VolumeEVM: A New Surface/volume integrated model

Jorge Rodríguez[*], Dolors Ayala,
jrodri@lsi.upc.es, dolorsa@lsi.upc.es

Abstract

Volume visualization is a very active research area in the field of scientific visualization. The Extreme Vertices Model (EVM) has proven to be a complete intermediate model to visualize and manipulate volume data using a surface rendering approach. However, the ability to integrate the advantages of surface rendering approach with the superiority in visual exploration of the volume rendering would actually produce a very complete visualization and edition system for volume data. Therefore, we decided to define an enhanced EVM-based model which incorporates the volumetric information required to achieved a nearly direct volume visualization technique. Thus, VolumeEVM was designed maintaining the same EVM-based data structure plus a sorted list of density values corresponding to the EVM-based VoI's interior voxels. A function which relates interior voxels of the EVM with the set of densities was mandatory to be defined. This report presents the definition of this new surface/volume integrated model based on the well known EVM encoding and propose implementations of the main software-based direct volume rendering techniques through the proposed model. *keywords:*

1 Introduction

Direct volume rendering techniques (DVR), also called voxel-based rendering by [3], create images of the VoI, directly from the volume data, without any intermediate surface representations, so any mapping strategy is required. The first algorithm based on this approach from gray level volumes was presented by Lenz et al. in [9]. In this approach, the volume is considered as translucent gel material and the image is obtained assigning color and opacity to sampled values and applying a model to determine how the volume reflects, scatters or occludes light passing through it.

Compared to surface rendering techniques, the major advantage is that all gray level information which has originally been acquired is kept during the

[*]This author has been supported by a grant from FONACIT (Venezuelan Council of Research in Science and Technology)

rendering process, becoming in an ideal tool for interactive data exploration [6]. Threshold values and other parameters which are not clear from the beginning can be changed interactively. Furthermore, DVR allows a combined display of different aspects such as opaque and semitransparent surfaces, cuts and maximum intensity projections.

A current drawback of DVR techniques is that large amount of data which has to be handled does not allow real time applications for visualization. Also, manipulating and analysis processes demand polygonal representation of the object to perform these tasks in an intuitive and friendly way. Futhermore, a complete framework for visualization and manipulation of volume data demands the integration in the system of some non-volumetric objects, such as virtual tools commonly designed by CAD.

In order to profit the superiority in visualization of the DVR techniques and the versatility for interaction and manipulation tasks of the surface rendering approach, a surface/volume integrated model must be defined. Thus, in order to profit the potential of the EVM as an intermediate model for surface rendering, an enhanced version has been created, incorporating the volumetric information required to achieve a cogent direct volume visualization. This report present this integrated model and discuss the implementation of the three main direct volume visualization strategies, i.e. splatting, shear-warp factorization and ray casting, using the proposed EVM-based integrated model.

2 Background

This section gives the reader a brief state of the art in the area of direct volume rendering and integrated model for volume visualization.

2.1 Common Optical Model

The optical model to describe the behavior of the light passing through the volume data is typically defined by the *low-albedo volume rendering integral* [1], [5], [7], [11], which analitically computes $I_\lambda(x, r)$, the amount of light of wavelength λ coming from ray direction r that is received at location x on the image plane:

$$I_\lambda(x, r) = \int_0^L C_\lambda(s)\mu(s)e^{(-\int_0^s \mu(t)dt)}ds \tag{1}$$

Where, L is the length of the ray r. If we think of the volume as being composed of particles with certain densities μ, then these particles receive light from μ all surrounding light sources and reflects this light towards the observer according to their specular and diffuse material properties. In addition, the particles may also emit light on their own. Thus, in Eq. 1, C_λ is the light of wavelength λ reflected and/or emitted at location s in the direction of r. To account for the higher reflectance of particles with larger densities, we must weigh the reflected color by the particles density. The light scattered at s is then attempted by the densities of the particles between s and the eye according to the exponential attenuation function.

At least, in general case, the Low-Albedo integral cannot be computed analytically [11]. Hence, practical volume rendering algorithms discretize the integral into a series os sequential intervals i of width Δs:

$$I_\lambda(x,r) = \sum_{i=0}^{Ł/\Delta s} C_\lambda(s_i)\mu(s_i)\Delta s \cdot \prod_{j=0}^{i-1} e^{(-\mu(s_j)\Delta s)} \tag{2}$$

Using a Taylor series approximation of the exponential term and dropping all but the first two terms, we get the familiar compositing equation [10].

$$I_\lambda(x,r) = \sum_{i=0}^{Ł/\Delta s} C_\lambda(s_i)\alpha(s_i) \cdot \prod_{j=0}^{i-1}(1-\alpha(s_j)) \tag{3}$$

This expression is denoted as *discretized volume rendering integral* [13], where opacity $\alpha = 0$ means transparency. Expression 3 represents a common theoretical framework for all volume rendering algorithms.

