Real-Time Maximum Intensity Projection

Lukas Mroz, Andreas König and Eduard Gröller

Vienna University of Technology, Institute of Computer Graphics *

Abstract

Maximum Intensity Projection (MIP) is a volume rendering technique which is used to extract high-intensity structures from volumetric data. At each pixel the highest data value encountered along the corresponding viewing ray is determined. MIP is commonly used to extract vascular structures from medical MRI data sets (angiography). The usual way to compensate for the loss of spatial and occlusion information in MIP images is to view the data from different view points by rotating them. As the generation of MIP is usually non-interactive, this is done by calculating multiple images offline and playing them back as an animation.

In this paper a new algorithm is proposed which is capable of interactively generating Maximum Intensity Projection images using parallel projection and templates. Voxels of the data set which will never contribute to a MIP due to their neighborhood are removed during a preprocessing step. The remaining voxels are stored in a way which guarantees optimal cache coherency regardless of the viewing direction. For use on low-end hardware, a preview-mode is included which renders only more significant parts of the volume during user interaction. Furthermore we demonstrate the usability of our data structure for extensions of the MIP technique like *MIP with depth-shading* and *Local Maximum Intensity Projection (LMIP)*.

keywords: Volume Visualization, Maximum Intensity Projection, angiography

1 Introduction

The ability to depict blood vessels is of enormous importance for many medical imaging applications. CT and MRI scanners can be used to obtain volumetric data sets which allow the extraction of vascular structures. Especially data originating from MRI, which are most frequently used for this purpose, exhibit some properties which make the application of standard volume visualization techniques like ray casting [4] or iso-surface extraction [6] difficult. MRI data sets contain a significant amount of noise. Inhomogeneities in the sampled data make it difficult to extract surfaces of objects by specifying a single iso-value.

MIP exploits the fact, that within angiography data sets the data values of vascular structures are higher than the values of the surrounding tissue. By depicting the maximum data value seen through each pixel, the structure of the vessels contained in the data is captured. A straight forward method for calculating MIP is to perform ray

* Institute of Computer Graphics, Vienna University of Technology, Karlsplatz 13/186/2, A-1040 Vienna, Austria. email:{mroz, koenig, groeller}@cg.tuwien.ac.at

casting and search for the maximum sample value along the ray instead of the usual compositing process done in volume rendering. In contrast to direct volume rendering, no early ray termination is possible and the whole volume has to be processed. Depending on the quality requirements of the resulting image, different strategies for finding the maximum value along a ray can be used.

- **Analytical solution:** for each data cell which is intersected by the ray the maximum value encountered by the ray is calculated analytically. This is the most accurate but also computationally most expensive method [7].
- **Sampling and interpolation:** as usually done for ray casting, data values are sampled along the ray using trilinear interpolation. The cost of this approach depends on how many interpolations that do not affect the result can be avoided [7][9].
- **Nearest neighbor interpolation:** values of the data points closest to the ray are taken for maximum estimation. In combination with discrete ray traversal this is the fastest method. As no interpolation is performed, the voxel structure is visible in the resulting image as aliasing [1].

Recent algorithms for MIP employ a set of approaches for speeding up the rendering:

- **Ray traversal and interpolation optimization:** Sakas et al. [7] interpolate only if the maximum value of the examined cell is larger than the ray-maximum calculated so far. For additional speedup they use integer arithmetic for ray traversal and a cache-coherent volume storage scheme. Zuiderveld et al. [9] apply a similar technique to avoid trilinear interpolations. In addition, cells containing only background noise are not interpolated. For further speedup a low-resolution image containing lower-bound maximum estimations for each pixel is used. Cells with values below this bound can be skipped when the final image is generated. Finally a distance-volume is used to skip empty spaces.
- **Use graphics hardware:** Heidrich at al. [3] use conventional polygon rendering hardware to simulate MIP. Several iso-surfaces for different threshold values are extracted from the data set. Before rendering, the geometry is transformed in a way, that the depth of a polygon corresponds to the data value of its iso-surface. MIP is approximated by displaying the z-buffer as a range of gray values.
- **Splatting and shear warp:** Several approaches [1],[2] exploit the advantages of shear-warp rendering [5] to speed up MIP. Cai et al.[1] use an intermediate "worksheet" for compositing interpolated intensity contributions for projection of a single slice of the volume. The worksheet is then combined with the shear image to obtain the maxima. Several splatting modes with different speed/quality tradeoffs are available, run-length encoding and sorting of the encoded segments by value are used to achieve further speedup.

