The Visual Computer

Shadow generation for volumetric data sets

Rolf H. van Lengen, Jörg Meyer,
Mathias Matzat, Hans Hagen

University of Kaiserslautern,
Department of Computer Science,
P. O. Box 3049, D-67653 Kaiserslautern, Germany
e-mail: {lengen|jmeyer|hagen}@informatik.uni-kl.de
matzat@aed-graphics.de

Shadow information is essential for visual perception of complex scenes. Both shadow location and orientation clarify spatial relationships between objects. Shadows improve depth perception and the impression of realism. Our algorithm, integrated into a hybrid rendering system for medical data sets, is based on a shadow *Z*-buffer technique for fast and efficient shadow generation. Volumetric data contain information about the grid points only. Such data do not provide surface information that could be projected immediately onto the shadow map. To solve this problem, we have implemented two techniques. The first uses a modified adaptive version of the well-known marching cubes algorithm for the special characteristics of medical data sets. The algorithm uses material properties for a precise representation of object boundaries, generating volumetric objects quickly and effectively. There are two representations of the same data set: we use a view-independent approximation to display shadows and the original representation of the volume for object visualization in full precision. The second algorithm uses a ray-tracing approach to create shadow maps. The same routine is used for object rendering, but is restricted to depth-value generation. Semitransparent objects are handled by storing an intensity profile in addition to the depth value.

Key words: Volume visualization – Scientific visualization – Medical imaging – Shadow *Z*-buffer – Shadow map

Correspondence to: R.H. van Lengen

1 Introduction

Visual perception of a scene can be enhanced by different depth cues, such as occlusion, hidden surfaces, and shading. Shadow generation adds another depth cue to the scene because it clarifies the spatial relationship of objects in space. Figure 1 shows an example of how the position and orientation of shadows can be used to describe the location of an object with respect to a surface.

Without shadow information, it is impossible to judge the position of a sphere with respect to a surface (Fig. 1a). In Fig. 1b the sphere obviously touches the ground, while Fig. 1c, with the object in the same position, shows a levitating sphere. It is important to note that all three images are identical, differing only in the existence or absence of the shadow and its position. Shadows enable us to gain knowledge about the distribution of objects in space in a two-dimensional image without real depth. Experience and skill help us to interpret the situation correctly. Therefore, generation of shadow information is mandatory in three-dimensional computer graphics to avoid ambiguities and misinterpretations.

Shadow generation requires us to consider two aspects. First, we have to determine the shape. If a shadow is mapped onto a plane, the shape is created simply by projecting the object's silhouette onto the plane. If a shadow is mapped onto a nontrivial surface, we have to handle a more complicated situation for which we have to find a suitable method. Second, we have to calculate a light intensity at the position where shading is required. In general, a point that is not directly exposed to the light source is not completely dark, but is illuminated by reflections from other objects in the neighborhood. Simple illumination models use a constant term to describe diffuse lighting, but precise evaluation of light intensities requires a global illumination model (ray-tracing, radiosity).

This paper describes a method for creating shadow information for the three-dimensional visualization of hybrid medical data sets. The algorithm is integrated in our Computer Aided Image Processing for Medical Applications (CoMED) system. CoMED is a hybrid rendering engine for volumetric and geometric data with applications in medical imaging and diagnostics. The data sets are provided by imaging techniques such as computed tomography (CT) and magnetic resonance imaging (MRI). These volumetric data can be combined with geometric data from CAD. Both data sets share the same rendering pipeline. To speed up

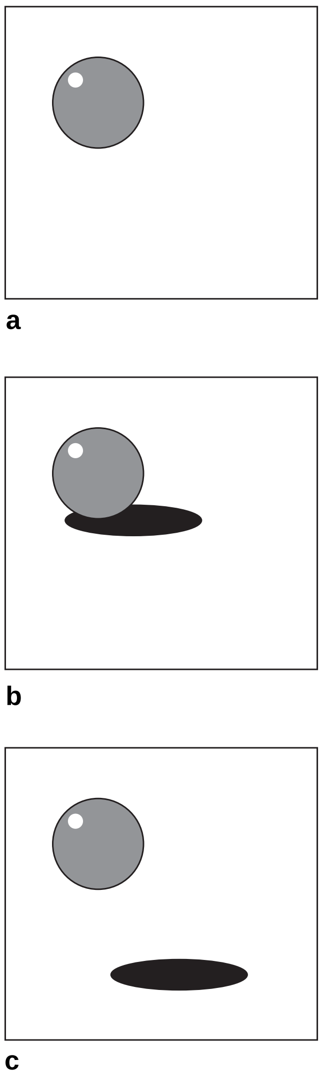


Fig. 1a–c. Shadow information is essential for the visual perception of an object location

the visualization process, the system is designed to run on a heterogeneous cluster of small workstations with low memory requirements.
First we describe some standard methods for shadow generation.