In practice, this computation is done for *R, G, B* separately. As it is a sum of intensities of individual samples, each intensity is attenuated by the product of transparencies accumulated as the light passes from sample to observer. The computation can be done recursively by processing one sample at a time, accumulating color and opacity separately:

$$C_{out} = C_{in} + (1-\alpha_{in})\alpha_i C_i \tag{4}$$

$$\alpha_{out} = \alpha_{in} + (1-\alpha_{in})\alpha_i C_i \tag{5}$$

This is a *front-to-back ordering* (FTB), which can be reversed to work *back-to-front* (BTF), in which case only the color needs to be accumulated:

$$C_{out} = C_i\alpha_i + C_{in}(1-\alpha_i) \tag{6}$$

We want to point out that compositing steps are associative, thus group of samples can be composited and then composite the groups, as long as the order is maintained. However, the compositing is not commutative which means that the order of compositing is important.

2.2 Direct volume rendering approaches

A wide variety of DVR implementations have been reported in the literature. Regardless all these variations, four approaches can be identified. The classical approaches for DVR can be classified into two categories: The *image-order approach*, also called *backward rendering*, such as *Raycasting* and the *object-order approach*, also called *forward rendering*, such as *splatting*. Alternatively, other two different approaches have been proposed, in order to accelerate the performance of the visualization process, which can not be exactly classified in any of the previous categories: The *shear-warp factorization* and *3D texture-mapping hardware*. A brief description of these four strategies will be presented in the following:

- *Image-order approach*:

  Image-order algorithms operate by casting rays from each image pixel and and resampling it along the casted ray at the sampled points, hence the name of Raycasting for the excellence technique of this approach [10] (see figure 1) . The composition of colors and opacities along the ray can be performed BTF or FTB order. This approach has as main drawback that the spatial data structure must be traversed once for every ray, resulting in redundant computation because the ray caster spend more time calculating the location of sample points and performing arithmetic than other classes of volume rendering algorithms.

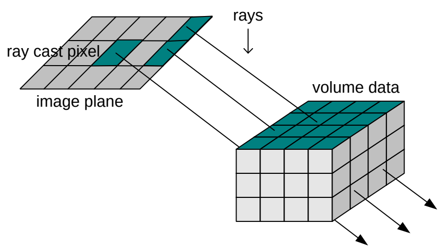


Figure 1: image-order approach

  Acceleration techniques

  A commonly used acceleration technique for ray casters is *early ray termination* by neglecting those objects whose reflected light does not reach the viewer, but it only can be used if FTB composition order is performed. Other authors prefer the space leaping. This strategy is based on the physical fact that reflection occurs only where material parameters change, so homogeneous regions have zero gradients and not reflect any light. Therefore, they can be skipped during rendering. *Space leaping* can be performed by several methods, distance coding or proximity clouds [20], [16], spatial data structures exploiting data coherence such as octrees [12] and bounding boxes. Several acceleration strategies by parallelization have also been reported.

- *Object-order approach*:

  Object-order algorithms operate by splatting voxels into the image, hence the name of Splatting [19] for the most known algorithm in this approach (see figure 2). Instead the previous approach, these algorithms access the volume volume data in storage order, so each voxel is visited only once avoiding redundant computation. The splatters operate by iterating over the voxels computing, for each one, its contribution to the image by convolving the voxel with a filter that distributes the voxel's value to a neighborhood of pixels. Unfortunately, computing accurately the resampling filter that calculates the contribution of a given voxel to the image is expensive, because it is highly view dependant. The method can be adapted for either fast execution or high image quality, but not both simultaneously, and it is difficult to achieve a fast software implementation that does not compromise too much on image quality. In counterpart, the object oriented approach permits efficient usage of volume memory (efficient cache usage).

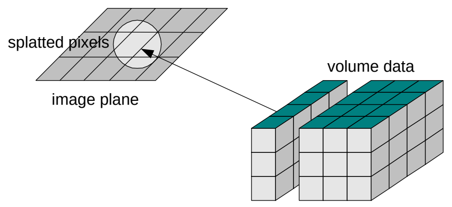


Figure 2: object-order approach

Cell projection algorithms is another object-order technique, but in this case the projection is performed by using polygon scan conversion. These techniques are often used for irregular volume data, so it is out of the scope of this thesis.

Acceleration techniques

Object-order approach approach permits gains of speed by taking advantage of coherence in the volume. This is typically achieved by using spatial data structures that encode the location of high-opacity voxels. Structures used for this purpose include octrees and pyramids. Also, run-length encoding can be profited instead the previous approach where the usage of this encoding is prohibitive.

