack” of intervals. Each of these intervals is formed by a start height and the attribute within it, similar to a run-length encoding scheme (Figure 1).

The stack-based terrain representation is a natural representation for data generated by borehole logging. A borehole provides a top to bottom sequence of materials at a given X-Y position (Figure 2). A common way to obtain a geomodel of the subsurface is by means of an interpolation/extrapolation procedure of the boreholes samples, obtaining a layer-cake model as a result [38]. This generated model fits perfectly with the stack-based representation since each cell of the terrain can store a single borehole record as a stack, including both materials and height of the geological formations (e.g. water, petroleum, clay, rock, etc.) and geological properties (e.g. density, permeability, resistivity, etc.).

Although the stack-based representation is appropriate for 3D terrain models and geological structures, it is not efficient for other kinds of volumetric data, such as medical datasets. The main problem is that such datasets may not be formed by a set of horizontal layers, thus presenting quite complex and thin structures like vessels. Therefore, the proposed representation could use even more memory than a voxel model. Moreover, volumetric medical datasets are usually acquired from CT and MRI scanners as a stack of images in which every

Table 1: Memory consumption comparison

Dataset	Number of voxels	Memory usage (MB)			Compression (%)	
		Voxel model	**Octree (6 levels)**	**SBRT**	**SBRT - Voxel model**	**SBRT - Octree**
Dataset A	200 × 250 × 320	61.03	9.87	2.55	95.8	74.1
Dataset B	400 × 500 × 400	305.17	50.06	10.10	96.7	79.8
Dataset C	800 × 1000 × 800	2441.41	570.90	52.02	97.9	90.9

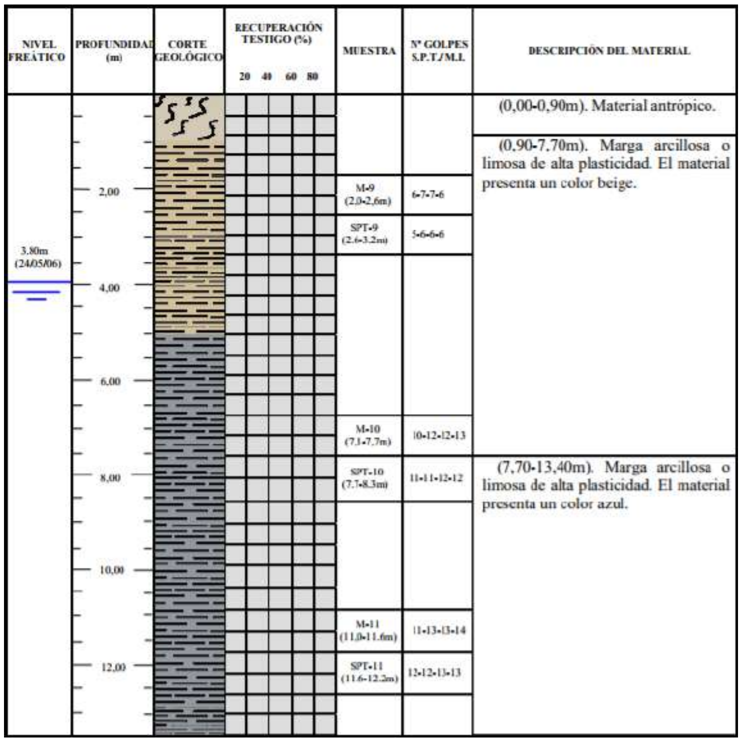


Figure 2: A real example of borehole log extracted from a geotechnical survey of the subsurface of the University of Jaén (Spain)

pixel stores an intensity value coded by a 16 or 32-bit floating point value. Thus, in order to compact then in a stack-based structure, a discretization procedure must be performed first, by labeling each intensity range with a single value. Of course this introduces an error, and even if done carefully may discard intensity values that are important for the analysis of the images.

Despite the fact that certain properties or blended materials could also be considered continuous in natural geological formations, this representation must be discrete for compression to be effective and to be able to carry out analysis operations in an efficient way. Nevertheless, an interpolation procedure can be performed for visualization purposes (Section 4).

A test to measure the memory usage of different representations, namely voxel model, octree and stack-based representation is summarized in Table 1. The datasets were obtained from the DINOloket database, which provides subsurface data from several spots in Netherlands [39]. The results show a much smaller memory usage of the SBRT compared to the rest. Our method requires 95% less space than a voxel representation. Even the memory usage of a SBRT is much lower than that of an octree, 74% less for Dataset A, 79% less for Dataset B and 90% less for Dataset C.

As already mentioned in Section 2, several geoscientific visualization research works carry out a visualization based on triangle meshes. The most usual technique for generating a triangle mesh from a volumetric dataset is *Marching Cubes* (MC) [40]. This algorithm and all its variants generate a set of triangles which represents an isosurface from a given isovalue. Realtime visualization of cross-sections or a specific stratum/material would require generating multiple meshes by applying MC on each isosurface as preprocessing. This may result in redundant geometry and in a model that would take up much more space than a SBRT. For instance, Figure 12, later in this paper, illustrates how complex a surface/subsurface model can be, and the amount of triangles which would be required to render each feature.

4. Rendering method

4.1. GPU-based raycasting

The approach described in this section is based on the well-known GPU raycasting algorithm. The key idea of the algorithm is to cast rays from the virtual camera through every pixel in the framebuffer, combining the color values sampled along the ray in a fragment shader [26].

4.1.1. Geometry setup

Rendering starts by sending a simple geometry to the GPU which serves as start and exit points of the rays. Often, as in our case, the geometry sent to the GPU is the bounding box of the volumetric dataset. This approach can lead to the problem of *empty space skipping*: A significant number of rays may pass across empty space (i.e. air, transparent material, etc.), penalizing the performance without any effective contribution to the final result. Because of this, several alternative approaches replace the bounding box with a tighter bounding geometry [41] [42]. We decided to use the bounding box in order to keep the simplicity of the model, and also because it fits the shape of a volumetric terrain fairly closely. Also, later in this section we propose a partial solution that minimizes the effects of the empty space.

4.1.2. Data access

In this step the ray samples the model, which results in a data texture/buffer access. In our method, first the X-Y position is obtained by projecting the ray to the terrain grid. Therefore, a single stack is obtained from this X-Y position. Then, the final value is sampled by iterating on the stack attributes. Depending on the sampling ray position, the stack is iterated starting at the top, if the ray is above the bounding box equator, or at the bottom, if it is below. This trivial optimization improves the performance by 15% on average. DVR methods usually store volume data in a 3D texture (or in a 2D array texture). However, there are multiple possible strategies to encode a stack-based representation of a model in GPU memory, as we discuss in Section 4.2.

Once the value is sampled we need to obtain a mapping between it and a color. DVR methods usually use a Transfer Function (TF) for this purpose. The TF is a scalar function which assigns optical properties (such as color or opacity) to each data value [4]. We simplify this function by means of a color lookup table since we encode the data as discrete values (i.e., materials). The color palette of the TF has been chosen in order to provide a good contrast among the materials and provide a visual clue to the kind of material (e.g., sand in yellow or water in blue tones). An illumination model can also contribute to the final color in order to simulate the light interaction and increase realism. Our method uses a deferred strategy: In a first pass only the value of the TF is retrieved. Then, in a second pass a normal vector is calculated from the depth buffer, and the final pixel color is calculated (Figure 3). We use a Lambertian diffuse model for the illumination according to Equation 1. Components i_a and i_d control the intensity of ambient and diffuse lighting respectively, k_d is the diffuse Lambertian reflectance, $\vec{L}$ is a direction vector from the object to the light source and $\vec{N}$ is the normal vector to the surface in a point p.

$$I_p = i_a + k_d(\vec{L}.\vec{N})i_d \tag{1}$$

Normal vectors are usually computed by simulating a density gradient, but this solution requires volume data representing intensity values, as for instance in medical imaging. Much of the work of the literature focused on the calculation of surface normals for discrete volume data take an image space approach [43] [44]. Surface normals are typically obtained by calculating the horizontal and vertical gradients for each pixel locally from the depth buffer data. The resulting shaded models using this approach might not present a smooth surface, since a voxel may cover many pixels if the model is near to the virtual camera. In contrast, an object space approach provides a better approximation of the surface since it takes into account the actual geometry of the model. For this purpose, a discrete voxel model can be represented as a binary voxel model simply by labeling each voxel as occupied or empty space. A straightforward method to obtain the normal is to convolve a $k \times k \times k$ kernel at the current position where the ray is located ($V_{0,0}$). A voxel $V_{i,j}$ of the kernel contains the unit vector direction from $V_{0,0}$ to $V_{i,j}$. The final normal vector $\vec{N}$ is calculated by summing the vectors associated to each voxel whose center are in an empty space (see Figure 4). The time consumed for this method as well as the smoothness of the surface is strongly determined by the kernel size chosen. The larger the kernel size, the smoother the surface, but more computation time is required due to the increase in the number of data accesses.

The approach we propose is hybrid: we calculate the normal vector with the convolution method described above but in image space rather than in object space. Instead of calculating the normal vector in the raycasting pass, we store in a framebuffer object the color retrieved by the transfer functionwe store the color retrieved by the transfer function in a framebuffer object (without any illumination applied). In the next pass, the resulting world position (p) of every ray casted through each pixel (r) is unprojected from the depth buffer.