2 Previous and related work

Several methods for shadow generation (Watt and Watt 1992) are known. Most of them can be applied to polygons only. Others do not support transparent objects. For medical imaging, we need an algorithm that is fast, efficient, and ready to handle transparent layers of tissue. During an animation, we would like to keep the extra computational overhead for shadow generation low, in order to maintain efficiency.
In 1978, Atherton and colleages proposed an area subdivision algorithm for shadow generation. It is based on a hidden surface method, which was published in the year before by the same authors (Atherton and Weiler 1977). The algorithm determines the visible parts of the polygonal scene, viewed from the perspective of the light source. The visible polygons and their fractions make up the illuminated part of the scene. All other parts are in the shadow. Since all operations are performed in object space, the algorithm provides a very precise representation of shadows in the scene independently of the camera position.
The concept of shadow volumes, introduced by Crow (1977) and improved by Bergeron (1987), is a well-known technique that works primarily on polygonal scenes, but can also be used for more complicated surface representations. Each polygon, together with a light source, defines a shadow volume, eventually clipped at the viewing pyramid. A point is in the dark if there is an odd number of intersections between the shadow volume and a straight line, which is defined by the camera position and the point under consideration. If the number is even, the point is illuminated.
Appel (1968) integrates shadow generation into a scan line algorithm for hidden surface elimination. In a preprocessing stage, a light source is surrounded by an appropriate sphere. All polygons of the scene are projected successively from the view of the light source onto the sphere. For each polygon a list is created. It consists of all shadow polygons that partially or completely overlap the polygon in the projection. During visualization, the visibility problem is solved for the current scan line. Each part of the scan line corresponds to a visible region of a polygon or to the background of the scene. For shadow generation, the visible region of a polygon is projected onto each shadow polygon from the corresponding list. In case of overlapping, the shadow polygon produces a shadow and the corresponding portion of the scan line is dark. A major disadvantage of this method is that it is applicable to point sources only, and it also requires an ex-

tremely large amount of memory to store shadow polygon lists. Efficiency and computation time depend on scene complexity.

Ray-tracing or ray-casting algorithms provide another method for shadow generation. For each visible location in the scene, a line is drawn from the intersection point to each light source (shadow ray). An obstacle is detected if there is an intersection with any other object in the scene between the intersection point and the light source (Glassner 1989). In this case, the intersection point is in an area that is obscured by the obstacle with respect to the light source. This means that the intensity of the corresponding light source is neglected for the conclusive evaluation of pixel intensity. The final result for the intensity of a pixel is calculated by summing up weighted partial intensities, which must be attenuated in proportion to distance.

Ray-tracing algorithms can potentially produce high-quality photo-realistic images. Although several approaches for acceleration have been implemented, e.g., the light buffer (Haines and Greenberg 1987), one of the major drawbacks of ray-tracing algorithms is their complexity in time (Foley et al. 1989).

Traditional shadow *Z*-buffer algorithms can merely handle opaque objects because only the depth value of the closest intersection with respect to a certain light source is stored. Especially for medical data, it is important to reveal the interior structure of an object. Hence it is necessary to render parts of the object transparently. Our approach handles both opaque and transparent objects. For those pixels in the shadow map that are covered by transparent objects from the scene, an intensity profile is stored in addition to the depth value.

Imaging techniques, such as CT or MRI, produce volumetric data sets, which are arranged in a three-dimensional regular grid, with scalar or vector information at each vertex. In addition to scanned or measured data, a material label derived from a segmentation algorithm can be attached to each vertex.

We use a ray-casting technique to visualize presegmented volumetric data sets with label information (Lengen and Meyer 1994). Each ray traverses and samples the data set at discrete positions along the ray. Each sample provides the color and transparency of the material that is most likely at this position. A weighted average of all samples according to transparency determines the final color of the pixel.

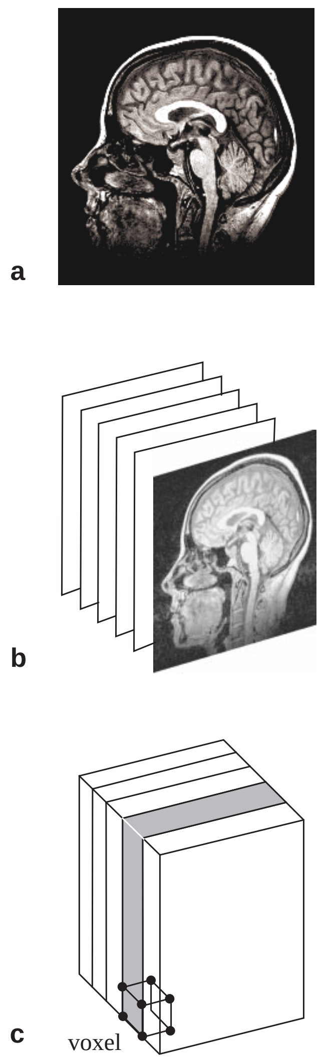


Fig. 2. **a** MRI scan; **b** array of slices; **c** block processing

Our system processes volumetric data sets that are arranged in slices of two-dimensional grids. Each pair of slices is called a block (Fig. 2). During visualization, these blocks must be loaded once only. The rendering stage processes each block and its neighborhood separately, so that at most four slices must be stored in memory at the same time (Lengen 1992). This drastically reduces the amount of memory as it is no to keep the whole data set in memory. It also enables us to run the system on small workstations, even on a standard PC. Our system is also able to handle geometric data, e.g.,