- *Shear-Warp factorization*:

  This algorithm, proposed by [8], tends to combine the advantages of image-order and object-order approaches. The method is based on a factorization of the viewing matrix into a 3D shear parallel to the slices of the volume data, a projection to form a distorted intermediate image, and finally a 2D warp to produce the final image (see figure 3). This simple factorization allows to construct an object-order algorithm with the same kind of advantages as an image-order algorithm. In fact, it has been recognized as the fastest software renderer to date.

  What makes it possible is transforming the volume to an intermediate coordinate system for which there is a very simple mapping from the object

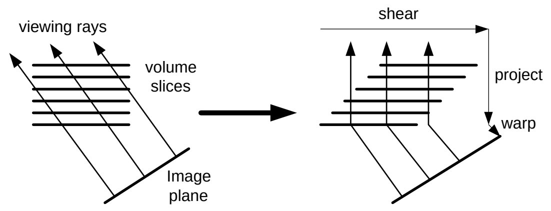


Figure 3: Shear-warp factorization

coordinate system and which allows efficient projection to 2D image. This intermediate coordinate system is also called *sheared object space.*

The factorization of the view transformation matrix, M_{view}, can be written as follows:

$$M_{view} = M_{warp2D} \cdot M_{shear3D} \cdot P \tag{7}$$

where P is a permutation matrix which transposes the coordinate system in order to make the z-axis the principal viewing axis. $M_{shear3D}$ transforms the object into sheared object space, and M_{warp2D} transforms sheared object coordinates into image coordinates. For a general parallel projection $M_{shear3D}$ has the form of a shear perpendicular to the z-axis:

$$M_{shear3D} = \begin{pmatrix} 1 & 0 & S_x & 0 \\ 0 & 1 & S_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad or \quad M_{shear3D} = \begin{pmatrix} 1 & 0 & S_x & t_x \\ 0 & 1 & S_y & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

Where s_x and S_y (as well as t_x and t_y) can be computed from the elements of M_{view}. The final term of the factorization is a matrix which warps sheared object space into image space:

$$M_{warp2D} = M_{view} \cdot P^{-1} \cdot M_{shear3D}^{-1} \tag{9}$$

The projection in sheared object space has several geometric properties that simplify the compositing step of the algorithm:

– Scanlines of pixels in the intermediate image are parallel to scanlines of voxels in the volume data.

– All voxels in a given voxel slice are scaled by the same factor (interesting for perspective projection which is not covered here)

  - Every voxel slice has the same scale factor when projected into the intermediate image, and this factor can be chosen arbitrarily. In particular, a unity scale factor can be chosen so that for a given voxel scanline there is a one-to-one mapping between voxels and intermediate-image pixels.

  A useful implication of these properties is that every voxel in a given slice of volume has the same resampling weights. Each slice voxels is simply translated, so one set of resampling weights can be precomputed and then reused for every voxel. This fact eliminates the problems associated with efficient resampling in object-order volume rendering algorithms. One of the aims is to maximize the benefit of both spatial data structures and early ray termination.

  Acceleration techniques

  This algorithm is an acceleration technique by itself. However, special list-based data structures based on run-length encoding can further improve the performance of the algorithm. The idea is to skip transparent voxels in object space (space leaping) and non-visible lines in the pixel scanlines (early ray termination). Besides of the run-length encoding that allows to skip empty regions (*voxel transparency encoding*), a list-based encoding has been chosen for runs of opaque pixels of the intermediate images (*pixel opaqueness encoding*). A pixel is called opaque if the intensity of the corresponding viewing ray is below an user-defined threshold. For each opaque pixel a pointer is given that directs to the next non-opaque pixel. Both data structures are used in combination to implement space leaping and early ray termination.

  Run-length encoding forces the algorithm to proceed along predefined lines in the voxel volume. This processing fits ideally into the shear-warp paradigm since it traverse the slices of the volume in a front-to-rear order relative to the viewer position. Moreover it processes each slice scanline by scanline. However, a problem occurs if the viewing direction changes so that a different face of the data set cube is used for shear-warp. In this case, a second run-length encoded data set must be available. In total there are three such compressed data sets, one for each main viewing direction (x, y, z). As acknowledged, this requires threefold overhead.