As a MIP contains no shading information, depth and occlusion information is lost (see Figure 1a). Structures with higher data value lying behind a lower valued object even appear to be in front of it. The most common way to ease the interpretation of such images is to animate the viewpoint while viewing. Another approach is to modulate the data values by their depth to achieve a kind of depth shading [3] (see Figure 1b). As the data values are modified before finding the maximum, MIP and depth shaded MIP of the same data may display different objects.

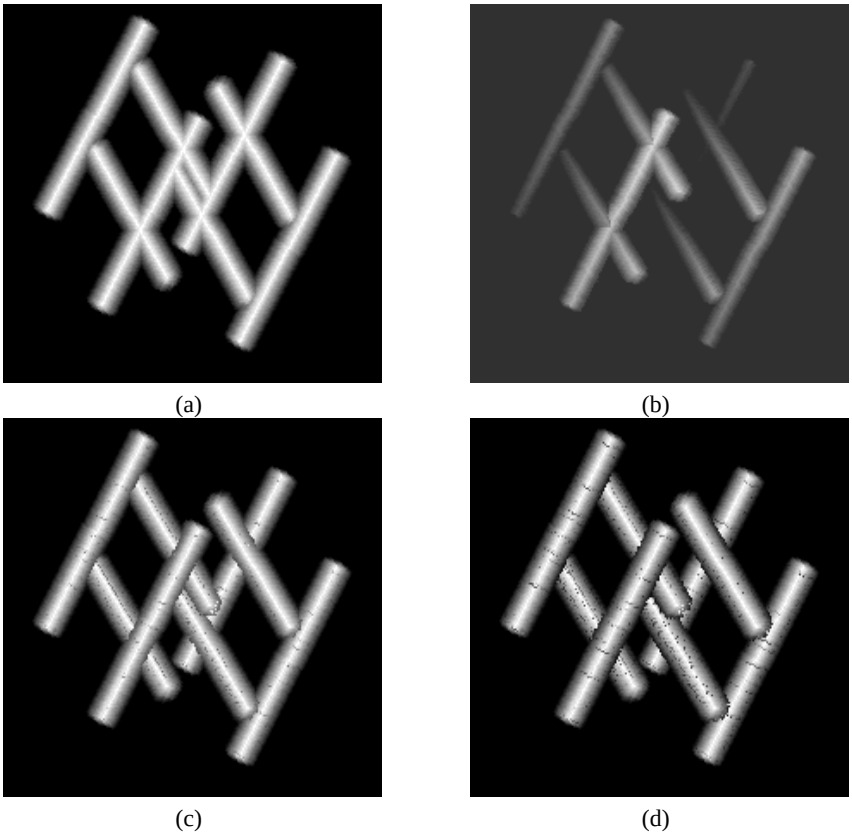


Fig. 1. a) MIP of a test data set containing soft bounded cylinders; b) Although depth shading provides some depth impression, the cylinders seem to intersect; c), d) While LMIP provides more information on the spatial structure, the results are sensitive to the threshold: c) high, d) low

Depth shading provides some hints on the spatial relation of objects. However it's performance is rather poor especially for tightly grouped structures. In such cases Closest Vessel Projection [7] or LMIP [8] can be used. As for MIP the volume is traversed along a ray, but the first local maximum which is above a user defined threshold is selected as the pixel color. If no value above the threshold is found along a ray, the global maximum along the ray is used for the pixel. With a high threshold value this method produces the same result as MIP, with a carefully selected one, less intense structures in front of more intense background are depicted, producing an effect similar to shading (see Figures 1c, d). As this method is very sensitive to the setting of the threshold, the ability to interactively tune this parameter is extremely important.