data from CAD, which can be combined with volumetric data so that we can display both data sets in the same image.
Area subdivision, shadow volumes, and scan line methods can be applied to polygon scenes only. They tend to be very slow for complex scenes. We need a method that provides shadow generation for non-polygonal scenes. As an alternative to the three algorithms already mentioned, our fourth method can handle objects in different representations. It uses a shadow ray that traverses the scene. The major drawback is the fact that the whole data set must be loaded into system memory in order for obstacles to be detected. Our ray tracer does not fulfil this prerequisite because it uses block processing, which implies that only a part of the data set is available in system memory at any time. Therefore we use a preprocessing technique that collects all the necessary shadow information prior to visualization.
The method is based on Williams' *Z*-buffer shadow algorithm (Williams 1978), which provides an efficient way of creating shadow information on scenes of arbitrary complexity. Shadows can be cast onto any other object in the scene. We implement a two-step method. First, we render the scene from the perspective of the light source. Instead of calculating colors and transparencies, only the z values of the closest object with respect to the light source are stored in a shadow map. It is necessary to calculate an individual shadow map for each light source.
In the second step, we render the scene from the camera perspective. Each visible object point is mapped into the coordinate system of the light source and projected onto the associated shadow map. If the depth value from the shadow map is lower than the z value of the projected point, then the point is in shadow; otherwise it is irradiated by that light source.
The following sections describe how to apply this principle to volume visualization.

3 An approximation approach to shadow generation

A volumetric data set can be interpreted as a regular grid of vertices with associated data values that are concentrated in single points. It is impossible to render these discrete data directly or to project them onto a shadow map. Such a data set can be sampled by a ray-tracing technique in order to obtain information about material composition and surface properties, or the data can be converted into a surface representation so that we can use standard algorithms for polygon rendering.

3.1 A surface representation for shadow maps

The most popular algorithm that creates surfaces from volume data is the marching cubes algorithm (Lorensen and Cline 1987). The surface divides the data set into two subsets containing the vertices inside and outside the surface. The surface itself is represented as a set of triangles. According to a predefined criterion, eight vertices of a voxel are subdivided into two sets of vertices (inside and outside) with respect to the surface. Triangle faces inside the voxel separate the two subsets. Within a voxel, there are 256 different configurations. Neglecting symmetrical cases, this large number can be reduced to 15 unique topologies. Surface information for each of these cases is stored in a table. Empty voxels or voxels with all vertices marked contain no surfaces. The algorithm guarantees a closed surface constructed of polygons (Montani et al. 1994). Triangle vertices are located on voxel edges that are connected to vertices from disjoined subsets (Matzat 1994).
Originally, the algorithm was intended to create isosurfaces on discrete data sets. The inside/outside classification criterion is a predefined threshold. The position of a triangle vertex on a voxel edge is determined by linear interpolation. In our system, we have a material label associated with a vertex that is evaluated as a criterion for an inside/outside test. Since we have binary information only, the triangle vertex would always be positioned at the center of a voxel edge and would cause serious artifacts, as we would always have the same angles for triangles. Therefore we use an additional threshold to identify a proper position for the vertex on the voxel edge. A fixed threshold would not be sufficient here because, if the threshold is out of the range of the two voxel corners, a new vertex (created by interpolation) would no longer be inside the voxel. Therefore

we use an adaptive threshold, which is the weighted average of neighboring voxels. The result of this method is a smooth surface that encloses all voxel vertices with identical material labels.

With this method we can create shadow maps for different kinds of objects. For the volume, two slices in a sequence are loaded into system memory each time, forming a block that is actually a single layer of voxels. For surfaces, we register the appropriate triangles into the voxels. These triangles, are transformed into the coordinate system of the light source, and are then projected onto the corresponding shadow map. For each pixel covered by the projection of the triangle, the distance between the light source and the triangle intersection is calculated. If the new triangle is closer to the light source than all the previous ones, the shadow-map entry is replaced by the new *z* value. A fast scan-conversion method projects the triangles onto the shadow map. It is important to note that the shadow maps can be calculated simultaneously for all light sources.

The method described is applicable only to light sources located outside the scene because the scene as a whole must be projected onto the shadow map. When a light source is inside the scene, we create a cube of six shadow maps, completely enclosing the scene (shadow cube). Before projecting a triangle onto the shadow cube, we determine the relevant surfaces of the cube, i.e., those faces that are affected by the triangle. Triangles must be clipped at the viewing pyramid to provide an accurate projection onto a cube face. If the result is a quadrangle, we decompose the polygon into triangles in order to apply our standard scan-conversion algorithm. During visualization, it is necessary to check in which viewing pyramid a sample point is located. According to the result, we evaluate the corresponding face of the shadow cube (Matzat 1996).

3.2 Artifacts

Shadows created by the shadow *Z*-buffer technique sometimes show artifacts due to different sampling rates that were used for shadow map generation and image rendering. Rough edges and Moiré patterns caused by self-shadowing are typical flaws. Reeves et al. (1987) introduce a method called percentage closer filtering (PCF), which can be applied to smooth the edges of a shadow (Fig. 3).

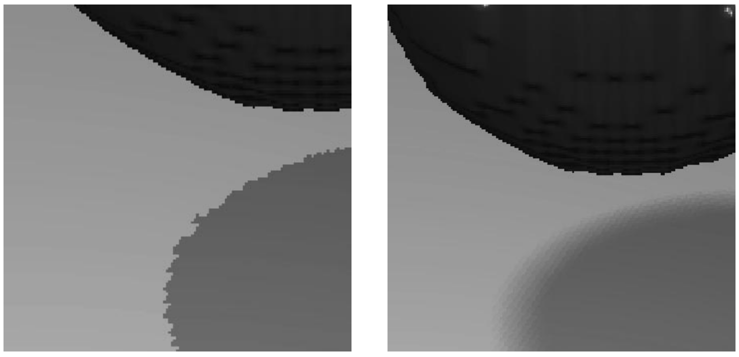
A PCF filter determines the percentage of a pixel not struck by a light source, thus avoiding a binary decision and abrupt discontinuities in light intensity. A small set of pixels in the neighborhood is checked for shadow features, and the ratio between unshaded pixels and the size of the whole area determines the fraction of light that illuminates the point.