  Initially, the intermediate projection plane is defined from the viewing direction, i.e by selecting the most parallel face of the data set cube. This determines the corresponding run-length encoded data set. Next, the algorithm processes the voxel transparency and data run-length encoded and image on two voxel lines in parallel. It first jumps to the non-transparent voxels. If the corresponding pixel is opaque it continues to jump over empty voxels and opaques pixels until non-transparent voxels and non-opaque pixels are found. See figure 4

- *3D texture-mapping hardware*:

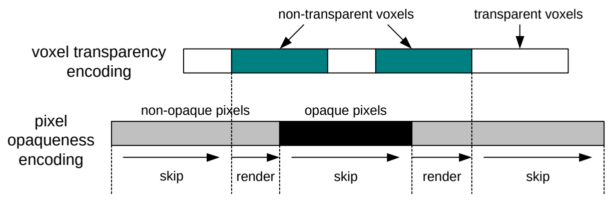


Figure 4: Integration of space leaping and early ray termination in shear-warp algorithm

The 3D texture-mapping approach, popularized by [2], is a direct volume visualization technique, using 3D textured data slices, combined using a blending operator. Unlike ray casting, where each image pixel is built up ray by ray, this approach takes advantage of spatial coherence. The 3D texture is used as a voxel cache, processing all rays simultaneously, one 2D layer at a time. Texture mapping and compositing operations are performed by hardware very quickly. Therefore, the rendering time is negligible if compared to other software-based approaches.

In this approach, the technique for visualizing volume data is composed of two steps. First the volume data set, represented by the three-dimensional texture block, is loaded in the texture buffer of the graphics subsytem. Then, texture data is sampled with planes parallel to the image plane and stacked along the direction of view. These planes are rendered as polygons, clipped to the limits of the texture volume. These clipped polygons are textured with the volume data, and the resulting images are blended together, from back to front, towards the viewing position. As each polygon is rendered, its pixel values are blended into the frame buffer to provide the appropriate transparency effect. Commonly, the representation of an unit of volume information in texture memory is called a *texel* (texture element).

There a number of common blending functions used in volume visualization:

– *The over operator* [15] is the most common way to blend for volume visualization. Volumes blended with the over operator approximate the flow of light through a colored, translucent material. The translucency of each point in the material is determined by the value of the texel's alpha channel. Texels with higher alpha values tend to obscure texels behind them, and stand out through the obscuring texels in front of them.

– *The attenuate operator* simulates an X-ray of the material. With attenuate, the texel's alpha appears to attenuate light shining through the material along the view direction towards the viewer. The texel alpha channel models material density. The final brightness at each pixel is attenuated by the total texel density along the direction of view

– *The MIP operator*, by Maximum Intensity Projection, is used in medical imaging to visualize blood flow. MIP finds the brightest texel alpha from all the texture slices at each pixel location. MIP is a contrast enhancing operator; structures with higher alpha values tend to stand out against the surrounding data.
– *The under operator* permits FTB rendering of volume slices giving the same result as the over operator blending slices from back to front.

Acceleration techniques

3D texture-mapping hardware approach is the fastest among all the volume visualization processes. The incorporation of the graphics hardware capabilities can be considered as an acceleration technique by itself. However, as in the previous approaches, data complexity can be controlled by the using of hierarchical data structures, such as octree [17], [18], in order to skip empty regions of the volume or to ensure that only non-void regions will be loaded in texture memory.

2.3 Surface and Volume data integration

As we have pointed out above both surface and direct volume rendering are widely used in the field of volume visualization and according to the particular requirement each one has advantages respect to the other. Also, a complete framework for visualization and manipulation of volume data demands the integration in the system of some non-volumetric objects, such virtual tools commonly designed by CAD.

In order to profit the superiority in visualization of the DVR techniques and the versatility for interaction and manipulation tasks of the surface rendering approach, a surface/volume integrated model must be defined.

Two approaches of integrated models can be identified [14]: *Data conversion approach* and *independent representation approach.*

2.3.1 Data conversion approach

This approach is based on converting polygonal model and volume data into a common codification scheme, so data conversion techniques have to be applied to reduce either surface to volume data through scan-conversion techniques or volume data to surface representation through a mapping strategy to extract the isosurface fitted into the volume (see figure 5).

Data conversion has the advantage that specialized rendering algorithms are not required because all the data is reduced to a well known representation (surface- or volume-based). However, to avoid sacrificing one of the rendering approaches both surface and volume representation must be maintained and the conversion techniques have to allow passing from each representation model to the another one, so the model will be expensive in storage and speed.

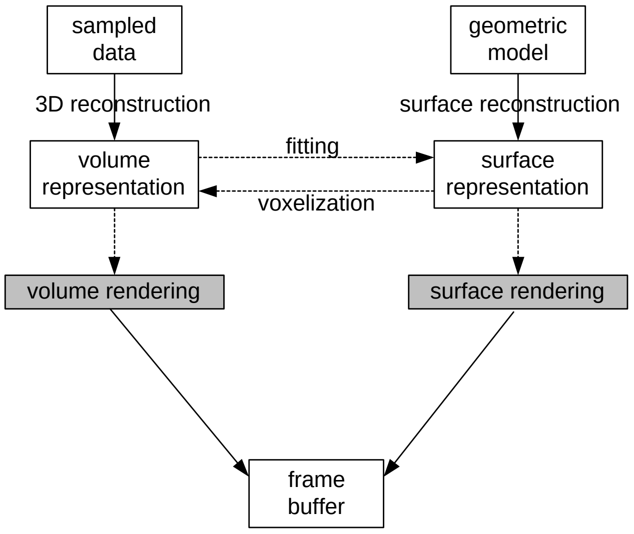


Figure 5: Surface/voume integrated model by data conversion approach

2.3.2 Independent representation approach

This approach preserves surface and volume data in their respective representation schemes. Data rendering is performed independently by two separated rendering processes. One executes a conventional volume rendering technique from the volume data set and the other performs a conventional surface rendering technique on the geometric model. Obviously, the combination of both approaches is achieved only in the final image (see 6).