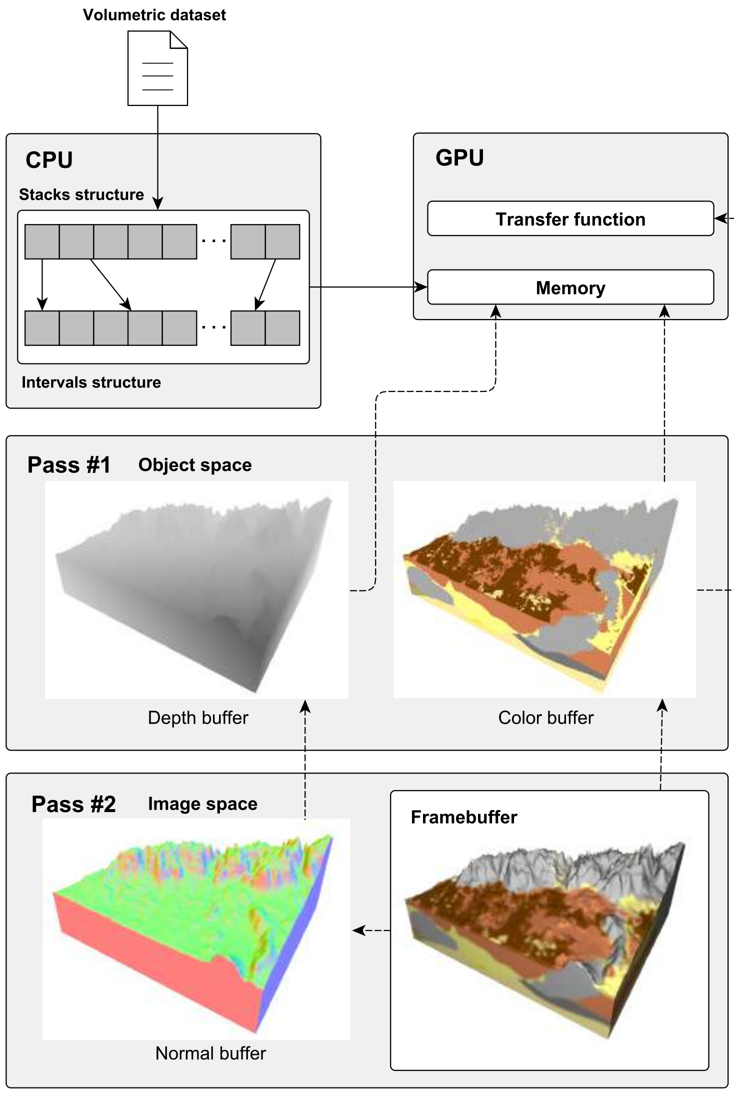


Figure 3: Deferred shading strategy used. The dotted lines indicate an usage relationship

This 3D position is the central point of an axis-oriented grid which represents a convolution kernel. Then, we calculate the depth value (d') for the center of each cell of the grid. Following this, we project the center of the cells (p') to image space and retrieve the actual depth value from the pixels obtained (d). With these values, we can assume that if $d' > d$ the cell that contains p' is labeled as occupied or as empty space otherwise. Once we calculate the occupation value for all the cells the convolution can be performed. The time consumed

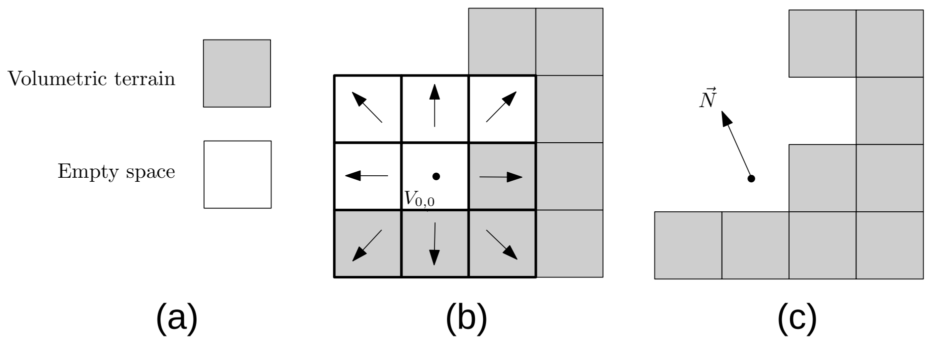


Figure 4: Calculation of a normal vector in an object space. $\vec{N}$ (c) is the resulting from the sum of the vectors contained in empty space voxels (b). Also, a legend is shown (a)

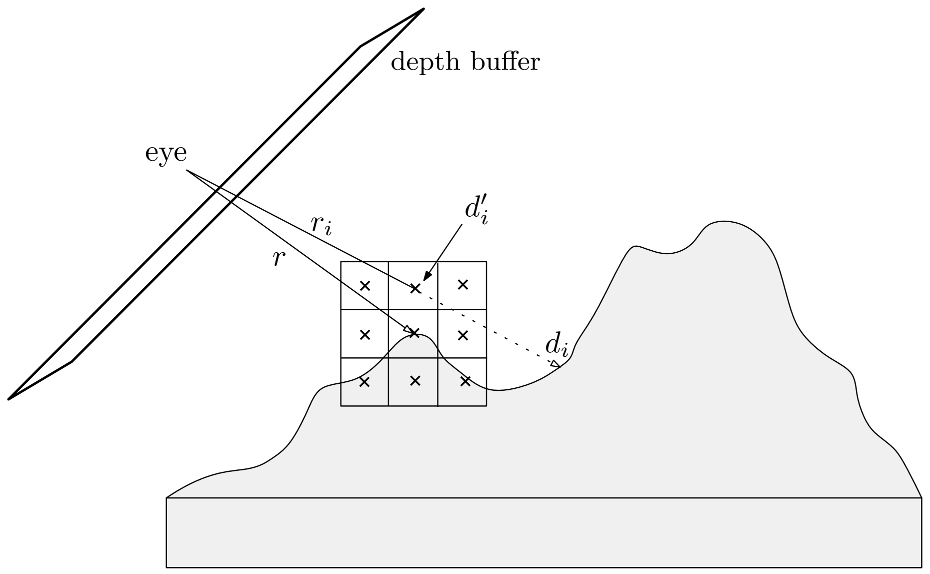


Figure 5: Calculation of a normal vector in image space. The ray r_i provides two depth values, an actual (d) and an estimated (d') with which the binary value of the kernel (occupied or empty space) is calculated

by this approach is less dependent on the size of the grid than in a strategy based on the object geometry. Table 2 and Figure 6 show a comparison of the computational cost and visual quality achieved using different kernel sizes. For a kernel size $5 <= k <= 9$, the drop in performance is affordable, therefore, in our framework, the kernel size is an adaptive parameter depending on the SBRT dimensions within this range.

Figure 5 depicts our hybrid strategy. Note that this method only can compute a reduced number of different normal vectors in comparison with gradient

Table 2: Relative drop in performance when using different kernel sizes over rendering without lighting

Kernel size	Drop in performance		
	Dataset A	Dataset B	Dataset C
$3 \times 3 \times 3$	3%	4%	3%
$5 \times 5 \times 5$	18%	21%	16%
$7 \times 7 \times 7$	43%	48%	41%
$9 \times 9 \times 9$	62%	66%	59%
$11 \times 11 \times 11$	75%	78%	72%

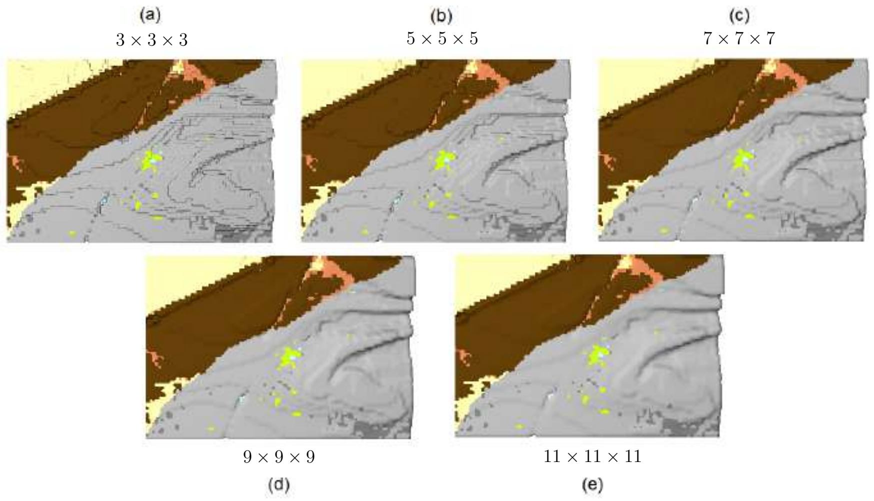


Figure 6: Visual results when applying different kernel size to a SBRT with a grid dimension of 800×1000. Each subfigure shows a region with a dimension of 200×300

estimation techniques, but in practice they are enough for a good visual quality in scientific visualization.

Several authors suggest that the normal vectors can be calculated in a previous step and passed in a buffer to the raycasting pipeline [45] [46]. However, this out-of-core approach requires significant extra memory in comparison with the techniques explained previously. This out-of-core approach would require an extension of the original SBRT including solely the normal vectors of the visible intervals. Assuming each component is encoded with a single precision floating-point number (32 bits), Dataset A (Table 1) would need an extra amount of memory of approximately 16 MB; Dataset B would need 64 MB, while Dataset C would need more than 256 MB. This memory footprint remains very large in comparison with the memory usage required by a SBRT. Applying a method for normal vector compression [47] would only save a 16% of its memory requirements on average. Moreover, it should be noted that this precomputation procedure must be carried out each time a different visual operation is requested, as explained in Section 3, which would have a significant impact on the overall system performance.

In a similar way, the color buffer resulting from the first rendering pass is subjected to a filtering process. The convolution kernel is a Gaussian smoothing operator which is performed in our hybrid scheme.