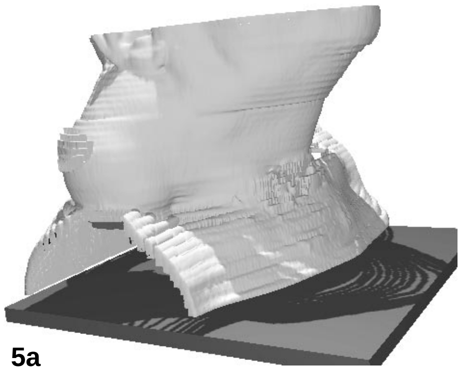
Self-shadowing is a difficult problem because the objects in the scene are not necessarily sampled at the same locations as those that were used for shadow-map generation. Usually they are completely different. Self-shadowing occurs if the distance *t* between a sample point and the light source is greater than a distance *s*, which is the depth value of a shadow map pixel, provided that both points belong to the same object surface (Fig. 4a).

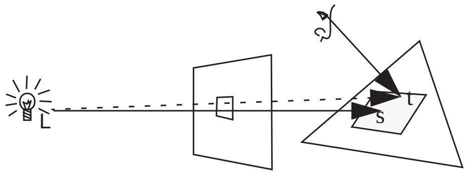
Image locations falsely identified as part of the shadow area severely disturb the visual perception of a scene. A standard method to reduce the error is to subtract a small bias from the distance between the test point and the light source before comparing it to the depth value in the shadow map. A bias can be a constant or a value chosen randomly from an interval (Reeves et al. 1987). Nevertheless, there are some cases when a single bias is not sufficient to avoid self-shadowing. The interval depends on the dimension of a given scene and requires interactive searches to find a practical value.

Our method tries to avoid the necessity of interactive manipulation. If a triangle is projected onto the shadow map, then for each pixel in the shadow map we select the largest depth value of a clipped triangle. We use the linearity of the triangle to reduce the number of tests on the four vertices of a shadow map pixel. If the largest value at one of the four corners is lower than the previous entry in the shadow map, we replace the old value (Fig. 4b). All other locations on the triangle surface inside the viewing pyramid of the pixel are located in front of the test point and will be illuminated. Once the corner with the largest value is determined, we can use the same corner for all other shadow-map pixels because they all have the same topology.

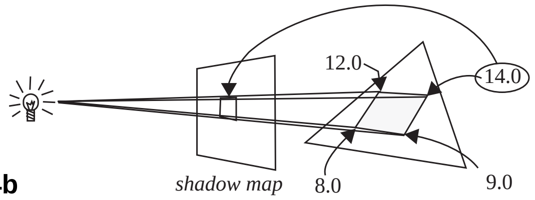
Filtering is applied during rendering because only those pixels that contribute to the visible part of the shadow in the final image must be filtered.


3a 3b


5a



4a



4b

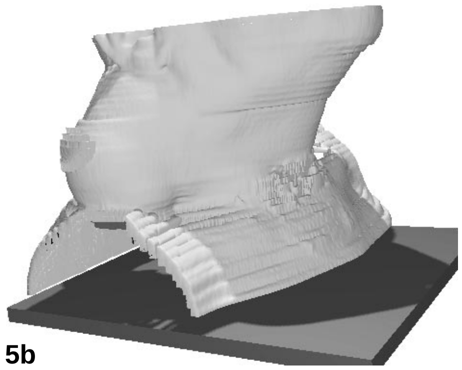

5b

Fig. 3. **a** unfiltered shadow of a sphere; **b** PCF filter

Fig. 4a, b. Self-shadowing

Fig. 5. **a** artifacts in detailed structures (marching cubes); **b** shadow is based on volume rendering (same algorithm as used for final image)

PCF filtering is less complex than bilinear filtering. Bilinear filtering is only possible in image space, since averaging z values would bear no relation to the geometry of the scene. For each technique, a certain neighborhood of the sample point in the shadow map must be scanned. Bilinear filtering would require the evaluation of the complete rendering function, i.e., the illumination model, for each sample position. This turns out to be very time consuming because we are using a very complex illumination model. PCF filtering just makes simple comparisons of depth values and averages binary numbers.

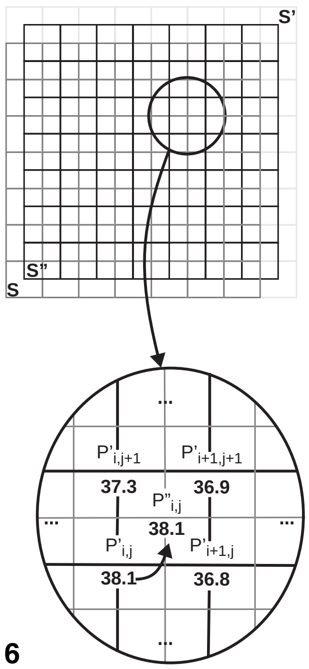
Shadows created by this method approximate the shape of the real shadow. This is due to the different representations of the data set, i.e., the boundary surface created by marching cubes, and the original data set, which is the basis of scene rendering. The main reason why we do not use a surface representation for the object itself is the quality and smoothness of the surface. In our tests, we never achieved such high quality and smoothness of surface with surface rendering as we did with volume rendering. With volume rendering, parameters can be chosen carefully so that the ray diffuses into the object up to a certain degree. This results in a much more realistic image.