Independent representation has the drawback of the visual integration of the objects in the final image demands hybrids algorithms to resolve transparencies and intersections between objects from different models.

3 A new surface/volume integrated model

This section defines a new surface/volume integrated model based on the EVM. For this purpose, a particular variation of the data conversion approach (see section 2.3) has been designed using an EVM-based hybrid model as a common codification scheme (see fig 7).

According to the proposed framework, the *VolumeEVM* is defined as the EVM-based hybrid model and two rendering algorithms from this model have to be created: *The EVM-based volume rendering*and *the EVM-based surface*

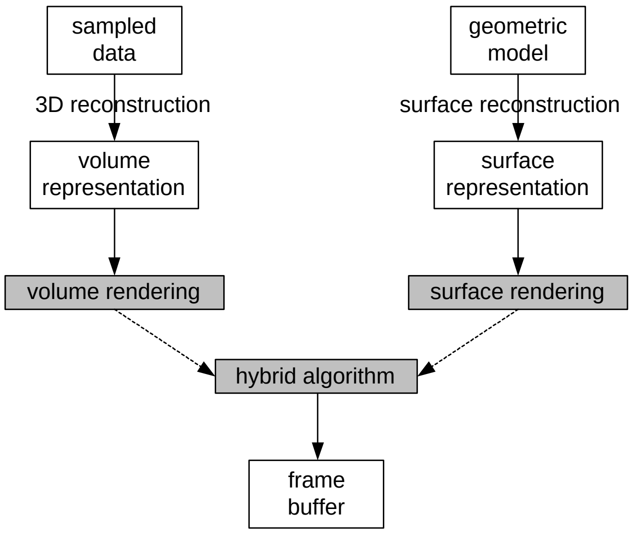


Figure 6: Surface/voume integrated model by independent representation approach

rendering. Thus, the input sampled data is directly formatted to a standard volume representation, but the input geometric model must be voxelized in order to obtain its binary volume representation before any processing. Once the input data has been converted to the standard volume representation, the VolumeEVM for each input data set is computed. When the VolumeEVM of the VoI has been built one of the two rendering strategies is performed depending on whether the VoI comes from binary or grey-level volume representation.

3.1 VolumeEVM: a new hybrid model

As a common codification scheme to unify the representation of inputs coming from both sampled data and geometric models, we have defined the *VolumeEVM*, a particular hybrid model based on the EVM encoding which incorporates the information about the interior voxels of the extracted VoI for a given segmentation threshold. When the input data is depicted by a geometric model a binary voxelization is performed, so the interior information is irrelevant and all this voxels are set as a maximum value of grey-level. Typically, this objects would be rendered by the EVM-based surface rendering algorithm, though the EVM-based volume rendering process may be achieved with any inconvenient.

To define the VolumeEVM model, the original EVM data structure, reported in previous work [4] for surface rendering, has been complemented with two new elements: a *densities table (DT)* and a *relation (v)* which assigns to each interior voxel of the EVM its corresponding density value from the table. The densities table contains the grey-level value for all the voxels which lie in the interior

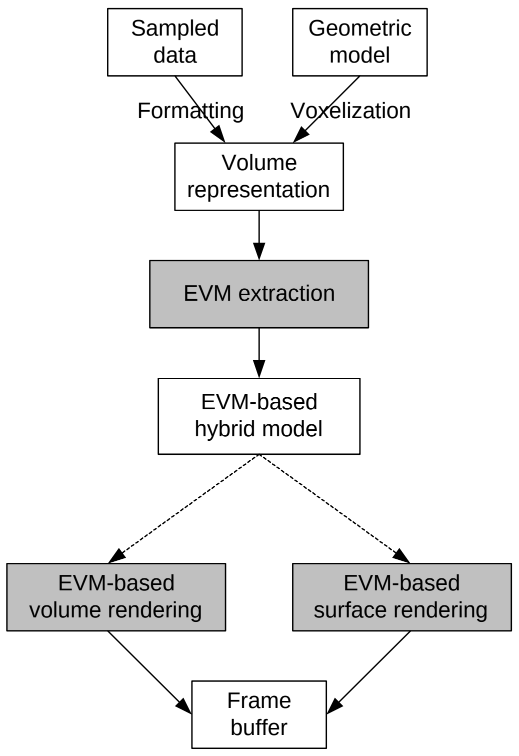
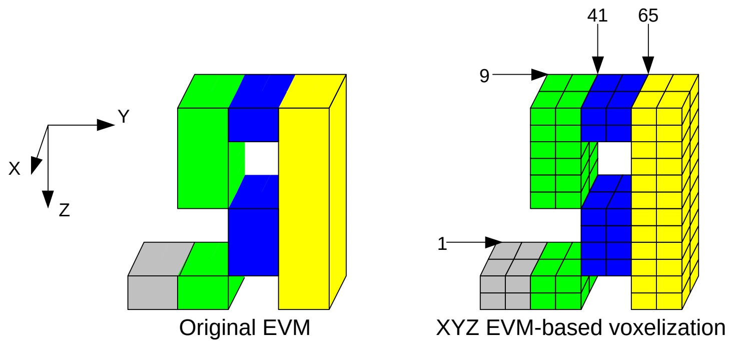