In the following we present a novel approach to generate MIP images. In Section 2 we discuss how to preprocess the volume data to eliminate voxels which will not contribute to a MIP. Furthermore we present a volume storage scheme which gets rid of overhead associated with skipping those voxels for rendering. In Section 3 the algorithm

for generating MIP is presented. In Section 4 extensions of the algorithm for generating depth-shaded MIP and LMIP using our data structure are discussed.

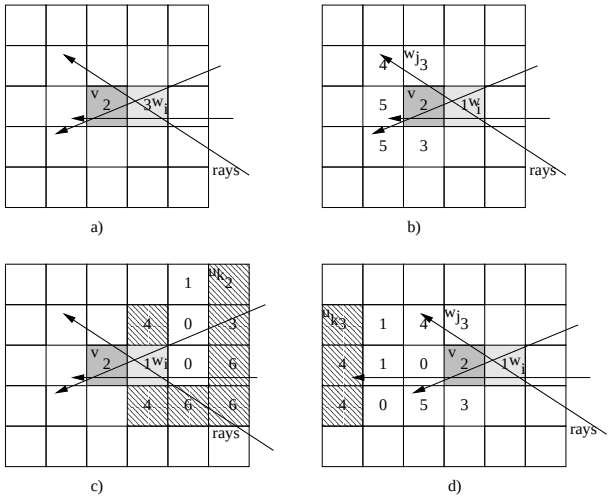


Fig. 2. Detection of voxels v which do not contribute to MIP. For simplicity the different cases are shown in 2D. a) v irrelevant for rays through w_i as $d(w_i) \geq d(v)$ b) v irrelevant for rays through w_i. If discrete 26-connected rays are tracked as in our algorithm, the two voxels with a value of 3 need not to be considered, as a ray through v and w can not pass through them. c) v irrelevant for rays through w_i as ray maxima are determined by voxels u_k. d) v irrelevant for rays through w_i, as ray maxima are determined by voxels u_k.

2 Preprocessing and Volume Storage

Most approaches to optimize volume traversal aim on excluding from the traversal and rendering process voxels which contain less-important information like low-valued background noise. In fact, in addition to this low-importance data, volumetric data sets usually contain a remarkable amount of voxels which never contribute to a MIP image. A voxel v is never visible and can be discarded if all possible rays through the voxel hit another voxel w with $d(w) \geq d(v)$ either before or after passing through v, where $d(v)$ is the data value at voxel v. This fact can be exploited when original voxel values are used for rendering using nearest neighbor interpolation. The hidden voxels can be identified by classifying all 26 neighboring voxels w_i according to the behavior of rays passing through w_i and v. If v does not contribute to the rays through any of it's neighbors, it can be removed. For efficiency reasons, this process is subdivided into four steps, first considering only the values of the direct neighbors w_i:

– If for a direct neighbor $d(w_i) \geq d(v)$, all rays through w_i and v will have at least the value $d(w_i)$, v is not needed (see Figure 2a).

– If for a direct neighbor $d(w_i) < d(v)$, but all possible viewing rays incoming from w_i through v hit direct neighbors w_j of v, with $min(d(w_j)) \geq d(v)$, all rays through w_i and v will have at least a value of $min(d(w_j))$. Thus v has no influence on rays through w_i (Figure 2b).
– If the first two tests fail for voxel w_i, the rays through v and w_i still may be influenced by more distant voxels u_j. If $min(d(u_j)) \geq d(v)$ and every ray hits some u_i, all rays through w_i and v are not influenced by v. (Figure 2c)
– Else, rays entering v through w_i and leaving through w_j may be blocked by distant voxels u_k with $min(d(u_k)) \geq d(v)$. In this case, v has also no influence on the maxima of the rays. (Figure 2d)