Due to thresholding artifacts in surface reconstruction, noticeable differences appear in detailed structures (Fig. 5). In this case, it is necessary to use the same rendering technique and the same data representation for both shadow-map generation and final rendering.

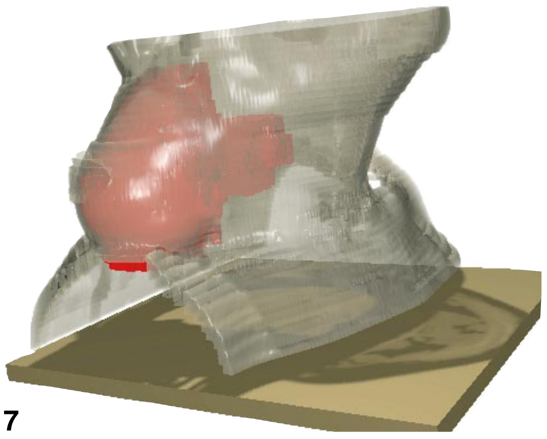
Both of the images in Fig. 5 comprise the same size of shadow maps, i.e., 100×100 pixels, which is about 2.26 % of the final image size. According to the notation used by Reeves et al. (1987), we selected a neighborhood of three pixels in each direction, a resolution factor of 1, and a bias of 50% of the length of a voxel diagonal to create the images. In contrast to Williams' (1978) proposal, we did not use 16-bit integers, but single-precision floating point numbers to represent the depth values in the shadow map. This avoids errors due to depth quantization.

4 Creating shadow maps with ray tracing techniques

Upgrading a ray tracer to generate shadow maps is straightforward and easily implemented. Further processing of material properties, colors, and light intensities is not necessary, since only depth information is required for the shadow map. The same routines that are used for image rendering can also be applied to shadow precomputation. The only difference is a switch of viewing perspective from the camera to the light sources. In contrast to projection methods, our ray-tracing approach is not capable of rendering several shadow maps simultaneously. In our case, this means that due to single block processing (see Sect. 2), the whole data set must be reloaded for each light source. Usually, the number of light sources is restricted, and shadow maps can be saved for subsequent computations. We can reuse a shadow map if the position of the respective light source and the number and position of objects in the scene do not change.
Artifacts due to ray-tracing techniques that possibly show up in the final image are similar to those noticeable in the images created with marching cubes shadow maps (see Sect. 3). Jagged edges can be smoothed with PCF filters in the way already described (see Sect. 3.2). Subsampling can be applied to avoid self-shadowing problems, but it is very inefficient and time consuming. Some ray-tracing systems do not support subsampling. Therefore, we have developed an effective pseudo-subsampling technique with a minimum of computational overhead.
For pseudo-subsampling, we extend the shadow map S by an additional row and column. The pixel corners of the shifted shadow map S' are centered on the pixels of the original shadow map S (Fig. 6). For each entry in the new shadow map S'', we create a new pixel value $P''_{i,j}$,from the maximum of the four pixels $P'_{i,j}$, $P'_{i+1,j}$, $P'_{i,j+1}$, and $P'_{i+1,j+1}$,that are covered by the new pixel.



6



7

Fig. 6. Pseudo-subsampling

Fig. 7. 3D Reconstruction of chin and throat, shadow with semitransparency: tumor is visible in red through transparent skin

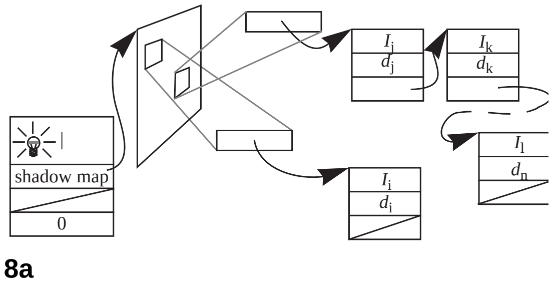
The results of this heuristic approach are very promising, provided that the shadow map is at least as large as the final image.
Both methods described are only capable of rendering shadows for opaque objects. Especially in medical imaging, it is often necessary to take a closer look into the interior structures of an object (Fig. 7). Therefore, exterior layers must be rendered transparently. The only way to work around this flaw is either to neglect object transparency for the shadow maps, or to ignore semitransparent and invisible objects. Both methods generate unrealistic and irritating results.
In the next section we describe an extension to the shadow *Z*-buffer algorithm that can also handle shadows of transparent objects.

