Performing Maximum Intensity Projection with the Visualization Toolkit

Stefan Bruckner*

Seminar Paper
The Institute of Computer Graphics and Algorithms
Vienna University of Technology, Austria

Abstract

Maximum Intensity Projection (MIP) is a volume rendering technique commonly used to depict vascular structures. The Visualization Toolkit (*VTK*) is an open source, freely available software system for 3D computer graphics, image processing, and visualization. In this paper, MIP and related methods are presented and *VTK*'s capabilities to use these methods are examined.

Keywords: Visualization Toolkit, *VTK*, Maximum Intensity Projection, MIP

1 Introduction

For many scientific applications three-dimensional arrays of data generated by Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) have to be analyzed. Since the examination of individual cross-section images (commonly known as slices) generated by these methods requires high expertise and experience, volume rendering techniques are increasingly applied on such datasets to ease interpretation. They allow experts to see the interior structures of a volume, while still depicting spacial relationships.

Methods used to visualize volume data include Surface Rendering and Direct Volume Rendering:

Surface Rendering (SR): SR methods are also called feature-extraction or iso-surfacing and fit planar polygons or surface patches to constant-value contour surfaces [1], [2]. SR methods are usually faster than DVR methods since they traverse the dataset once, for a given threshold value, to obtain the surface and then conventional rendering methods (which may be in hardware) are used to produce the images. New views of the surface can be generated quickly. Using a new threshold is time consuming since the original dataset must be traversed again.

Direct Volume Rendering (DVR): DVR (also called composite rendering) methods map elements directly into screen space without using geometric primitives as an intermediate representation. Color and opacity transfer functions, which control the mapping of scalar values to color and/or opacity values, have to be specified. These values are accumulated along a ray path and displayed at the corresponding pixel location. DVR methods are especially good for datasets with amorphous features such as clouds, fluids, and gases. A disadvantage is that the entire dataset must be traversed for each rendered image [3].

MIP is a method which only displays the highest intensity value seen through each pixel. It actually can be viewed as a simplified DVR, which does not require explicit definition of transfer functions. In MIP, each value in the scalar field is associated with an intensity and the maximum intensity seen through a pixel is projected onto that pixel. It looks more like a search algorithm than a traditional volume color/opacity accumulation algorithm.

MIP is often employed in clinical practice for depicting vascular structures. The data is a set of slices where most areas are dark, but vessels tend to be brighter. This set is collapsed into a single image by performing a projection through the set that assigns the brightest voxel over all slices to each pixel in the projection. In contrast to DVR, MIP does not require the tedious generation of color and opacity transfer functions, but unlike SR it does still display density information. In addition, since datasets usually contain a lot of noise, threshold values for SR, which allow extraction of vascular structures are difficult to find.

This paper gives an overview over MIP and related methods and shows how *VTK* can be used to perform these techniques efficiently.

In Section 2 the basic MIP algorithm is described. Section 3 introduces extensions and optimizations that have been proposed for the MIP technique. In Section 4, the MIP functionality of *VTK* is examined and practical examples for performing MIP using *VTK* are given. Finally, this paper is concluded in Section 5.

2 Algorithm

The algorithm usually used for MIP is ray casting. For every pixel in the output image a ray is shot into the volume data. The maximum value along this ray is determined and displayed at the corresponding pixel location.

Figure 1 depicts this basic algorithm: For every pixel (1) in the viewing plane (2) a ray (3) is cast through the volume (4). Although other voxels are intersected by the ray, only the voxel with the highest intensity value (5) is projected onto the viewing plane at that pixel position.

Depending on the quality requirements of the resulting image, different strategies for finding the maximum value along a ray can be used [4]:

Analytical solution: For each data cell which is intersected by the ray the maximum is calculated analytically. Trilinear blending of the cell vertices is usually assumed for data values within a cell, thus the maximum of a third degree polynomial has to be calculated. This is the most accurate but also computationally most expensive method.

Sampling and interpolation: As usually done for ray casting, data values are sampled along a ray. Values are then interpolated. The use of simple nearest neighbor interpolation introduces aliasing effects - the voxel structure is visible in the resulting image (Figure 2). For better quality, trilinear interpolation can be used, which does impact performance. The cost of this approach depends on how many interpolations can be avoided that will not affect the ray maximum.