Voxel elimination based on the first two tests can be performed rapidly by accessing just direct neighbors of each voxel. The tracking of rays to more distant voxels can be implemented by recursively scanning the sub-volume containing all affected rays. Recursion in a certain direction is terminated if a voxel $d(u_j) \geq d(v)$ is found which blocks all rays in this direction. To avoid unnecessary recursive checks, voxels scheduled for removal are replaced by the minimum of their obscuring values (which is always $\geq$ their original value). To compensate for noise in the data and to increase the number of rejected voxels a user definable tolerance ε can be included into the comparison process, artificially lowering the value of each checked voxel v by ε. For voxel reduction rates using different elimination efforts and tolerance values please refer to Table 1.

data set	*resolution*	*Fast, ε=0%*	*Full, ε=0%*	*Fast, ε=1%*	*Full, ε=1%*
mr_angio	256x256x64	25%	47%	29%	53%
mr01	256x256x74	33%	46%	45%	55%
mr03	256x256x124	29%	58%	29%	58%

Table 1. Volume reduction rates. Fast optimization considers only direct neighbors, full optimization performs recursion up to a distance of 10 voxels. The results show the percentage of voxels removed. The fast optimization takes about 5s on a PII/333, a non-optimized implementation of the full preprocessing takes 3-4 minutes for the 256x256x74 data set.

Although distance volumes are an efficient way to skip large empty volume regions, they are less suited for volumes with large numbers of small empty spaces or even single empty voxels. In addition, several accesses to distance data stored in the empty space are usually required to skip a gap.

The data structure commonly used to store volume data is a 3D array which stores explicitly data values and encodes their position in space implicitly in the value's position in storage.

A way to completely get rid of unwanted voxels from a volume is to store the required voxels in an array of voxel-positions and to encode the value of the voxels implicitly into their position in this array (See Figure 3). The required sorting of all voxels according to their data value can be performed in linear time, as the limited range of possible data values allows to use histogram based sorting. The coordinates of a voxel within a volumetric dataset can be packed into a 32 bit integer, allowing the encoding

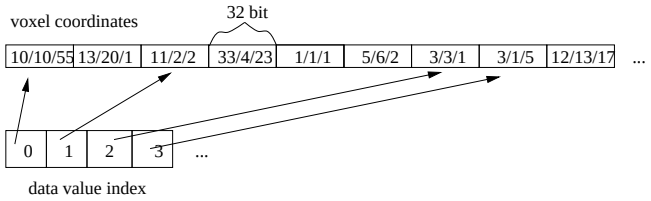


Fig. 3. Value-sorted array structure: All voxels are sorted according to data value and their position in space is stored in an array. An additional index points to the first voxel in the array for each possible data value

of volumes of up to 2048 x 2048 x 1024 voxels. A straight forward conversion of a 16 bit/value volume to a 32-bit position representation would double the memory cost. Omitting the voxels which have been marked by the preprocessing step as irrelevant leads to a factor of 1.0 to 1.5 in storage size compared to the original data. By storing voxels in a value-sorted array we gain several important advantages for MIP:

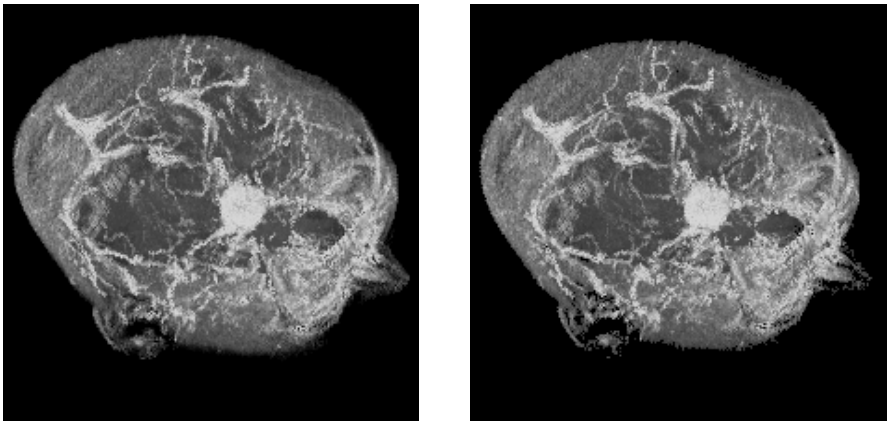
- The voxels can be splatted in the order of ascending data values as the array is traversed. Therefore **comparing** the value of the actual voxel with the screen content is **not necessary**. The value of the actual voxel is always $\geq$ the content of the screen, thus it's projection can be written into the image.
- As the array is traversed in the same way independent of the viewing direction, **optimum cache coherency** is always achieved.
- As blood vessels are represented by high data values, lower data values usually contain less important information. If an interactive display of the full data set is not possible on the given hardware, lower intensity values can be simply skipped (See Figure 4) to achieve interactivity during user interactions. The algorithm can be adjusted to display MIP at **any desired framerate**. The number of voxels which can be displayed at a specific frame rate can be easily derived automatically from a measurement of the number of voxels which can be rendered per second.

The voxel array structure does not store neighborhood information. It is therefore most useful for algorithms which work with voxels rather than cells. Acquiring the eight data values of a cell from the array without access to the original volume data would be a rather inefficient task.

3 Maximum Intensity Projection

As the voxels of the volume are processed in an arbitrary spatial order which is defined by the order within the value-sorted array, fast incremental techniques for projecting voxels along a ray can not be used. The parallel projection P of a point in 3D, however, can be expressed as the sum of independent projections of it's x, y, and z components.

$$\mathbf{P}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \mathbf{P}\begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} + \mathbf{P}\begin{pmatrix} 0 \\ y \\ 0 \end{pmatrix} + \mathbf{P}\begin{pmatrix} 0 \\ 0 \\ z \end{pmatrix}$$



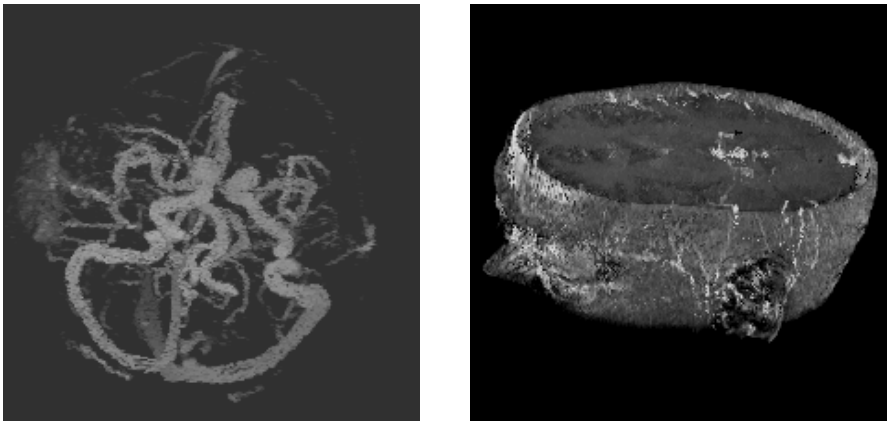
(a) (b)

Fig. 4. a) MIP of the mr01 data set, 2.6M voxels. b) Interactive preview of the same data at 10 fps on a P233MMX with approximately 25% of the volume data displayed. Only slight differences are noticeable