Figure 7: Dataflow of the our surface/volume integrated approach

of the orthogonal polyhedron corresponding to the VoI computed for a certain given threshold. The relation $v : EVM \rightarrow Z$, was defined as the axis-ordered voxelization of the EVM's interior (see figure 8), so each discrete unit in the interior of EVM is numbered through v.

v(x,y,z)=k, DT[k]=d

It is interesting to note that only the relevant voxels for visualization are stored, the volumetric information of the voxels discarded for the VoI's threshold is not encoded in the VolumeEVM. A densities table and the corresponding voxelization relationship encode the volumetric information of the EVM-represented VoI in only one traverse direction, let's say XYZ traversal order. However, object-order visualization algorithms typically require to traverse the voxelized object in all possible axes order, so seven tables more (for a total of eight) must be stored, according to the eight octants where the viewpoint can be located. Actually, these eight tables are resumed in just four, because the other ones can be obtained as the complementary cases by traversing each table in reverse sense. Obviously, we are storing redundant densities but in general this redundancy produces less values than the total density values of the com-



Densities table in XYZ order

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	...	112

Figure 8: VolumeEVM data structure. Original EVM with colored slices. Densities table is the bag of all density values of the discretized interior of the VoI. The relation $v : EVM \rightarrow DT$ permits to assign a cell of DT per voxel of the VoI's interior respect to certain traverse sorting. At the figure, a XYZ-table has been built.

plete volume data set. Furthermore, in other compressed alternative schemes for direct volume rendering, such as run-length encoding of volume data, this redundancy is mandatory.

Filling densities tables is carried out during the EVM extraction process. Along of marching process over the voxels set to identify and collect the extreme vertices, the densities which hold the given threshold are collected too. Marching has to be repeated three times more, one per axis direction of traversal, in order to achieve the four sorted tables of densities.

Once defined the EVM-based hybrid model as the unified representation of the objects, the rendering technique from this new model has to be defined. VolumeEVM is actually a dual representation scheme. When the interior of the VoI is assumed as homogeneous, this information is obviated and the model becomes exactly the EVM-based surface rendering algorithm [4]. On the other hand, when the interior of the VoI is relevant for the visualization, a direct volume rendering technique has to be used, taking into account all the density information contained in the model. In that case, an *EVM-based volume rendering technique* must be applied.

In the following, the main direct volume rendering techniques explained in section 2.2, will be discussed and adapted using the VolumeEVM encoding to define a EVM-based volume rendering technique per DVR approach.

3.2 Splatting using VolumeEVM

Implementation of splatting is straightforward using VolumeEVM because this DVR object-order approach performs the visualization by marching over the voxels computing, for each one, its contribution to the image. As VolumeEVM encodes the densities values in a coordinate axis-based sorting, accessing to the densities table can be achieved linearly while the interior voxels are being visited. Thus, VolumeEVM guarantees the direct access of densities values during the splatting process. As the proper composition order depends on the octant where the viewpoint is located, a sorted table of densities is required per viewpoint location, eight in total resumed in four tables. If the order among the axes is considered 48 different tables must be built. In our implementation the axes order is ignored.

3.2.1 Advantages

The EVM-based splatting has the advantage that only interior voxels are considered. It would not be an exaggeration to say that the space leaping is encoded implicitly in the EVM encoding, so any extra auxiliary data structure or additional storage overhead is required to skip all the irrelevant voxels. In addition, as well as the conventional splatting, early ray termination can be implemented when the contributions to a certain pixel on the image plane is negligible, composing only for the non-saturated pixels.

3.3 Shear-warp factorization using VolumeEVM

Shear-warp factorization using VolumeEVM is a bit more interesting problem. As it has been said in section 2, this method perform the visualization process in two steps: shearing + warping. Once the sheared objects has been projected, the work in 3D has finished. Thus, the VolumeEVM has a relevant paper only during the shearing because the warping is performed over a 2D image and is always the same.