*stefan.bruckner@chello.at

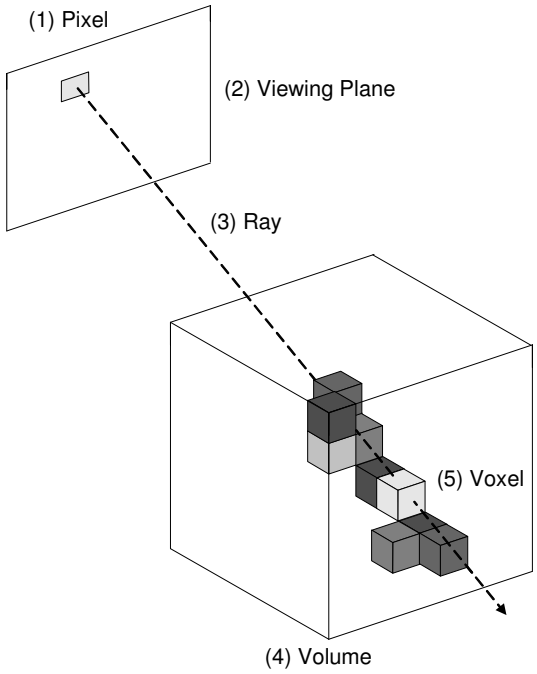
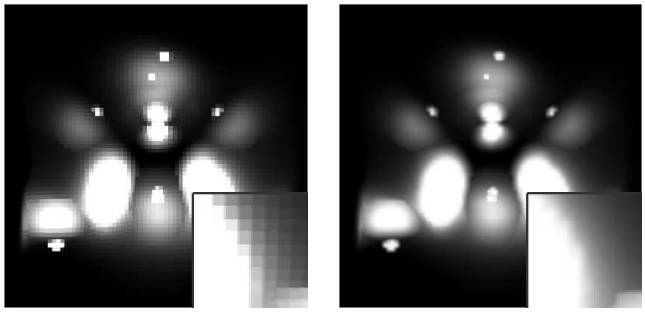


Figure 1: Maximum Intensity Projection through ray casting

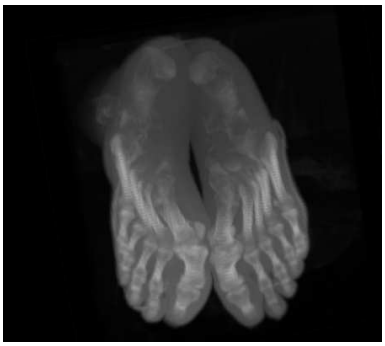


(1) Nearest-neighbor (2) Trilinear

Figure 2: A MIP rendered dataset using nearest-neighbor and trilinear interpolation

(1) Isosurface

(2) MIP



(3) DMIP

Figure 3: A human feet dataset rendered using an Isosurface ray cast function, MIP and DMIP - the problem of missing depth information in MIP is clearly visible

(1) MIP (2) LMIP

Figure 4: A human kidney dataset rendered with MIP and LMIP [5]

3 Improvements

MIP does not convey depth relationships, nor does it depict overlapping structures as a focal increase in opacity. As a MIP image contains no shading information, depth and occlusion information is lost. Structures with higher intensity values behind lower values structures appear to be in front of them, thus making interpretation of images somewhat difficult. The following sections describe strategies to solve these problems.

3.1 Extensions

To overcome the limitations of the basic MIP algorithm, some extensions have been proposed:

Depth-shaded Maximum Intensity Projection (DMIP): In depth-shaded MIP data values are modulated by their depth to achieve a kind of depth shading. One of the simplest depth cues to implement is an intensity ramp that dims parts of the volume farther from the eye (3).

Local Maximum Intensity Projection (LMIP): The LMIP image (Figure 4) is created by tracing an optical ray from the viewpoint in the viewing direction, and then selecting the first local maximum value encountered that is larger than a preselected threshold value. LMIP thus requires one threshold parameter to be tuned. If no local maximum value larger than the threshold value is encountered, the global maximum value along the ray is used. While MIP images are obtained irrespective of the direction of traverse, i.e. front to back or back to front, LMIP images are dependent on the traversing direction [5].