The projections of each possible voxel coordinate along a coordinate axis can be precalculated and stored into a template. A combination of three templates `xs`,`ys`,`zs`, can be used to inexpensively calculate the position of each voxel's projection on the screen. For increased efficiency, the templates can immediately store offsets into the image buffer instead of x/y screen coordinates. The resulting rendering loop is simple and efficient:

```
xs=make_template(viewmatrix, volumesize_x);
do the same for ys, zs;
for intensity = 0 to max do
  color = gray[intensity];
  for i = first[intensity] to last[intensity] do
    voxpos = value_sorted_array[i];
    screen[xs[voxpos.x]+ys[voxpos.y]+zs[voxpos.z]]=color;
  end
end
```

Similar to other MIP algorithms this approach is also a tradeoff between speed and quality. As each voxel of the data set is projected onto exactly one pixel, the size of the resulting image depends on the size of the volume. Furthermore the composition of multiple integer templates for determining the position of the projection in combination with the projection of a voxel onto exactly one pixel may produce "holes" within the projection of the volume. Although clearly apparent in a static image, due to the high interactivity of this method these artifacts do not disturb the perception of the data during interactive work. The artifacts can be significantly reduced by using sub-pixel coordinates in x-direction. As the templates store offsets into the image buffer ($x + img\_width * y$) instead of x/y coordinates, subsampling in y-direction is not possible.



(a) (b)

Fig. 5. a) MIP of the mr03 angiography data set with depth shading. b) LMIP of mr01

4 Extensions

The algorithm presented in the previous section can be easily extended to generate depth-shaded images (Figure 5). Similar to the screen position templates, three z-templates are generated to calculate the depth of each voxel's projection. The intensity value of each voxel is modulated by it's depth and written into the image buffer only if the value of the pixel at this position is smaller than the new value. As the depicted maximum values do not directly correspond to data values, applying this method to preprocessed volume data may produce incorrect results.

The second extension of the algorithm is capable of generating LMIP images providing the possibility to interactively adjust the threshold parameter. For generating a MIP image, the order of examining samples along a ray is not relevant. Straight forward LMIP requires the samples to be processed front to back along the viewing ray in order to find the first local maximum. Using z-templates for voxel depth calculation and two z-buffers per pixel allows to extract the closest local maximum from samples arriving in arbitrary order. The `z[pix]` buffer stores the depth of the currently visible voxel, while `zb[pix]` stores a "back-clipping" distance for each pixel. As the voxels are processed in order of ascending data value, all voxels below the LMIP-threshold can be first projected using the simple and fast MIP algorithm without z-calculation and local minimum tracking. Among all voxels above the threshold which are processed later, closest local maxima have to be found. At the beginning, z[] is initialized to contain the maximum possible distance. With `z` and `v` containing the depth and value of a projected voxel, the closest local maximum along a ray is found by

```
if (z<z[pix])
  screen[pix]=v; zb[pix]=z[pix]; z[pix]=z;
else if (z<zb[pix])
  screen[pix]=v; z[pix]=z;
```

The first condition detects voxels closer than the currently displayed voxel. As they have at least the same intensity as the currently displayed one, they are entered into the `screen` and `z[]` buffers, the back clipping plane is moved forward to the previous `z[]` position (Figure 6a). The second condition takes care of new voxels which are placed between the currently displayed voxel and the back clipping distance and are at least of the same intensity as the current one. In this case the `screen` and `z[]` are set to the new voxel. Voxels behind the back clipping distance are ignored (Figure 6b).

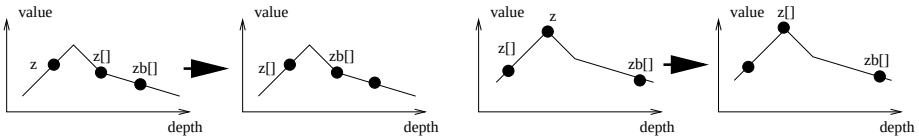


Fig. 6. Local Maximum Intensity Projection for non-sequential samples

5 Future Work

A shear-warp based projection could be used instead of templates with little impact on performance, with the advantage of removing most of the artifacts. Additional future work will concentrate on preprocessing and the use of cells instead of voxels in order to allow high-quality MIP rendering using the same sorted data structure.