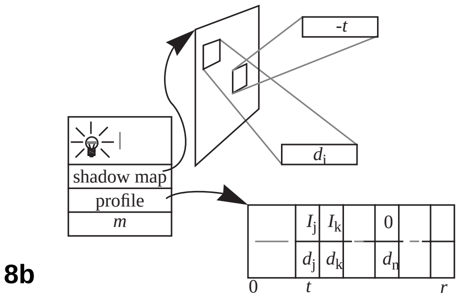
5 Shadows of transparent objects

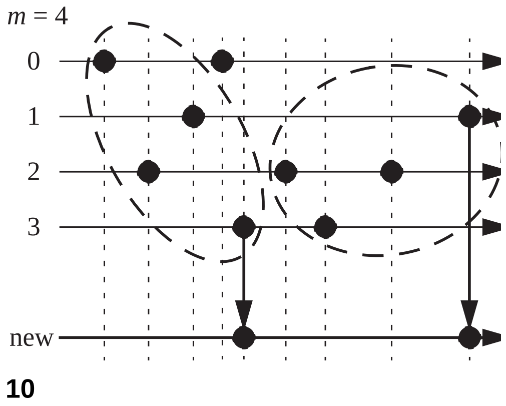
Shadow maps, introduced in the previous sections, consist of simple two-dimensional arrays. Each element stores a floating point number that represents the distance for complete absorption of a ray. If a ray passes one or more transparent objects, then only a certain percentage of light is absorbed from the medium. We develop a special data structure for extended shadow maps (Matzat 1996), which, if necessary, replaces the depth value by an intensity profile for each ray.

5.1 Extended shadow maps

Each light source is associated with a data structure that contains the entire information of an extended shadow map (Fig. 8). Each entry of the final map consists of two pointers and an integer. The first pointer refers to the shadow map, and the second one points to an array of intensity profiles. The integer specifies the length of this array. Figure 8 shows two potential states of the extended shadow map. Figure 8a describes the situation during preprocessing, i.e., the generation of the shadow maps, and Fig. 8b shows the visualization stage.
A shadow map entry can be interpreted in two ways. In the preprocessing stage, the entry is used as a pointer to a concatenated list. Each list entry consists of three elements. The first element contains the light intensity of a ray at a specific location. The second stores the distance value of this location with respect to the light source. The last element is a pointer that refers to the next element in the list, if it exists. When we create a shadow map, we have to check each sampling position along the ray to see if the light intensity has changed. If a predefined threshold is exceeded, we append a new element to the list, which is initialized with the corresponding values.
When the extended shadow map is complete, we convert it into a more compact representation. After this transformation, each element in the shadow map is reduced to a floating point number. We parse the shadow map and analyze each entry to prepare it for the transformation. If a pointer refers to a single list element, there are two cases: first, there could be an intersection with an opaque object that completely blocks out the light. In this case, the distance, which is stored in the list element, replaces the pointer, and the memory for the list element is freed. Second, the ray could have passed through a transparent object, but did not intersect any other objects, which means that the object casts no shadow on the surface of another object and only self-shadowing could occur. This case is masked by setting the distance to infinity.
If a shadow map entry points to a list that contains more than one element, its ray must have traversed several semitransparent objects. We call this a nontrivial list. In this case we create a linear array with the same number of elements as the sum of elements in all nontrivial lists. Each element consists of two entries, one for the remaining light intensity I_i and one for the depth value d_i. We copy the list contents sequentially to the array in the original order. The intensity value 0 marks the end of the list in the array. Finally, we replace the pointer in the shadow map with an index to the start position of the list in the linear array. We use a negative value for the index to separate it from the depth values. This data structure implements effective memory management that requires exactly the same amount of memory as the method for precise shadow generation introduced in the previous Section, provided that there are no transparent objects in the scene. If there are the shadow map finally only contains floating point numbers to store the distance values and a pointer to the array of intensity profiles. Another advantage is the fact that after transformation we no longer use any concatenated







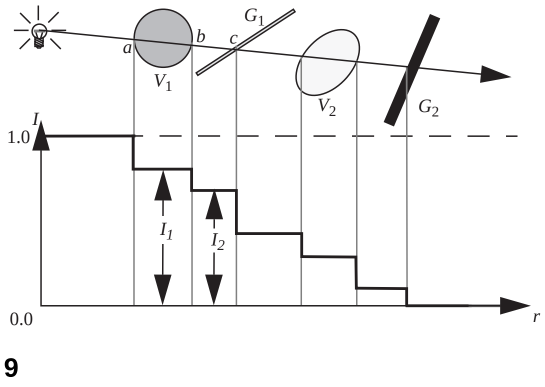


Fig. 8. Extended shadow map: **a** during preprocessing; **b** during rendering

Fig. 9. A ray intersecting several semitransparent volumetric and geometric objects

Fig. 10. Merging four intensity profiles

lists. This saves us a lot of pointer overhead. Additionally, traversing a list is more expensive than reading from an array during shadow calculation.

5.2 Generation of extended shadow maps

Since the appearance of a shadow primarily depends on the silhouette of an object, only the modulation of light intensity is measured on the object surface. A certain percentage of light is emitted by diffuse and specular reflection. The remaining percentage of light is transmitted through the surface. This means that we have abrupt changes in light intensity on a surface, and we assume exponential reduction of light intensity inside a volume. To simplify the model, we further assume that, in between two discrete sampling locations of the intensity profile, the light intensity does not change, even though there might be a volume in between. During visualization, object shadows are distinctly visible only in the projection on other objects rather than inside the obscuring object itself. The algorithm could be easily extended to handle multiple marching cubes surfaces. This would increase the complexity of the algorithm, but could be used to simulate outer surfaces shadowing inner surfaces. Multiple surfaces can also approximate density gradients inside the volume.

Figure 9 shows an example of a ray that intersects several semitransparent volumetric and geometric objects.