3.2 Optimizations

In contrast to extending the MIP method, another approach is to optimize the performance of MIP algorithms so that images can be generated in real-time at high frame rates. By interactively changing the point of view, the user is able to perceive lost depth information. Several efficient methods for computing MIP images have been proposed:

Optimization of ray traversal and interpolation: Sakas et al. [6] propose to optimize the ray traversal by evaluating cell maxima only if the maximum value of the examined cell is larger than the ray-maximum calculated so far. For further speedup integer arithmetics for ray traversal and a cache-coherent volume storage scheme can be used. Zuiderveld et al. [7] also generate a low resolution image before the main rendering step. This low-resolution image contains lower-bound estimations for the maximum of pixel clusters and while generation of the final image cells with values below this bound can be skipped. Finally a distance-volume is used to skip empty spaces efficiently.

Use of graphics hardware: For high performance, conventional polygon hardware can be exploited to simulate MIP. The hidden surface removal algorithm used on these boards is almost always depth-buffer (z-buffer) based. Heidrich et al. [8] use this z-buffer to perform maximum operations. Therefore several surfaces for different threshold values are extracted from the dataset so that brighter points are closer to the image plane than darker points. Before rendering, the geometry is transformed in a way, that the depth of a polygon corresponds to the data value of its iso-surface.

The VolumePro board [9] is a purely hardware based solution for volume rendering and also MIP. The board uses shear-warp based projection to create images at high frame rates. Rays are sent into the dataset from each pixel on a base plane, which is co-planar to the face of the volume data that is most parallel and nearest to the image plane. Because the image plane is typically at some angle to the base-plane, the resulting base-plane image is warped onto the image plane. The main advantage of this algorithm is that voxels can be read and processed in planes of voxels (so called slices) that are parallel to the base-plane. Within a slice, voxels are read from memory a scanline of voxels at a time, in top to bottom order. This leads to regular, object-order data access. Sample values along rays are calculated using trilinear interpolation.

Splatting and shear-warp factorization: The shear-warp factorization rendering algorithm [10] is one of the fastest object space rendering algorithms. Therefore, it can be used as an alternative to conventional projection. The advantage of shear-warp factorization is that sheared voxels are viewed and projected only along the three coordinate axes, i.e. along the principle viewing direction in sheared object space rather than an arbitrary viewing direction, which makes it possible to traverse and evaluate the maximum slice by slice. Neighbor voxels share the same footprint and are exactly one pixel apart. The dataset is sheared first according to the shear transformation and an intermediate image, which is called ”shear image”, is generated by voxel projection in sheared space. Each voxel is splatted according to its footprint in sheared object space. After all voxels have been splatted, the intermediate shear image is warped to create the result image displayed on the screen according to the warp transformation. In order to calculate different voxel contributions to one pixel resulting from splatting parallel pipes rather than cubes, an additional intermediate image plane called ”worksheet” is introduced. The worksheet is another image buffer which has the same size as the shear image. For each sheared data slice, voxels are first splatted to the worksheet and values are accumulated in the usual way. After accumulating all contributions of one slice to the worksheet, pixels in the worksheet are compared with pixels in the shear image one by one to evaluate their maximum value. The effect of introducing the worksheet is that a partial image is calculated first containing the contributions of one voxel-slice only before it is compared with the final shear image [11].

Elimination of irrelevant voxels Volumetric data sets usually contain a remarkable amount of voxels which never contribute to a MIP image. If all possible rays through a voxel V hit another voxel W with the data value of V being lower than the

data value of W either before or after passing through V, the voxel V can be discarded since it does not contribute to MIP [12].

Alternative storage schemes: The data structure commonly used to store volume data is a 3D array which explicitly stores data values and encodes their position in space implicitly in the values's position in storage. But since the direction of traversal is irrelevant, alternative storage schemes can be used. Preprocessing steps can be used to reduce the number of voxels for each view-set. The remaining voxels can now be stored in an array of voxel-positions, where the value of the voxels is implicitly encoded into their position in this value-sorted array. This way it is possible to completely get rid of unwanted voxels.