4.1.3. Compositing

The color computed at each iteration of the loop has to be combined with those calculated in previous iterations. Essentially, the composition (also called alpha blending) in DVR can be accomplished in a front-to-back or in a back-to-front order. We decided to use the former since it is more suitable to optimizations such as *early ray termination* as is outlined below.

4.1.4. Advance ray position

The reconstruction of the volumetric data as well as any other type of signal requires to set a sample rate carefully. A subsampled reconstruction will result in artifacts such as Moiré patterns whilst supersampling may considerably affect the performance. In order to avoid these two drawbacks, we choose an adaptive sampling strategy. This strategy advances the ray with a varying step size depending on the traverse mode: empty space skipping mode (coarse step) or sampling mode (fine step). At first, in empty space skipping mode, the step size is equal to the resolution of the terrain cell. This ensures a single access per stack. Once a collision with a non-empty value is detected, the ray moves backwards one step and the traverse mode is changed to sampling mode. From here, the step size is reduced according to the Nyquist-Shannon sampling theorem: the sampling rate must be at least twice as high as the maximum frequency of the signal. In a DVR context, the maximum frequency (f_m) is a 3D vector in which each component corresponds to the minimum resolution in each dimension: the resolution of the grid cell for X and Y components and the minimum relative height from among all the stacks for Z component. Therefore, in sampling mode the step frequency (f_s) will be $f_s >= 2f_m$.

4.1.5. Ray termination

The criterion used for an early ray termination is based on the composed opacity. If the opacity is above some threshold $(1-\epsilon)$, we can assume that new contributions to the alpha blending will be irrelevant, and therefore the traverse is stopped.

4.2. Texture management

An issue that produces a major impact on the performance of any rendering method is the texture access. Fetching a texel is one of the most time-consuming operations on the GPU, but fortunately texture access exhibit spatial and temporal coherence that can be exploited to improve performance. Modern GPU architectures use caches where recent texture reads are stored following an LRU strategy [48]. A further refinement is the use of *texture swizzling* strategies [49].

To improve cache coherency, texture swizzling uses space-filling curves such as Peanno-Hilbert or Morton sequence for data organization in the texture. Also, approaches based on the view direction in raycasting have been proposed [50]. In our case, we focus on cache-friendly strategies at the stack representation level.

In the following, we show different memory layouts for the storage of a stack-based representation on GPU memory. The procedure to sample a particular attribute by the raycaster is divided into two steps: the stack identification and its iteration. These two steps are common to every proposed layout; the only differences are the number of textures (or buffers) used and how they are traversed.

In a first approach (namely 1-Textures, Figure 7.b) we maintain a couple of 2D textures: an indices texture and an intervals texture. The aim of the former is to serve as a spatial index for the stacks descriptions stored in the intervals texture. Indices texture has the same dimensions than the support grid of the stack-based representation ($grid_{width} \times grid_{height}$); hence the related stack is obtained by projecting the X-Z components of the ray position on this grid. This projection provides a 2D index which is used as an index for the indices texture. For each texel of this texture we use the components R and G from the originals RGBA. In R component we encode a pointer (i.e., a 1D index) to the intervals texture while the number of intervals in this stack is encoded in G component. The intervals texture stores each stack in a row-major order. Here we also use two components to encode an interval: the R-G components store its attribute and the accumulated height respectively. The pointer (i_0) and the number (n) of intervals stored in the indices texture marks the beginning and the end of the stack in the intervals texture, so that the required interval is obtained by iteration between the i_0 and i_{n-1} positions.

In the second approach (2-R-Supertexels, Figure 7.c), a single 2D texture is used. In this texture we encode two levels, one for indices and one for intervals. We construct the texture by first obtaining the maximum number of intervals among all the stacks, m. Then, in order to store each stack, we create regions of $\lceil\sqrt{m}\rceil x \lceil\sqrt{m}\rceil$ dimensions, where $\lceil x \rceil$ is the ceiling operator. These regions act as "supertexels" to provide a pseudoindices scheme. Therefore, the actual dimensions of the texture are calculated by:

$$\begin{aligned} texture_{width} &= grid_{width} * \lceil\sqrt{m}\rceil \\ texture_{height} &= grid_{height} * \lceil\sqrt{m}\rceil \end{aligned} \tag{2}$$

This strategy allows a straightforward way to get the related stack since all the regions are squares. Similarly to the previous approach, the ray position must be projected in the support grid, but, in contrast, an offset has to be added to the cell obtained. Once the region has been identified, we only have to iterate through it to locate the interval. Also, this texture encodes the attribute and the height of an interval in their R-G components. Alternatively, the stacks can be saved in non-squared regions by finding the rectangle with the minimum

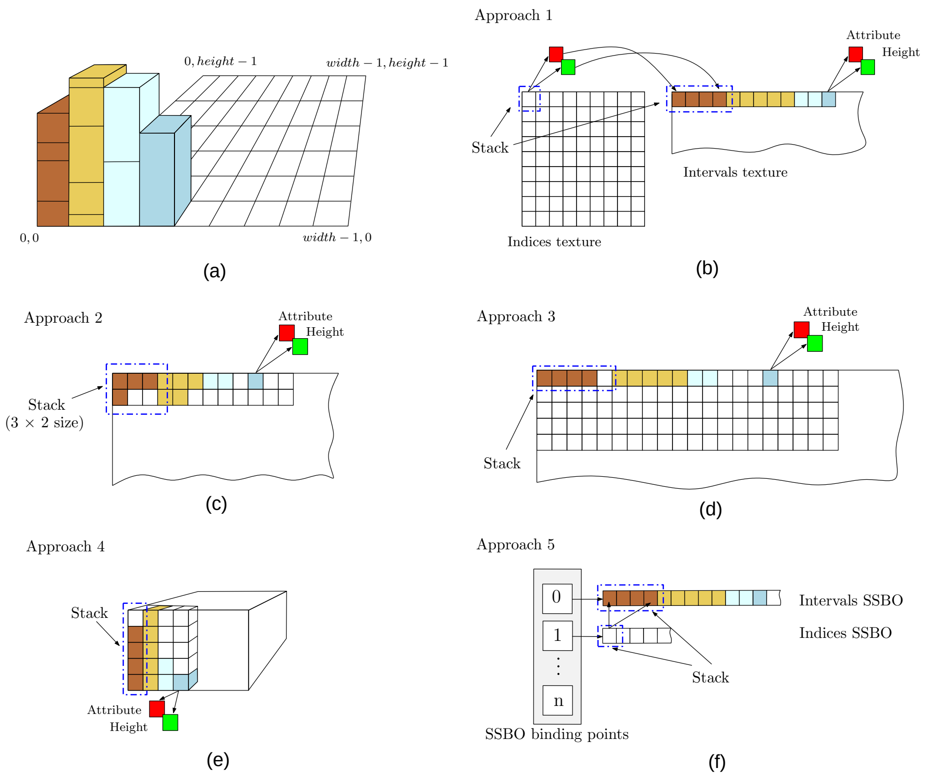


Figure 7: Memory storage patterns (b, c, d, e, f) for a stack-based geomodel (a)

perimeter. This can be formulated as a simple optimization problem:

$$\begin{aligned} \underset{width,height}{\text{argmin}} \quad & width + height \\ \text{subject to} \quad & width * height >= m \\ & width > 0 \\ & height > 0 \end{aligned} \tag{3}$$

This is the preferred layout in this approach when m is not prime. Otherwise the former layout (square regions) should be chosen.

Another possible approach is that proposed by Natali *et al.* [28]. They used a 3D texture of $width \times height \times m$ dimensions in which the intervals are stored along the z axis (approach 4-3D-Texture, Figure 7.e). In contrast, we use a 2D texture of $(width * m) \times height$ dimensions (approach 3-L-Supertexels, Figure 7.d). The stack identification is accomplished in a similar way as explained above, but with the difference that we only have to add an offset to the x index. Then, the stack iteration will be done in a linear manner.

Despite in DVR the prevailing memory schemes use textures to store volumetric data; current graphics visualization APIs provide other ways to send data to the GPU such as Uniform Buffer Objects (UBO). Recently OpenGL in its 4.3 version has included a new method for storing large amount of data in the GPU, the Shader Storage Buffer Objects (SSBO). Actually, a SSBO is an enhanced and more flexible version of UBO: OpenGL implementations must support UBOs of at least 16KB whilst the guaranteed minimum for a SSBO is 128 MB. Furthermore, the specification of SSBO allows the definition of a non-fixed size array. A SBRT can be represented in a straightforward way as an array of intervals, being this scheme well suited for its storage in a SSBO due to its internal memory layout: In conjunction with SSBOs, OpenGL introduced the std430 layout which packs scalar arrays more efficiently than the old std140 [Sellers16]. An approach based on SSBOs is also proposed (approach 5-SSBO). Similarly to approach 1-Textures, we populate two SSBOs, one for indices and another for intervals (Figure 7.f).