3.3.1 Shearing the EVM

In this case, a *sheared EVM* must be obtained from the original one through the *shear* operation. The shearing works by displacing set of voxels slice by slice, so the EVM structure must be filleted according to the voxelization destructing the original EVM. Therefore, sheared EVM can not be the result of any direct transformation from the original one. Nevertheless, as the sheared EVM is obtained slicing, by orthogonal planes, the region between two adjacent planes of vertices (also called *slice* in the EVM notation), each fillet produces a one voxel-width EVM which is completely defined by the corresponding section (see the figure 9).

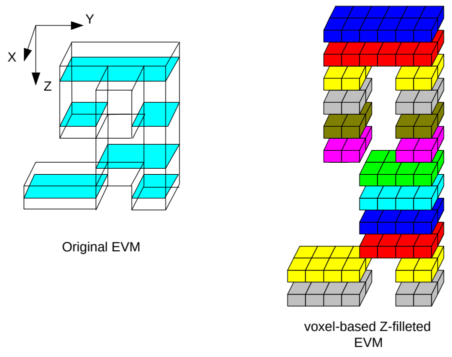


Figure 9: Original EVM with sections. Decomposition of EVM in multiples one voxel-width EVM along Z-axis.

Once the EVM has been filleted and multiples EVMs have been created, the corresponding displacements take place. Each one voxel-width EVM is displaced independently according to the shearing transformation parameters. The shearing operations will be completed when the union among all the displaced EVMs has been finished, so a sheared EVM has been created. Nevertheless, it will be interesting to see what happens with the densities table, because to complete the shearing of the VolumeEVM the corresponding shearing of the table must be guaranteed. Fortunately, 3D shearing performs displacements only respect two coordinate axes, so one coordinate axis remains unchanged (Z-axis in figure 10). Thus, ZXY- and ZYX-table have encoded the densities values by slicing the object respect to the Z-axis, so a Z-based shearing does not affect the order of the densities on the table. In summary, choosing the proper densities table the sheared EVM will have just the same table that the original one.

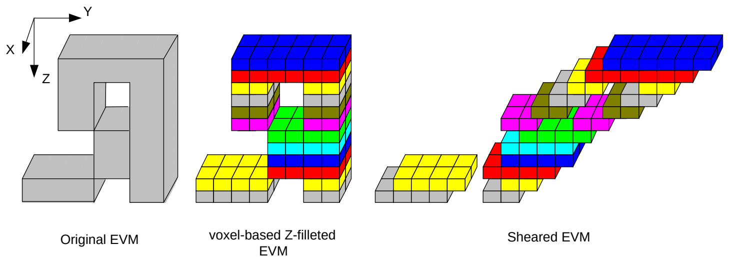


Figure 10: Original EVM . EVM filleted along Z-axis. Y-Sheared EVM. Original EVM and the sheared one have the same densities table.

3.3.2 Projection and warping

Once the sheared EVM and the corresponding table has been created, the projection is as easy as to perform an axis-aligned splatting of the sheared EVM. That means, to traverse the object by Z-slices and splats the voxels against the image plane computing the corresponding composition on the overlapped pixels. After the projection has concluded, a visualization of the sheared object (a distorted image of the original one) has been achieved, so a 2D warping is required to bring the correct appearance back. The last pass is a 2D operation performed directly over the image.

3.3.3 Advantages

Again, like in the previous method, an EVM-based implicit space leaping scheme is profited and only interior voxels are considered. The empty (or irrelevant) voxels are ignored during the shearing and projection processes saving unnecessary processing time. Also, the shearing is achieved by moving only the extreme vertices of the sections, it does not use a voxel-based approach for displacements as the conventional shearing, so to move a chain of many voxels, only the first and final elements are considered.

3.4 Ray casting using VolumeEVM

Now, we are going to discuss the VolumeEVM-based ray casting which has proven to be the less efficient EVM-based DVR. Regardless the poor efficiency achieved, it is interesting to point out that the visualization using this approach is possible. Furthermore, the EVM would be an very useful auxiliary structure to control de skips along the ray traversal.

The main drawback is the accessing to the densities values. While in an object-order approach a sorting of the table coinciding with the marching order is always possible, in the ray casting method each ray demands an specific order depending on its traversal, so direct accessing is not possible and navigation along the EVM is required to find the index of the corresponding density value per visited voxel along the ray traversal (see figure 11).

Searching can be accelerated using the geometry knowledge of the EVM. Partitioning the object by sections or by boxes (OUoDB) and storing the densities according to this partitioning, a linear search on the table is required only when the corresponding region of the voxels has been reached, so the searching space has been reduced dramatically. However, even in that case, a navigation along the structure is required per voxel and it is a very time consuming process in comparison with the conventional approach where the complete voxelization is always loaded in memory and direct access is guaranteed.