4 Performing MIP in *VTK*

4.1 Basic MIP

In *VTK* volume rendering is only implemented for structured points datasets, unstructured datasets cannot be directly volume rendered. However, an unstructured grid can be resampled to take advantage of *VTK*'s volume rendering capabilities. Volumetric data is represented by the class `vtkVolume`. It contains information about position, orientation as well as pointers to the volume's properties and the data mapper [13].

In order to perform volume rendering in *VTK*, transfer functions have to be defined. Briefly, a transfer function defines the mapping from scalar input data (e.g., density values in a CT dataset) to the color and/or opacity values which are actually displayed. The MIP technique itself, in contrast to composite DVR, does not require transfer functions, but since *VTK*'s ray casting classes can be used to perform multiple rendering methods, transfer functions have to be assigned. To produce a common MIP image the opacity function is set to constant one, i.e. all voxels are displayed completely opaque. The color function is a linear function which assigns black to the lowest scalar value and white to the hightest value, thus MIP images are always gray-scale images.

The actual projection is performed by a mapper class, `vtkVolumeRayCastMapper`, which gets a structured points dataset as input and produces a mapped structured points dataset. The mapper class uses a `vtkVolumeRayCastFunction` which is responsible for casting one ray. Different ray casting techniques can be performed by simply changing the type of ray cast function used by the volume mapper.

For the `vtkVolumeRayCastMIPFunction`, either nearest-neighbour or trilinear interpolation can be used. A method call allows switching between those two. Another interesting feature is ray step size. It either can be set manually or automatically. When using the automatic setting, *VTK* chooses the best (i.e. lowest) distance that guarantees a predefined frame-rate. Of course, for high frame-rates, this might introduce aliasing, since the ray step sizes might be chosen too large. Program 4.1 shows how MIP rendering is set up in *VTK*.

In order to optimize performance *VTK* allows cropping volumes to be defined. Cropping is a method of defining the region of interest of the volume using six planes - two along each major axis. These planes break the volume into 27 regions. The cropping volume can be defined by setting a bit flag. In practice, only a few bit masks are useful. Figure 5 depicts the most common cropping volumes. For datasets with large empty regions, cropping volumes are a simple method to speed up rendering.

Additionally, to use arbitrary bounding volumes `vtkProjectedPolyDataRayBounder` can be utilized. It allows clipping of rays against a polygonal object. This clipping

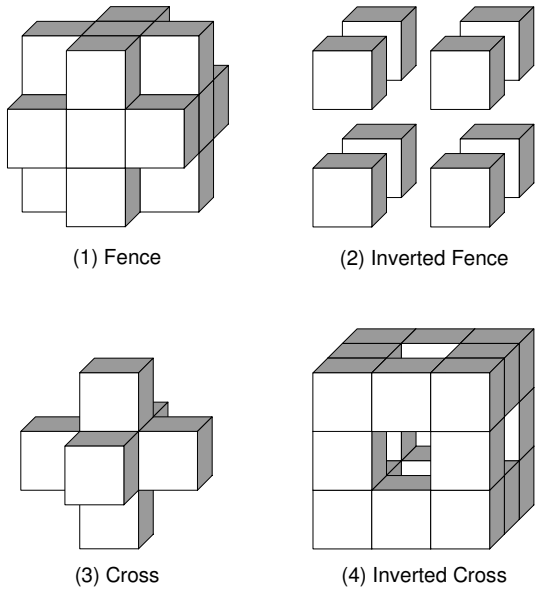


Figure 5: Cropping volumes in *VTK*

is done using the graphics hardware (if available). The ray bounder finds the nearest and farthest locations at which the ray can intersect the polygonal data and traverses the ray only in this interval. Program 4.2 shows how the ray bounder can be set up to clip the ray against a sphere.

A more advanced way to eliminate empty regions in a volume is to create a geometric iso-surface of the volume and then use this data for ray bounding. This provides a very "tight fit" and can therefore have significant impact on performance.