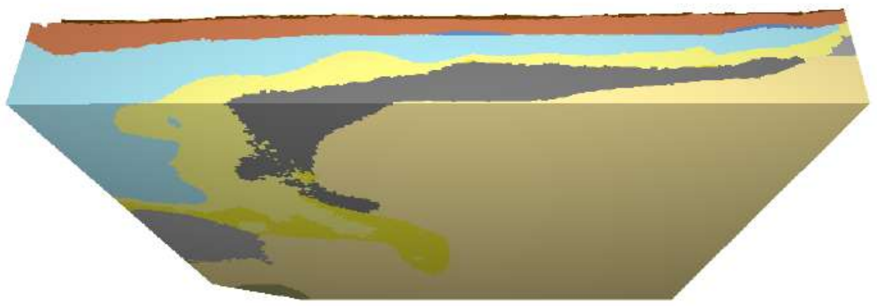
In order to clarify these approaches, an illustrative example is explained. Given a SBRT with a grid of $width \times height$ cells and a maximum stack size of 5 intervals (Figure 7.a), the memory storage patterns are described next: in approach 1-Textures, the indices texture has the same dimension that the grid of the stack-based representation ($width \times height$), being the interval data sequentially added through the interval texture. To set the size of the interval texture, we first need to know the total number of intervals of the stack-based terrain. Then, with Equation 3 we can determine the texture dimension assigning to m the total number of intervals. If this value is prime the Equation 2 can be used. Also, this value can be increased in order to use Equation 3. Approach 1-Textures can waste some memory since the last cells of last row usually are not used in this latter case. For approach 2-R-Supertexels it is necessary to set the size of the supertexels. The largest stack determines this value (5 intervals, in the example). The texture contains supertexels of 3×2 texels. The approaches 3-L-Supertexels and 4-3D-Texture are similar. Finally, for the approach 5-SSBO two arrays containing the indices and the intervals are stored in two separate SSBOs. This example is shown in Figure 7.

4.3. Visual operations

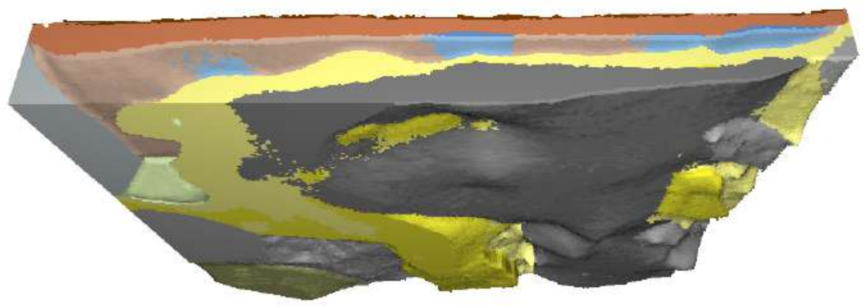
In this subsection we describe a set of operations common in geoscientific applications and their implementation in our system. These operations, such as the selective visualization of boreholes or the hiding of strata of materials, provides geoscientists with visual tools to take decisions at a glance.

4.3.1. Strata visualization

With the aim of showing hidden elements of the subsurface, each stratum can be attenuated or directly excluded from visualization. This is achieved by simply modifying the opacity of the material color in the color lookup table. By using raycasting for rendering, it is not necessary to construct a new geometry when hiding certain layer. Figure 8 shows some examples of layers attenuation.



(a)



(b)

Figure 8: Example of layer attenuation (b) of an original dataset (a)

4.3.2. Cross section visualization

Another way to visualize internal structures is via geological cross sections. These are 2-dimensional slices of the subsurface, usually vertical, used to study the the distribution of rock types, including their ages, relationships and structural features. Our system not only allows to perform sections by vertical cutting planes, but also by any freely-oriented plane in 3D space. In this case, the rendering algorithm will start at the intersection of the casted ray with the cutting plane. Figure 9 illustrates this feature.

4.3.3. Borehole log visualization

Borehole records are obtained by drilling the rock core and they contain relevant information on lithology, stratum thickness or physical properties of

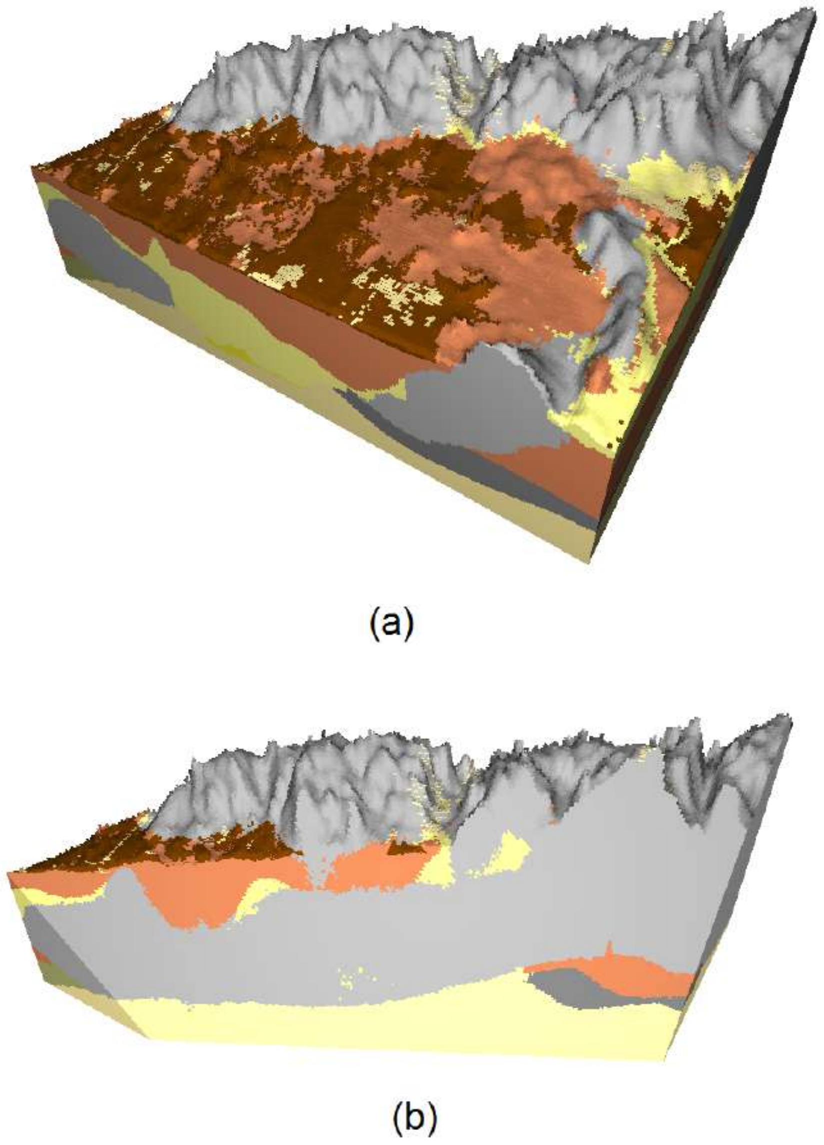


Figure 9: Cross-section (b) and original model (a)

the geological formations (Figure 2). The samples brought to the surface or the measurements made by the instruments lowered into the hole are mostly

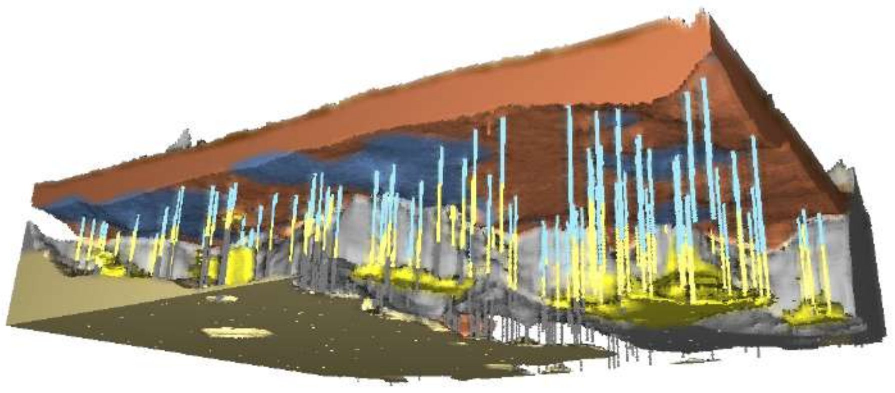


Figure 10: Example of borehole visualization. They are shown as cylinder of two materials (blue and yellow color)

the input data for the construction of subsurface models. We can simply label a stack as corresponding to a borehole by using a Boolean flag in the indices texture. When borehole log visualization is selected, only the stacks with this flag set are shown. Figure 10 shows an example of borehole log visualization.

4.3.4. Applying textures to terrain

We also provide an operation for adding an image texture to the surface of the terrain such as orthophotos or topographic maps. In a direct way, the pixel color can be obtained from the image and optionally combined with the material color, provided that the ray intersects with the terrain surface. In Figure 11 an example of this feature is shown.