The ray intersects volume V_1. At position a we determine the reduction of intensity due too passing through the surface. This value is valid for all sample positions inside V_1. This means that for each sample position we assume a light intensity of I_1. When traversing V_1, the ray takes samples from the volume at several positions, and for each sample we calculate the intensity reduction again. The ray leaves the volume at position b, and we compare the reduction of intensity to the previous sample. If it is greater than the old value, we append this position, together with the current intensity, to the intensity profile.

The area between b and c is empty, and therefore we disregard the reduction of light intensity in this region. Following the ray, on the next position we have an intersection with an object G_1 (geometry) at c and an associated intensity I_2. Accordingly, the shadow of the volumetric object V_1, which is cast onto G_1, obtains correct intensities.

5.3 Evaluation of the extended shadow map

We transform each sample point from object space into the coordinate system of the light source and project it onto the shadow map. If the corresponding value in the shadow map is positive, we have a depth value that can be compared to the distance between the point and the light source, as already mentioned. A greater distance means that the point is in the shadow of that light source. If the pixel value is negative, we have an index that refers to an intensity profile, which is stored in an additional array. The negated index addresses a list that is traversed until we find an interval of two entries i, i+1 with $d_i<d<d_{i+1}$, where d denotes the distance between the point and the light source. The intensity I, which is assigned to the point, is equal to the intensity of entry $i(I=I_i)$. If the distance value of the point is greater than all other entries ($d_n<d$), we assume that the light source is completely obscured for that point (I=0).

5.4 Intensity profiles and filtering

Artifacts induced by this method are similar to those described in Sect. 3.2. Jagged edges can be smoothed by PCF filtering, and self-shadowing can be avoided by pseudo-subsampling. Just as before, we extend the shadow map by one row and one column. One pixel in the final shadow map covers four pixels of the original map (see Sect. 4). If these four pixels contain lists with just a single entry, the new depth value corresponds to their maximum. For nontrivial lists with more than one entry, we must combine them and create a new list.

Variable n_l denotes the length of list l ($l\in\{0,\dots,3\}$). Usually we have four lists, but some of the lists can be empty. Hence m refers to the number of nonempty lists. For simplicity reasons, we rename the nonempty lists, so that they obtain sequential numbers. Each list element consists of two entries $d_{i,k}$ and $I_{i,k}(0 \le i < n_k,\ 0 \le k < m)$, for storing the distance values and intensities, respectively, of the four pixels. Variables d_j and I_j accordingly refer to the values in the new list.

The following algorithm merges four lists and creates the new list:

Algorithm 1. Merging intensity profiles

```
For ($k$=0; $k<m$; $k$++)
    $e_k=d_{0,k}$;
$j$=0;
$d_j=\max(e_k;\ 0\le k<m)$;
/* while new list is incomplete */
while ($d_j<\max(d_{n_{k'}-1}k;\ 0\le k<m)$)
{   /* check all lists */
    for ($k$=0; $k<m$; $k$++)
    {   /* search for next item */
        $i$=0;
        while ($d_{i,k}\le d\,j$) and ($i<n_k$)
        {
            $i=i+1$;
        }
        if ($i<n_k$)
        {   /* store next item */
            $e_k=d_{i,k}$;
            $i_k=I_{i,k}$;
        }
    }
    $j=j+1$;
    $d_{\mathrm{j}}=\max(e_k;\ 0\le k<m)$;
}
```

The algorithm takes the first element from each of the four lists. The element containing the largest

distance value is the first element in the new list. Subsequently, we check each list for the next element with a distance value larger than the last distance value in the new list. The element containing the largest distance value is appended to the new list in the same way as before. Figure 10 shows the first and second group of elements chosen by the algorithm.
We repeat this procedure until we reach the end of all lists. When the new list is complete, we copy it to the array that keeps the intensity profile and create the next list.

6 Conclusions

We have presented two variants of the shadow *Z*-buffer algorithm. They have been designed for applications in medical imaging and volume visualization. These principles can be transferred to any other kind of data, especially volumetric and geometric representations. The results turned out to be very promising because most of the additional computations required for shadow generation can be performed during a preprocessing stage, and thus can be reused throughout an animation sequence without further overhead. Shadow-map computation takes about 70% of the time required for image rendering. The process can be accelerated by the use of a smaller shadow map, but there is a trade-off between the size of a shadow map and the accuracy of the shadow. The memory required to store the shadow maps is in linear proportion to the size of the maps. Since we only have two-dimensional maps, our algorithm consumes much less memory than ordinary volume methods for shadow generation.
The marching cubes method facilitates the simultaneous calculation of several shadow maps and therefore is an effective approach for approximative shadow-map generation. The method is independent of the visualization system, but it uses different representations for shadow map and image generation.
Subsampling used to avoid self-shadowing turned out to be superior to any other offset method. Although our system requires sequential shadow-map generation, the results are more realistic than the results of the approximation. We avoid self-shadowing by using a filter technique on the shadow map.