6 Results

The volume preprocessing technique as well as all three rendering algorithms presented in this paper have been implemented into a Java applet (*http://www.cg.tuwien.ac.at/research/vis/vismed/RT-MIP/*). The rendering times in Table 2 have been measured on a PII/333 PC using the Java virtual machine of JDK 1.1.6 with a just in time compiler. A C++ version of the MIP rendering algorithm has shown an approximately 30% better performance than the Java version. For measurement a full preprocessing step with a tolerance of 1% has been applied to the data (See Table 1).

data set	*MIP*	*depth-shaded*	*LMIP*
mr_angio 256x256x64	198ms	275ms	274ms
mr01 256x256x74	212ms	279ms	359ms
mr03 256x256x124	343ms	457ms	388ms

Table 2. Frame rendering times for the different algorithms on a PII/333 PC. The timings for LMIP depend on the chosen threshold.

7 Conclusion

We have presented a new rendering algorithm for Maximum Intensity Projection. In combination with a preprocessing step which removes parts of the volume which do not contribute to a MIP image and a volume storage scheme which eliminates overhead for traversing empty regions of the volume, the algorithm achieves highly interactive frame rates. A preview mode for interactions on low-end hardware is provided basically "for free" by skipping voxels with low data values. Parallel projection is reduced to the addition of three values stored in lookup tables. Two extensions of the algorithm allow to generate depth-shaded MIP and LMIP at comparable speed. Although some artifacts are present in the image as no interpolation and a quick but jaggy projection method is used, the high interactivity of this method make it suitable for exploration of angiography data sets.

8 Acknowledgements

The work presented in this publication has been funded by the V^{is}M^{ed} project (`http://www.vismed.at`). V^{is}M^{ed} is supported by *Tiani Medgraph*, Vienna, `http://www.tiani.com`, and the *Forschungsförderungsfonds für die gewerbliche Wirtschaft*, Austria, `http://www.telekom.at/fff/`. The mr_angio data set is courtesy of Tiani Medgraph GesmbH, Vienna. The mr01 and mr03 data sets are available from the United Medical and Dental Schools (UMDS) Image Processing Group in London `http://www-ipg.umds.ac.uk/archive/heads.html`

References

1. W. Cai and G. Sakas. Maximum intensity projection using splatting in sheared object space. In *Proceedings EUROGRAPHICS '98*, pages C113–C124, 1998.
2. B. Csebfalvi, A. König, and E. Gröller. Fast maximum intensity projection using binary shear-warp factorization. Technical Report TR-186-2-98-27 at the Institute of Computer Graphics, Vienna University of Technology, 1998.
3. W. Heidrich, M. McCool, and J. Stevens. Interactive maximum projection volume rendering. In *Proceedings Visualization '95*, pages 11–18, 1995.
4. J. T. Kajiya. Ray tracing volume densities. In *Proceedings of ACM SIGGRAPH'84*, pages 165–174, 1984.
5. P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorisation of the viewing transform. In *Proceedings of ACM SIGGRAPH'94*, pages 451–459, 1984.
6. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM SIGGRAPH'87*, pages 163–189, 1987.
7. G. Sakas, M. Grimm, and A. Savopoulos. Optimized maximum intensity projection. In *Proceedings of 5th EUROGRAPHICS Workshop on Rendering Techniques*, pages 55–63, Dublin, Ireland, Italy, 1995.
8. Y. Sato, N. Shiraga, S. Nakajima, S. Tamura, and R. Kikinis. Lmip: Local maximum intensity projection - a new rendering method for vascular visualization. *Journal of Computer Assisted Tomography*, 22(6), 1998.
9. K. J. Zuiderveld, A. H. J. Koning, and M. A. Viergever. Techniques for speeding up high-quality perspective maximum intensity projection. *Pattern Recognition Letters*, 15:507–517, 1994.