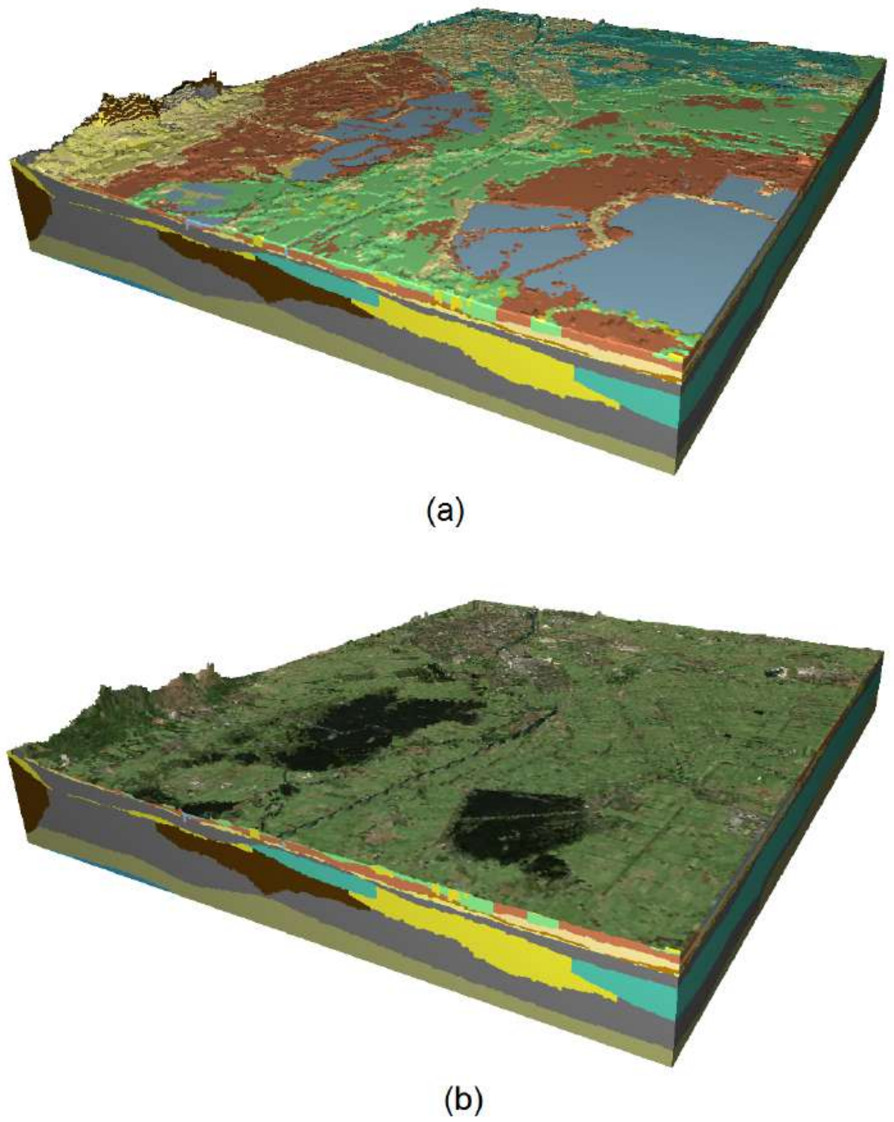


Figure 11: Example of orthophoto application. Original model (a) and model with an orthophoto applied on its surface (b)

5. Performance analysis

5.1. Comparison of memory layouts

To test the approaches described in Section 4.2), we performed a set of experiments. We measure and compare the millions of rays per second (MRPS)

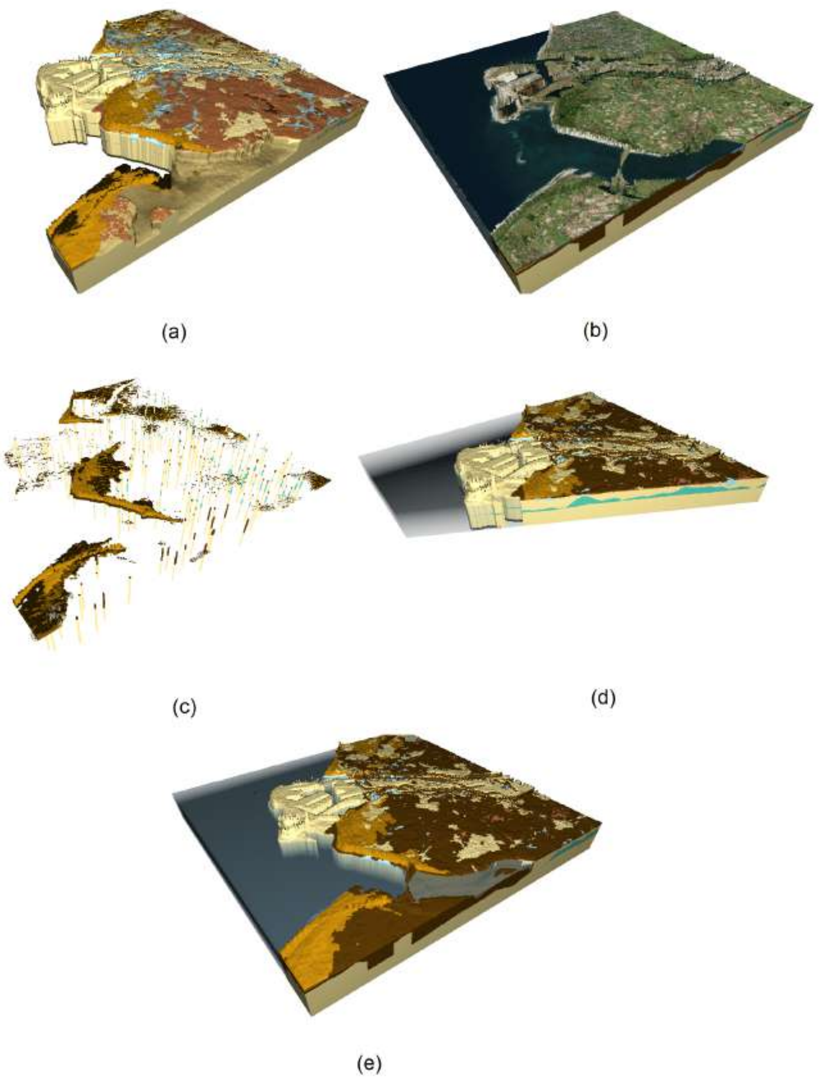


Figure 12: An original model (e) and different examples of visual operations applied: hiding of many layers (a), application of an orthophoto on the surface (b), borehole visualization combined with layer hiding (c) and visualization of cross-section (d)

by rendering the different layouts and their memory usage. We therefore tested four datasets with different grid dimension and stack sizes (previously used in Section 3); 200 × 250 with a number of 10 intervals maximum per stack, which corresponds to a model of 200 × 250 × 320 voxels (Dataset A); 400 × 500 with 15 intervals, which corresponds to a model of 400 × 500 × 400 voxels

(Dataset B) and 800 × 1000 with 16 intervals, which corresponds to a model of 800 × 1000 × 800 (Dataset C). Finally, Dataset D is a modification of Dataset C by deleting several layers and visualizing some stacks, see Figure 13. These datasets were also obtained from the DINOloket database. To measure the impact of raycasting direction we rendered the dataset from different virtual camera positions.

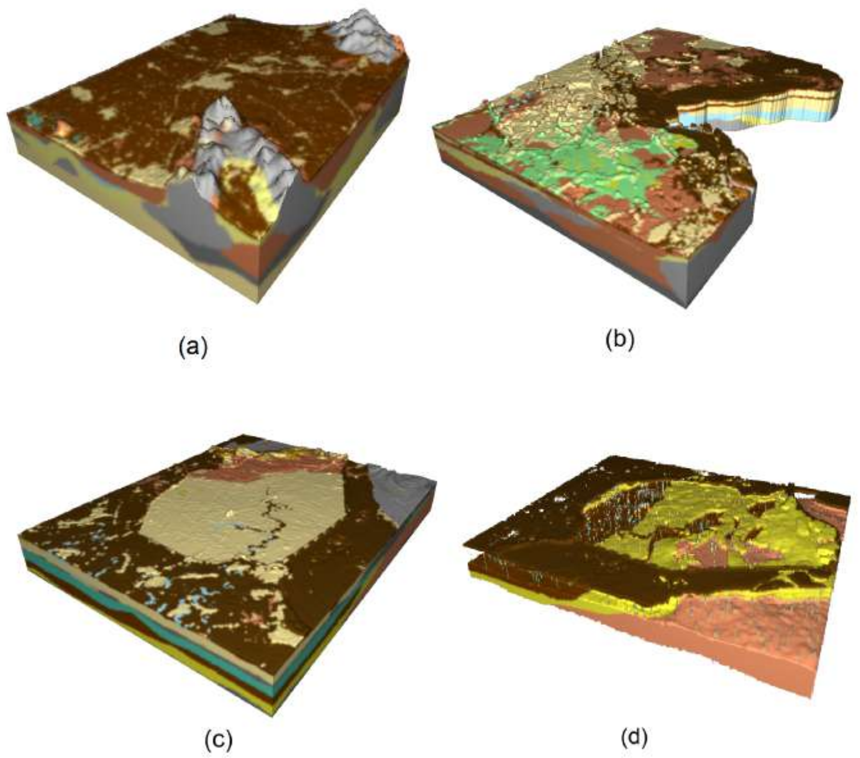


Figure 13: Overview of the datasets used in the experiments. Dataset A is shown in the left-upper corner (a), Dataset B is in the right-upper corner (b), Dataset C is in the left-lower corner (c) and Dataset D is in the right-lower corner (d)

The hardware setup for the experiments consisted of a PC equipped with an Intel Core i7-4790 CPU running at 3.60 GHz with 16 GB of RAM and an NVIDIA GeForce GTX 970 graphic card. The framework proposed has been implemented in C++ and OpenGL 4.5 as 3D graphic library.