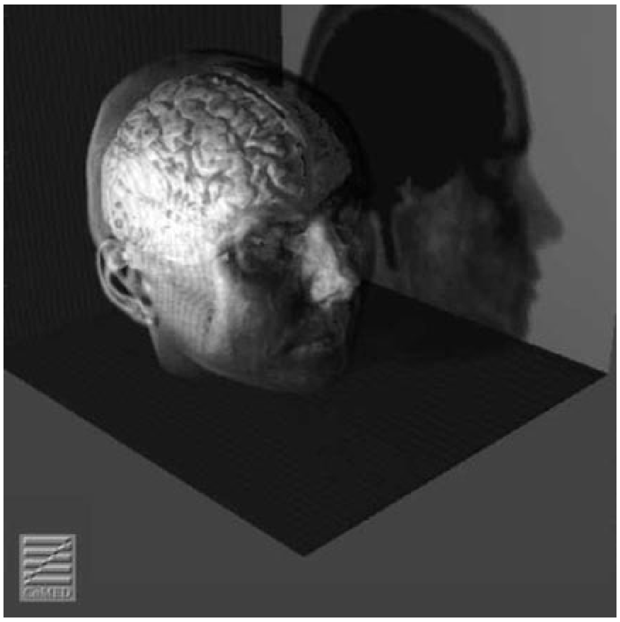


Fig. 11. 3D Reconstruction of an MRI scan

For our system, only the shadow *Z*-buffer technique was applicable. Transparent objects required an extension to the shadow *Z*-buffer. We have developed a data structure that enables us to store the additional information in a compact and efficient way. Here also, appropriate filter methods reliably help to avoid self-shadowing. Although this heuristic algorithm is not based on a physically correct model, the results, especially the simulations of penumbrae, show that the algorithm provides a fast method to create a good visual impression, and this enhances the plausiblity and quality of the final image (Fig. 11).

References

Appel A (1968) Some techniques for shading machine renderings of solids. AFIPS 1968 Springjoint Computer Conference 32:37–45

Atherton PR, Weiler K (1977) Hidden surface removal using polygon area sorting. ACM SIGGRAPH, Comput Graph, 11:214–222

Atherton PR, Weiler K, Greenberg D (1978) Polygon shading generation. ACM SIGGRAPH Comput Graph 12:275–281

Bergeron P (1987) A general version of Crow's shadow volumes, IEEE Comput Graph Appl 6:17–28

Crow FC (1977) Shadow algorithms for computer graphics. ACM SIGGRAPH, Comput Graph 11:242–248

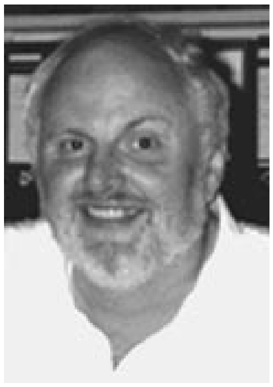
Foley J, Dam A van, Feiner S, Hughes J (1989) Computer graphics: principles and practice. Addison-Wesley Reading Mass

Glassner AS (1989) An introduction to raytracing. Academic, London
Haines EA, Greenberg DP (1987) The light buffer: a shadow-testing accelerator. IEEE Comput Graph Appl 6:6–16
Lengen RH van (1992) The priority Z-buffer. In: Hagen H, Müller H, Nielson GM (eds) Focus on scientific visualization. Springer, New York, pp 293–304
Lengen RH van, Meyer J (1994) Efficient 3-D visualization of hybrid medical data sets. Technical Report 257/94, University of Kaiserslautern, Germany
Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. ACM SIGGRAPH, Comput Graph 21:163–169
Matzat M (1994) Marching cube Algorithmus zur Oberflächenrekonstruktion medizinischer Daten. Thesis, University of Kaiserslautern, Deptartment of Computer Science, Kaiserslautern, Germany
Matzat M (1996) Die Schattenberechnung von Schatteninformationen für die Visualisierung medizinischer Daten. Thesis, University of Kaiserslautern, Deptartment of Computer Science, Kaiserslautern, Germany
Montani C, Scateni R, Scopigno R (1994) A modified look-up table for implicit disambiguation of marching cubes. Visual Comput 10(6):353–355
Reeves WT, Salesin DH, Cook RL (1987) Rendering antialiased shadows with depth maps. ACM SIGGRAPH, Comput Graph 21:283–291
Watt A, Watt M (1992) Advanced animation and rendering techniques. Addison-Wesley, New York
Williams L (1978) Casting curved shadows on curved surfaces. ACM SIGGRAPH, Comput Graph 12:270–274

JÖRG MEYER is a Research Assistant and PhD student at the University of Kaiserslautern, Germany. He received his MS in 1995 from the University of Kaiserslautern and specialized in the field of volume visualization, and medical imaging. His current research focuses on new rendering and display technologies and interactive volume visualization.

MATHIAS MATZAT received his MS from the University of Kaiserslautern in 1996. His research centers on scientific visualization and medical imaging. Currently he works for AED Graphics, Bonn.

ROLF HENDRIK VAN LENGEN is a Research Assistant and PhD student at the University of Kaiserslautern, Germany. He received his MS from the Technical University of Braunschweig, Germany, in 1989. His research interests include volume rendering, medical imaging, and scientific visualization.



HANS HAGEN has been Professor of Computer Science at the University of Kaiserslautern since 1988. He received his MA in 1979 at the University of Freiburg, Germany, and his PhD in 1982 from the Mathematics Department at University of Dortmund, Germany. From 1983 to 1986 , he was an adj. Assistant Professor at Arizona State University (Phoenix, Ariz.), and from 1986 to 1988 an Associate Professor at University of Braunschweig, Germany. His research interests include geometric modeling, scientific visualization, computer aided geometric design (CAGD), and computer graphics.