4.2 Extensions

In the current *VTK* distribution (4.0) only basic MIP is implemented. However, using the available source code, one can modify the existing `vtkVolumeMIPRayCastFunction` to perform extended techniques, such as DMIP or LMIP. A simple implementation of DMIP (Program 4.3) just uses the known values of current ray step and maximum ray length to compute the shading, which is then applied to the value before performing the maximum operation. The `ambient` and `changeRate` parameters were introduced to control the shading function. The value of `ambient` is the attenuation of the background image, `changeRate` is the rate of attenuation change per unit step along the ray. Setting both values to zero produces an unshaded MIP image.

This modification adds very little computational complexity, thus performance of MIP and DMIP is almost equal. For appropriate ray-step distances (appropriate means below the volume data's Nyquist frequency) this method produces a correctly shaded output image.

Implementing LMIP is more complex, since it requires access to some of *VTK*'s internal data structures and might involve changes to the volume mapper class. Performance optimizations by using a cache-coherent storage scheme, as proposed by Mrotz at al. [12], are also more invasive and might conflict with *VTK*'s architecture.

5 Conclusion

The MIP algorithm and its problems have been described. Various extensions to overcome these limitations have been introduced. It

was shown that *VTK*'s limited MIP functionality can be easily extended to use more advanced methods. For using volume rendering techniques other than ray casting, such as shear-warp factorization, new class structures can be built on vtk's visualization model.

However, since *VTK* is a multi-purpose toolkit its performance is quite bad compared to specially optimized algorithms, which provide real-time frame rates even on low-end systems. For application with demand for high performance, these algorithms are clearly more suitable.

In conclusion, *VTK* trades in optimized performance for high versatility. Still, its extensibility provides the option of using *VTK*'s framework to build customized algorithms for high performance applications.

References

[1] Harvey E. Cline, William E. Lorensen, Sigwalt Ludke, Carl R. Crawford, and Bruce C. Teeter. Two algorithms for the reconstruction of surfaces from tomograms. *Medical Physics*, 15(3):320–327, June 1988.

[2] Erich Hartmann. A marching method for the triangulation of surfaces. *The Visual Computer*, 14(3):95–108, 1998.

[3] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[4] L. Mroz, H. Hauser, and E. Gröller. Interactive high-quality maximum intensity projection. In Sabine Coquillart and Jr. Duke, David, editors, *Proceedings of the 21th European Conference on Computer Graphics (Eurographics-00)*, volume 19, 3 of *Computer Graphics Forum*, pages 341–350, Cambridge, August 21–25 2000. Blackwell Publishers.

[5] Sato Y, Shiraga N, Nakajima S, Tamura S, and Kikinis R. Local maximum intensity projection (lmip): a new rendering method for vascular visualization. *J Comput Assist Tomogr*, 22(6):912–7, 1998.

[6] Georgios Sakas, Marcus Grimm, and Alexandros Savopoulos. Optimized maximum intensity projection (MIP). In Patrick M. Hanrahan and Werner Purgathofer, editors, *Rendering Techniques '95*, Eurographics, pages 51–63. Springer-Verlag Wien New York, 1995.

[7] Karel J. Zuiderveld, Anton H. J. Koning, and M.. A. Viergever. Techniques for speeding up high-quality perspective Maximin Intensity Projection. *Pattern Recognition Letters*, 15:507–517, May 1994.

[8] Wolfgang Heidrich, Michael McCool, and John Stevens. Interactive maximum projection volume rendering. In Gregory M. Nielson and Deborah Silver, editors, *Visualization '95*, pages 11–18. IEEE, IEEE Computer Society Press, 1995.

[9] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings,*, Annual Conference Series, pages 251–260, Los Angeles, 1999. ACM Siggraph, Addison Wesley Longman.

[10] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear–warp factorization of the viewing transformation. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 451–458. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[11] Wenli Cai and Georgios Sakas. Maximum intensity projection using splatting in sheared object space. *Computer Graphics Forum*, 17(3), September 1998.

[12] Lukas Mroz, Andreas König, and Eduard Gröller. Maximum intensity projection at warp speed. *Computers and Graphics*, 24(3):343–352, June 2000.