Regarding the implementation of the textures and buffers involved in the experiments in Figure 7, we declared a 2D texture of indices with `GL_RG32UI` internal format and a 2D texture of intervals with `GL_RG16F` internal format for approach 1-Textures. The approaches 2-R-Supertexels and 3-L-Supertexels used a single 2D texture with `GL_RG16F` as the internal format. Also, we compared

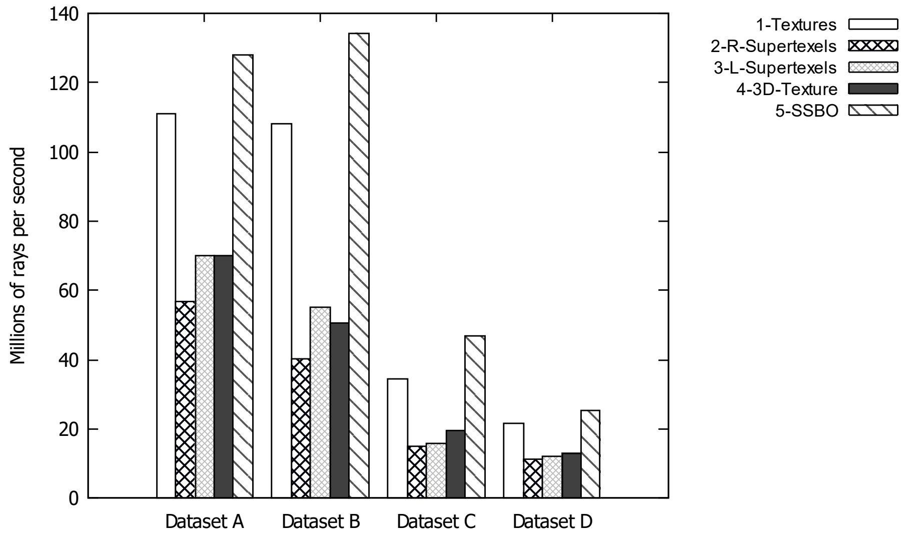


Figure 14: Minimum MRPS reached

the layouts proposed with the 3D texture strategy presented in [28] (approach 4-3D-Texture). Likewise, the internal format of the 3D texture was `GL_RG16F`. For approach 5-SSBO, the indices SSBO is filled by an array of structures formed by two 32-bits unsigned integers, whilst the Intervals SSBO by two 16-bits floating-point numbers. The rendering method has been performed on a 1056 $\times$ 884 viewport.

Figure 14 reports the minimum value of MRPS reached in the experiments for each dataset and approach (worst case). This case usually occurs when the ray has to traverse more empty space. On the other hand, Figure 15 shows the maximum value of MRPS reached (best case).

The plot referred to the worst case shows a similar pattern for each dataset. Approaches 1-Textures and 5-SSBO perform better than others. Due to the fact these approaches are linear, the prefetching and the cache work better at stack level. Furthermore, the layout provided by OpenGL for the SSBO seems better than approach 1-Textures. In Figure 15, results are less conclusive, obtaining in all datasets more than 200 MRPS: more than 200 frames per seconds for the chosen viewport dimensions.

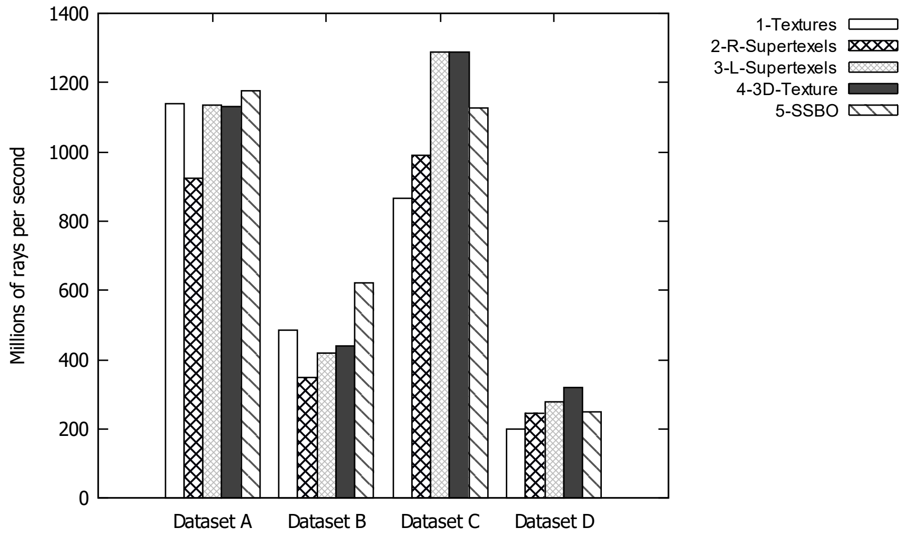


Figure 15: Maximum MRPS reached

The layout of approach 3-L-Supertexels presents some restrictions. The approach cannot be used for very large models. The graphics cards support textures up to a maximum size. For the card used in our experiments, this maximum size is 16384 $\times$ 16384 for 2D textures. Therefore, in order to encode a dataset with a dimension of 1024 $\times$ 1024 and more than 16 maximum intervals per stack, this layout needs at least a 17408 $\times$ 1024 2D texture, exceeding the GPU limits.

The memory requirements of each layout as well as the wasted memory have also been summarized in Table 3, being approaches 1-Textures and 5-SSBO the most efficient packing the data. In these approaches, the intervals storage (texture or SSBO) does not need a fixed interval size in order to identify a stack due to the indices storage. However, as can be noted, approach 1-Textures requires some extra memory for Dataset A that can be neglected. This is because the total number of intervals, 309433 in this case, is a prime value. In order to decompose the set of intervals in a rectangular texture, we summed one to this value to be able to use Equation 3 (obtaining a 479 $\times$ 646 texture). The outcomes for the remaining approaches show a noticeable wasting of memory storage.

In general, the approach 5-SSBO can be considered the best one in terms of performance since it reached the higher MRPS values on average. Moreover, the approaches 5-SSBO and 1-Textures provide the best results in memory usage and in the ratio consumed/wasted. SSBOs can use all the available GPU memory, which is considerably higher than the maximum size allowed for textures (as stated above, 16384 $\times$ 16384 dimensions). Therefore, the approach 5-SSBO is able to manage datasets larger than those who can be holded in 1-Textures,

Table 3: Storage requirements of memory layouts

	Dataset A			**Dataset B**			**Dataset C**		
	Wasted memory (MB)	**Consumed memory (MB)**	**Percentage of wasting (%)**	**Wasted memory (MB)**	**Consumed memory (MB)**	**Percentage of wasting (%)**	**Wasted memory (MB)**	**Consumed memory (MB)**	**Percentage of wasting (%)**
1-Textures	3.81×10^{-6}	1.56	2.4×10^{-4}	0.00	6.20	0.0	0.00	29.06	0.0
2-R-Supertexels	0.73	1.91	38.1	6.77	11.44	59.2	25.87	48.83	53.0
3-L-Supertexels	0.73	1.91	38.1	6.77	11.44	59.2	25.87	48.83	53.0
4-3D-Texture	0.73	1.91	38.1	6.77	11.44	59.2	25.87	48.83	53.0
5-SSBO	0.00	1.56	0.0	0.00	6.20	0.0	0.00	29.06	0.0

without the use of an out-of-core strategy. However, SSBOs are only available in modern GPUs supporting OpenGL 4.3 or higher. Consequently, approach 1-Texture provides a more general solution for older hardware.

5.2. Comparison with hierarchical data structures

Hierarchical space partition schemes such as quadtrees, octrees or kd-trees are widely used to represent and visualize environmental data using voxel models [51] or polyhedral meshes [52] [53]. In general, most of the papers focused on voxel models deal with binary datasets or use an isosurface in order to represent only their boundaries [54] [46] [55]. Nevertheless, there are some works that consider internal materials with which a comparison can be made, for instance, the work presented by Crassin *et al.* [56]. In Section 3 we already showed how SBRTs have far lower memory requirements than octrees (between 69% and 86% less), but this data structure has several additional limitations when used for representing volumetric terrains, as we explain next.

A first disadvantage in relation to a SBRT is that using a hierarchical structure is not the ideal representation for some simulation or analysis applications, as in the case of physical-based erosion simulations [57] [36]. This could involve too many ascents and descents in the octree hierarchy [58] and possibly lead to its reconstruction after the simulation process. On the contrary, a SBRT can be used both for rendering and analysis purposes in a direct and natural manner. Another issue to consider is that usually the volumetric dataset must have power-of-two dimensions. For instance, Dataset A originally has a $200 \times 250 \times 320$ dimension; therefore a dataset with $512 \times 512 \times 512$ voxels must be built. While it is true that this does not significantly affect performance, this does increase memory requirements and hierarchy construction time. A final disadvantage in relation to a SBRT is when a set of borehole logs is rendered. Each time a new borehole is added or removed to the visualization, a new hierarchy must be computed. In contrast, our approach performs these updates in real time. It should be noted that the efficiency of traversing the resulting hierarchy by rendering borehole logs could be $O(n)$ if they are dispersed through the volume.

In order to test the performance of a hierarchical structure with volumetric terrains, we compared the results obtained for our SBRT approach (5-SSBO) with the method proposed in [56], using the software provided by the authors, known as Gigavoxels. The method is an out-of-core approach, ensuring the rendering of datasets of billions of voxels. The system uses a memory layout based on bricks and node pools that, for a typical usage case, has a constant memory consumption on GPU of 459 MB; quite higher than the results obtained from SBRT layouts. The last disadvantage compared to the straightforward SBRT strategy is the preprocessing time required to compute the octree hierarchy. Although presumably the system uses a GPU acceleration to carry out this task, this process can take several minutes. In this specific case, as it is an out-of-core strategy, this time is even greater since a series of files have to be created. These results are summarized in Tables 4-7.