The conclusion to which this analysis leads is that EVM would be an excellent auxiliary structure to skip irrelevant information when the complete voxelization is maintained. Therefore, the EVM encoding would be a better acceleration technique for ray casting than distance transform, because the pre-

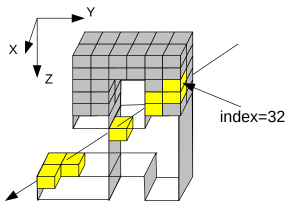


Figure 11: Ray casting approach. The marked voxels are the visited ones by the ray. Finding the corresponding density value requires to determine the position of the voxel in the interior of the EVM. At the figure, index=32 for a ZXY-order.

processing to build the auxiliary information is less expensive and the memory requirements are dramatically smaller.

4 Conclusions

VolumeEVM, a new surface/volume integrated model has been proposed using the EVM encoding. In recent reports, the EVM has proved to be an excellent intermediate model to perform surface rendering and manipulation of volume data. Now, we have incorporated volumetric information to the EVM data structure in order to achieve direct volume visualization from the EVM-based model maintaining the ability to carry out surface visualization and manipulation of volume data. The object-order methods, such as splatting and even shear-warp factorization, can be performed efficiently using the VolumeEVM. Furthermore, some advantages are introduced respect the conventional implementations, due to the implicit space leaping scheme of the EVM encoding. In contrast, the image-order method (ray casting) has resulted in a very time consuming process. However, using it as an acceleration technique of conventional ray casting becomes in an excellent auxiliary encoding, because both the pre-process to build it and the memory requirements are dramatically more less than other strategies such as distance transform technique.

References

[1] J.F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH'82*, pages 21–29, 1982.

[2] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *ACM Symposium on Volume Visualization, Washington, D.C.*, pages 91–98, 1994.

[3] K.H. Höhne and A. Pommert. *Mrain Mapping: The methods.* Academic Press, Inc., 1996.

[4] J.Rodrguez, D.Ayala, and A.Aguilera. Representation and boundary extraction of a 3D digital image using the evm model. Technical Report LSI-00-67-R, Universitat Politecnica de Catalunya, 2000.

[5] J.T. Kajiya and B.P. Von Herzen. Ray tracing volume densities. In *Proc. SIGGRAPH'84*, pages 165–174, 1984.

[6] A. Pommert M. Reimer U. Tiede K.H. Höhne, M. Bomans and G. Wiebecke. 3D visualization of tomographic volume data using the generalized voxel-method. *Visual Computing*, 6:28–36, 1990.

[7] W. krueger. The application of transport theory to the visualization of 3d scalar fields. *Computer in Physics*, 5:397–406, 1991.

[8] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization. *Proceedings SIGGRAPH'94*, pages 451 – 457, 1994.

[9] R. Lenz, P.E. Danielson, S. Cronströn, and G. Gudmundsson. Presentation and perception of 3D images. *Pictorial Information Systems in Medicine*, pages 459–468, 1986. NATO ASI Series F19, Springer-Verlag, Berlin.

[10] M. Levoy. Display of surfaces from volume data. *IEEE Computer graphics and applications*, 8(3):29 – 37, 1988.

[11] N. Max. Optical models for direct volume rendering. *IEEE Trans. Vis. and Comp. Graph.*, 1(2):99–108, 1995.

[12] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19:129–147, 1982.

[13] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *Volume Visualization 2000*, pages 81–90, 2000.

[14] I. Boada Oliveras. *Towards Multiresolution Integrated Surface and Volume Data Representations.* PhD thesis, Universitat Politecnica de Catalunya, 2001.

[15] T. Porter and T. Duff. Compositing digital images. In *Hank Christiansen, editor, Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, July 1984.

[16] Milos Sramek. Fast ray-tracing of rectilinear volume data. In *Virtual Environments and Scientific Visualization '96. Eurographics Workshops*, pages 201–210, 1996.

[17] R. Srinivasan, S. Fang, and Su Huang. Volume rendering by template-based octree projection. In *Visualization in Scientific Computing '97. Eurographics Workshops.*, 1997.

[18] X. Tong, W. Wang, W. Tsang, and Z. Tang. Efficiently rendering large volume data using texture mapping hardware. In *Data Visualization'99*, 1999.

[19] L. Westover. Footprint evaluation for volume rendering. In *SIGGRAPH'90*, pages 367–376, 1990.

[20] K.J. Zuiderveld, A.H. Konong, and M.A. Viergever. Acceleration of ray-casting using 3-D distance transforms. In *VBC'92 SPIE Conf.*, volume 1808, pages 336–346, 1992.