[13] William J. Schroeder, Lisa S. Avila, Kenneth M. Martin, William A. Hoffmann, and C. Charles Law. *The Visualization Toolkit User's Guide*. Kitware, first edition, 1998. This is the book to get if you want to learn how to use VTK.

Program 4.1 Setting up a MIP in *VTK* (Java code extract)

```
(...)

// Create and initialize opacity transfer function:
// the whole volume will be displayed opaque
vtkPiecewiseFunction myOpacityTransferFunction =
    new vtkPiecewiseFunction();
myOpacityTransferFunction.AddPoint(0, 1.0);
myOpacityTransferFunction.AddPoint(255, 1.0);

// Create and initialize color transfer function:
// Linear function from scalar value 0 (black) to
// value 255 (white)
vtkColorTransferFunction myColorTransferFunction =
    new vtkColorTransferFunction();
myColorTransferFunction.AddRGBPoint(
    0, 0.0, 0.0, 0.0);
myColorTransferFunction.AddRGBPoint(
    255, 1.0, 1.0, 1.0);

// Create a volume property and set its color
// and opacity transfer functions
vtkVolumeProperty myVolumeProperty =
    new vtkVolumeProperty();
myVolumeProperty.SetColor(
    myColorTransferFunction);
myVolumeProperty.SetScalarOpacity(
    myOpacityTransferFunction);

// Create the ray cast function
vtkVolumeRayCastMIPFunction myVolumeRayCastMIPFunction =
    new vtkVolumeRayCastMIPFunction();

// Create the volume mapper and set its ray cast function
vtkVolumeRayCastMapper myVolumeRayCastMapper =
    new vtkVolumeRayCastMapper();
myVolumeRayCastMapper.SetVolumeRayCastFunction(
    myVolumeRayCastMIPFunction);

// Create a file data source
vtkStructuredPointsReader myStructuredPointsReader =
    new VTKStructuredPointsReader();
myStructuredPointsReader.SetFileName("test.vtk");

// Set the mapper's input the the source's output
vtkVolumeRayCastMapper.SetInput(
    myStructuredPointsReader.GetOutput());

// Create the actual volume object and its mapper
// and property
vtkVolume myVolume = new vtkVolume();
volume.SetMapper(myVolumeRayCastMapper);
volume.SetProperty(myVolumeProperty);

(...)
```

Program 4.2 Ray bounding using a shpere data source (Java code extract)

```
(...) {
vtkSphereSource mySphere = new vtkSphereSource();

vtkProjectedPolyDataRayBounder myRayBounder =
    new vtkProjectedPolyDataRayBounder();

myVolumeRayCastMapper.SetRayBounder(myRayBounder);

(...)
```

Program 4.3 Modified version of *VTK*'s `CastMaxScalarValue` function in `vtkVolumeRayCastMIPFunction` to achieve depth-shaded MIP, changes are depicted in **bold face** (C++ code extract)

```
static void CastMaxScalarValueRay( T *data_ptr,
                                   VTKVRCDynamicInfo
                                        *dynamicInfo,
                                   VTKVRCStaticInfo
                                        *staticInfo,
                                   float ambient,
                                   float changeRate)
{

    (...)


    // Variable to hold the depth-shaded value
    float shadedValue;

    (...)

    // For each step along the ray
    for ( loop = 1; loop < num_steps; loop++ )
      {
      // Access the value at this voxel location
      nnValue = *(data_ptr + voxel[2] * zinc +
                  voxel[1] * yinc + voxel[0] );


      // Depth-shading of the value before maximum operation
      shadedValue = nnValue *
                    ( ambient +
                    (1.0f - (loop - 1.0f) / num_steps) *
                    changeRate);

      // If this is greater than the max, this is the
      // new max.
      if ( shadedValue > nnMax )
        {
        nnMax = shadedValue;
        }

      // Increment our position and compute our voxel
      // location
      ray_position[0] += ray_increment[0];
      ray_position[1] += ray_increment[1];
      ray_position[2] += ray_increment[2];
      voxel[0] = vtkRoundFuncMacro( ray_position[0] );
      voxel[1] = vtkRoundFuncMacro( ray_position[1] );
      voxel[2] = vtkRoundFuncMacro( ray_position[2] );
      }

      (...)

}
```