Table 4: Comparison of results for Dataset A. FPS means frames per seconds and MRPS, millions of rays per second

Approach	**FPS (worst)**	**FPS (best)**	**MRPS (worst)**	**MRPS (best)**	**Preprocessing time (s)**	**GPU memory usage (MB)**	**Main memory usage (MB)**
SBRT	137	1259	127.89	1175.28	0	1.56	0.00
Gigavoxels	158	560	147.49	588.76	115	459.33	286.86

Table 5: Comparison of results for Dataset B. FPS means frames per seconds and MRPS, millions of rays per second

Approach	**FPS (worst)**	**FPS (best)**	**MRPS (worst)**	**MRPS (best)**	**Preprocessing time (s)**	**GPU memory usage (MB)**	**Main memory usage (MB)**
SBRT	144	664	134.42	619.84	0	6.19	0.00
Gigavoxels	194	654	181.10	610.51	115	459.33	286.86

Table 6: Comparison of results for Dataset C. FPS means frames per seconds and MRPS, millions of rays per second

Approach	**FPS (worst)**	**FPS (best)**	**MRPS (worst)**	**MRPS (best)**	**Preprocessing time (s)**	**GPU memory usage (MB)**	**Main memory usage (MB)**
SBRT	50	1208	46.67	1127.68	0	29.06	0.00
Gigavoxels	241	690	224.974	644.12	920	459.33	2294.86

Table 7: Comparison of results for Dataset D. FPS means frames per seconds and MRPS, millions of rays per second

Approach	**FPS (worst)**	**FPS (best)**	**MRPS (worst)**	**MRPS (best)**	**Preprocessing time (s)**	**GPU memory usage (MB)**	**Main memory usage (MB)**
SBRT	27	265	25.20	247.38	0	29.06	0.00
Gigavoxels	24	46	22.40	42.94	920	459.33	2294.86

Regarding the efficiency of rendering, for Datasets A, B and C Gigavoxels reached a higher peak of MRPS in the worst case, being this gain slightly higher for Datasets A and C. However, in all datasets, our structure performs better than Gigavoxels when the ray hits quickly with non-empty data. As we stated before, when a set of isolated stacks is visualized, as in the case of Dataset D, an octree based solution is not efficient. If the boreholes are dispersed, the ray advances linearly instead of logarithmically. For this dataset, our approach achieves better results.

But of course octrees have a major advantage: they scale better than non-hierarchical representations, i.e., as datasets size grows, the impact over the performance will be lower. This is mainly due to their ability to compact empty regions, accelerating empty space skipping, one of the bottlenecks of our system. Also, rendering performance could be improved by including a level-of-detail scheme, such as that provided by quadtrees and octrees structures. Additionally, in some cases a hierarchical partition of the space could be beneficial for analysis and simulation procedures. This could allow the execution of an operation on a set of similar stacks within a region, thus improving the execution time. In order to include such benefits in our system, a 2D space partitioning solution with a stack as minimal element could be explored. Nevertheless, our framework is capable of rendering volumetric terrains with a grid dimension higher than 3000×4000 at interactive frame rates.

6. Discussion and conclusion

We have presented a real-time visualization system for surface-subsurface geological models. To accomplish this, we used a stack-based representation. We have highlighted their low memory requirements in comparison with other commonly used structures such as voxel grids or octrees, validating its use for the management and visualization of geomodels. Also, we have tested several approaches based on typical textures and the new SSBOs, to store this representation on GPU reaching an outcome quite acceptable for interactive applications. It should also be pointed out that, as far as we know, the use of SSBOs to store volumetric data has not been explored in the literature yet. Additionally, our paper presents an accurate, simple and efficient method to calculate normal surface vectors in image space. Finally, we have shown how several common visual operations of interest in geoscientific applications can be implemented in a straightforward way using the SBRT and the described visualization method.

Nevertheless, our rendering method can be improved in several ways. Although we have demonstrated that our solution can compete with more sophisticated data structures such as hierarchical grids with regard to performance, it is possible to adapt this idea to our data structure. Another important aspect is the visual quality. For instance, even though the method to compute the normal vectors for lighting could be adequate for geoscientific applications, it does not provide very realistic shading. Since discrete data are visualized, filtering methods can be applied by performing a low-pass filter in zones where material changes. As stated before, in this work we opted for a simple visualization pipeline, but any further method to improve the visual quality can be added.

A limitation of the approaches here described is that their support for editing operations is limited. Operations such as deleting intervals or changing their materials or sizes can be implemented in a trivial way; however, the addition of new intervals in any part of the stack requires a special treatment in the data structure. A straightforward solution would be to mark the affected intervals as deleted, recreating them in a extra free space at the end of the GPU memory reserved for the storage of the SBRT. After a certain number of these operations, or if there is not free space, a complete reconstruction of the SBRT can be performed. These update or reconstruction operations can be implemented in the GPU in a easy and efficient way. In contrast, updating an hierarchical spatial data structure would be much more difficult and computationally expensive. The approaches 1-Textures and 5-SSBO are easily adaptable to incorporate this mechanism; however, the rest of the approaches contain a data organization less suitable for updating processes.

In addition, in this work we have focused solely on geological features, but 3D GIS applications also need the management of vector data aside from raster data. Following this, we plan to extend the stack-based representation to handle at the same time both data types. This hybrid representation would allow consistent operations between data of different nature. Also, a framework for the analysis and visualization of geomodels that operates entirely on the GPU using GPGPU technologies such as CUDA or preferably OpenGL Compute Shaders is planned. Finally, the stack-based representation and the rendering method proposed can be implemented as a module for open-source GRASS GIS, since this tool handles volumetric terrains and subsurface structures by means of voxel models. Moreover, our rendering algorithm would improve one of the main drawbacks of GRASS, i.e., its 3D visualization.

Acknowledgment

This work has been partially funded by the Ministerio De Economía y Competitividad of Spain under the I+D+i research program TIN2014-58218-R, and by the University of Jaén through the predoctoral research grant Acción 15.

References

[1] J. Mateo Lázaro, J. Á. Sánchez Navarro, A. García Gil, V. Edo Romero, 3D-geological structures with digital elevation models using GPU programming, Computers & Geosciences 70 (2014) 138–146. `doi:10.1016/j.cageo.2014.05.014`.

[2] J. Guo, L. Wu, W. Zhou, J. Jiang, C. Li, Towards Automatic and Topologically Consistent 3D Regional Geological Modeling from Boundaries and Attitudes, ISPRS International Journal of Geo-Information 5 (2) (2016) 17. `doi:10.3390/ijgi5020017`.

[3] J. Hughes, J. Foley, Computer Graphics: Principles and Practice, The systems programming series, Addison-Wesley, 2014.
URL https://books.google.es/books?id=OVpsAQAAQBAJ

[4] J. J. Caban, S. Member, P. Rheingans, Texture-based Transfer Functions for Direct Volume Rendering, IEEE transactions on Visualization and Computer Graphics 14 (6) (2008) 1364–1371. doi:10.1109/TVCG.2008.169.

[5] A. Jjumba, S. Dragićević, Towards a voxel-based geographic automata for the simulation of geospatial processes, ISPRS Journal of Photogrammetry and Remote Sensing 117 (2015) 206–216. doi:10.1016/j.isprsjprs.2016.01.017.

[6] J. Xue, G. Zhao, W. Xiao, Efficient GPU out-of-core visualization of large-scale CAD models with voxel representations, Advances in Engineering Software 99 (2016) 73–80. doi:10.1016/j.advengsoft.2016.05.006.

[7] H. Mitasova, R. S. Harmon, K. J. Weaver, N. J. Lyons, M. F. Overton, Scientific visualization of landscapes and landforms, Geomorphology 137 (1) (2012) 122–137. doi:10.1016/j.geomorph.2010.09.033.

[8] GRASS Development Team, Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.0, Open Source Geospatial Foundation (2016).
URL http://grass.osgeo.org

[9] Pitney Bowes Inc., Mapinfo Engage3D, http://www.pitneybowes.com/uk/location-intelligence/geographic-information-system/mapinfo-engage3d.html (1996–2016).

[10] B. Benes, R. Forsbach, Layered data representation for visual simulation of terrain erosion, in: Proceedings Spring Conference on Computer Graphics, 2001. doi:10.1109/SCCG.2001.945341.

[11] C. Tomlin, Geographic information systems and cartographic modeling, Prentice Hall series in geographic information science, Prentice Hall, 1990.

[12] K. Arroyo Ohori, H. Ledoux, J. Stoter, An evaluation and classification of n D topological data structures for the representation of objects in a higher-dimensional GIS, International Journal of Geographical Information Science 29 (5) (2015) 825–849. doi:10.1080/13658816.2014.999683.

[13] A. M. Lemon, N. L. Jones, Building solid models from boreholes and user-defined cross-sections, Computers and Geosciences 29 (5) (2003) 547–555. doi:10.1016/S0098-3004(03)00051-7.

[14] G. Caumon, P. Collon-Drouaillet, C. Le Carlier De Veslud, S. Viseur, J. Sausse, Surface-based 3D modeling of geological structures, Mathematical Geosciences 41 (8) (2009) 927–945. doi:10.1007/s11004-009-9244-2.

[15] G. Caumon, G. G. Gray, C. Antoine, M.-O. Titeux, 3D implicit stratigraphic model building from remote sensing data on tetrahedral meshes: theory and application to a regional model of La Popa Basin, NE Mexico, IEEE Transactions on Geoscience and Remote Sensing 51 (3) (2012) 1613–1621. doi:10.1109/TGRS.2012.2207727.

[16] A. C. de Oliveira Miranda, W. W. M. Lira, R. C. Marques, A. M. B. Pereira, J. B. Cavalcante-Neto, L. F. Martha, Finite element mesh generation for subsurface simulation models, Engineering with Computers (2014) 1–20doi:10.1007/s00366-014-0352-3.

[17] F. Penninga, P. J. M. Van Oosterom, A simplicial complex-based DBMS approach to 3D topographic data modelling, International Journal of Geographical Information Science 22 (7) (2008) 751–779. doi:10.1080/13658810701673535.
URL http://www.tandfonline.com/doi/abs/10.1080/13658810701673535{#}.VegiRaDtlBc

[18] B. Crespin, R. Bézin, X. Skapin, O. Terraz, P. Meseure, Generalized maps for erosion and sedimentation simulation, Computers & Graphics 45 (2014) 1–16. doi:10.1016/j.cag.2014.07.001.

[19] L. Wu, Topological relations embodied in a generalized tri-prism (GTP) model for a 3D geoscience modeling system, Computers & Geosciences 30 (4) (2004) 405–418. doi:10.1016/j.cageo.2003.06.005.

[20] M. D. Jones, M. Farley, J. Butler, M. Beardall, Directable weathering of concave rock using curvature estimation, IEEE Transactions on Visualization and Computer Graphics 16 (1) (2010) 81–94. doi:10.1109/TVCG.2009.39.

[21] F. Jørgensen, R. R. Møller, L. Nebel, N.-P. Jensen, A. V. Christiansen, P. B. E. Sandersen, A method for cognitive 3D geological voxel modelling of AEM data, Bulletin of Engineering Geology and the Environment 72 (3-4) (2013) 421–432. doi:10.1007/s10064-013-0487-2.

[22] C. Watson, J. Richardson, B. Wood, C. Jackson, A. Hughes, Improving geological and process model integration through TIN to 3D grid conversion, Computers & Geosciences 82 (2015) 45–54. doi:10.1016/j.cageo.2015.05.010.

[23] S. Forstmann, J. Ohya, Visualization of large iso-surfaces based on nested clip-boxes, in: ACM SIGGRAPH 2005 Posters, SIGGRAPH '05, ACM, New York, NY, USA, 2005. doi:10.1145/1186954.1187098.

[24] E. S. Lengyel, Voxel-Based Terrain for Real-Time Virtual Simulations, Ph.D. thesis, University of California Davis (2010).

[25] Ç. Koca, U. Güdükbay, A hybrid representation for modeling, interactive editing, and real-time visualization of terrains with volumetric features, International Journal of Geographical Information Science 28 (9) (2014) 1821–1847. doi:10.1080/13658816.2014.900560.

[26] M. Hadwiger, J. M. Kniss, C. Rezk-salama, D. Weiskopf, K. Engel, Real-time Volume Graphics, A. K. Peters, Ltd., 2006.

[27] D. Patel, Ø. Sture, H. Hauser, C. Giertsen, M. Eduard Gröller, Knowledge-assisted visualization of seismic data, Computers and Graphics (Pergamon) 33 (5) (2009) 585–596. doi:10.1016/j.cag.2009.06.005.

[28] Natali, T. G. Klausen, D. Patel, Sketch-based modelling and visualization of geological deposition, Computers and Geosciences 67 (2014) 40–48. doi:10.1016/j.cageo.2014.02.010.

[29] L. Wang, Y. Yu, K. Zhou, B. Guo, Multiscale vector volumes, ACM Transactions on Graphics 30 (6) (2011) 1. doi:10.1145/2070781.2024201.

[30] K. Takayama, O. Sorkine, A. Nealen, T. Igarashi, Volumetric modeling with diffusion surfaces, ACM Transactions on Graphics 29 (6) (2010) 1. doi:10.1145/1882261.1866202.

[31] M. Scholz, J. Bender, C. Dachsbacher, Real-time isosurface extraction with view-dependent level of detail and applications, Computer Graphics Forum 34 (1) (2015) 103–115. doi:10.1111/cgf.12462.

[32] S. Mantler, S. Jeschke, Interactive landscape visualization using GPU ray casting, Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia - GRAPHITE '06 (2006) 117doi:10.1145/1174429.1174448.

[33] L. Ammann, O. Génevaux, J.-M. Dischler, Hybrid rendering of dynamic heightfields using ray-casting and mesh rasterization, in: Proceedings of Graphics Interface 2010, GI '10, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2010, pp. 161–168.

[34] M. Treib, F. Reichl, S. Auer, R. Westermann, Interactive editing of GigaSample terrain fields, Computer Graphics Forum 31 (2) (2012) 383–392. doi:10.1111/j.1467-8659.2012.03017.x.

[35] M. Natali, E. Lidal, J. Parulek, Modeling terrains and subsurface geology, in: Eurographics 2013-State of the Art Reports, 2012, pp. 155–173. doi:10.2312/conf/EG2013/stars/155-173.

[36] A. Peytavie, E. Galin, J. Grosjean, S. Merillou, Arches: A framework for modeling complex terrains, Computer Graphics Forum 28 (2) (2009) 457–467. doi:10.1111/j.1467-8659.2009.01385.x.

[37] F. Löffler, A. Müller, H. Schumann, Real-time Rendering of Stack-based Terrains, in: P. Eisert, J. Hornegger, K. Polthier (Eds.), Vision, Modeling, and Visualization (2011), The Eurographics Association, 2011. doi:10.2312/PE/VMV/VMV11/161-168.

[38] A. K. Turner, Challenges and trends for geological modelling and visualisation, Bulletin of Engineering Geology and the Environment 65 (2) (2006) 109–127. doi:10.1007/s10064-005-0015-0.

[39] J. L. Gunnink, D. Maljers, S. F. Van Gessel, A. Menkovic, H. J. Hummelman, Digital Geological Model (DGM): A 3D raster model of the subsurface of the Netherlands, Geologie en Mijnbouw/Netherlands Journal of Geosciences 92 (1) (2013) 33–46. doi:10.1017/S0016774600000263.

[40] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, in: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '87, ACM, New York, NY, USA, 1987, pp. 163–169. doi:10.1145/37401.37422.

[41] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, M. Gross, Real-time ray-casting and advanced shading of discrete isosurfaces, Computer Graphics Forum 24 (3) (2005) 303–312. doi:10.1111/j.1467-8659.2005.00855.x.

[42] A. Knoll, S. Thelen, I. Wald, C. D. Hansen, H. Hagen, M. E. Papka, Full-resolution interactive cpu volume rendering with coherent bvh traversal, in: 2011 IEEE Pacific Visualization Symposium, 2011, pp. 3–10. doi:10.1109/PACIFICVIS.2011.5742355.

[43] R. Yagel, D. Cohen, A. Kaufman, Normal estimation in 3 D discrete space, The Visual Computer 8 (5-6) (1992) 278–291. doi:10.1007/BF01897115.

[44] A. Kadosh, D. Cohen-Or, R. Yagel, Tricubic Interpolation of Discrete Surfaces for Binary Volumes, IEEE Transactions on Visualization and Computer Graphics 9 (4) (2003) 580–586. doi:10.1109/TVCG.2003.1260750.

[45] C. Sigg, T. Weyrich, M. Botsch, M. Gross, GPU-Based Ray-Casting of Quadratic Surfaces, Symposium on Point-Based Graphics (2006) 59–65doi:10.2312/SPBG/SPBG06/059-065.

[46] J. Baert, A. Lagae, P. Dutré, Out-of-Core Construction of Sparse Voxel Octrees, Computer Graphics Forumdoi:10.1111/cgf.12345.

[47] B. Dado, T. R. Kol, P. Bauszat, J. M. Thiery, E. Eisemann, Geometry and attribute compression for voxel scenes, Computer Graphics Forum 35 (2) (2016) 397–407. doi:10.1111/cgf.12841.

[48] T. Akenine-Möller, E. Haines, N. Hoffman, Real-time rendering, A. K. Peters, Ltd., 2009.

[49] J. Wang, F. Yang, Y. Cao, Cache-aware sampling strategies for texture-based ray casting on gpu, in: Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on, 2014, pp. 19–26. doi:10.1109/LDAV.2014.7013200.

[50] D. Jönsson, P. Ganestam, A. Ynnerman, M. Doggett, T. Ropinski, Explicit Cache Management for Volume Ray-Casting on Parallel Architectures, in: EG Symposium on Parallel Graphics and Visualization (EGPGV), Eurographics, 2012, pp. 31–40.

[51] S. P. Dunstan, A. J. B. Mill, Spatial indexing of geological models using linear octrees, Computers and Geosciences 15 (8) (1989) 1291–1301. doi:10.1016/0098-3004(89)90093-9.

[52] K. Weiss, L. De Floriani, R. Fellegara, M. Velloso, The PR-star octree: a spatio-topological data structure for tetrahedral meshes, in: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '11, ACM Press, New York, New York, USA, 2011, p. 92. doi:10.1145/2093973.2093987.

[53] G. Jansen, R. Sohrabi, S. A. Miller, HULK – Simple and fast generation of structured hexahedral meshes for improved subsurface simulations, Computers & Geosciences 99 (July 2016) (2017) 159–170. doi:10.1016/j.cageo.2016.11.011.

[54] B. Liu, G. J. Clapworthy, F. Dong, IsoBAS: A binary accelerating structure for fast isosurface rendering on GPUs, Computers and Graphics (Pergamon) 48 (2015) 60–70. doi:10.1016/j.cag.2015.02.002.

[55] M. Labsch, M. Hadwiger, P. Rautek, S. Bruckner, M. E. Gr, JiTTree : A Just-in-Time Compiled Sparse GPU Volume Data Structure, IEEE Transactions on Visualization and Computer Graphics 22 (1) (2016) 1025–1034. doi:10.1109/TVCG.2015.2467331.

[56] C. Crassin, F. Neyret, S. Lefebvre, E. Eisemann, I. Sophia-antipolis, GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering, ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 1 (212) (2009) 15–22. doi:10.1145/1507149.1507152.

[57] O. Št'Ava, B. Beneš, M. Brisbin, J. Křivánek, Interactive terrain modeling using hydraulic erosion, EuroGraphics Symposium on Computer Animation (2008) 201–210doi:10.2312/SCA/SCA08/201-210.

[58] T. Harada, Sliced grid: A memory and computationally efficient data structure for particle-based simulation on the gpu, in: W. Engel (Ed.), ShaderX7: Advanced Rendering Techniques, Charles River Media, 2009, pp. 685–698.