3D Data Processing
Volume Visualization
3D File Format

Miloš ŠRÁMEK, Leonid I. DIMITROV, Matúš STRAKA and Michal ČERVEŇANSKÝ *

THE `F3D` TOOLS FOR PROCESSING AND VISUALIZATION OF VOLUMETRIC DATA

In this paper we introduce the `f3d` format for storage of volumetric data together with a suite of tools for processing, segmentation and visualization of such data. Both the format and tools were developed for a highly variable and rapidly evolving academic environment, where new data processing and visualization tasks emerge very often. The tools address all the steps of a volume visualization pipeline: starting with import of external formats, over preprocessing, filtering, segmentation to interactive visualization.

1 INTRODUCTION

Volume visualization is a complex task which consists of several consecutive steps, such as data preprocessing, filtering, segmentation and, finally, rendering. Although specialized systems and automatic applications exist optimized for processing of data with special properties, a researcher in an academic environment often encounters a situation, when it is necessary to process an unknown data set in a very specific manner. Such demands can be defined, for example, by other researchers or in debugging a data processing algorithm. In such a situation, a set of simple-to-use flexible general-purpose tools for processing of volumetric data may come in handy.

In this paper, we describe one such suite of tools, which was built at the Commission for Scientific Visualization of the Austrian Academy of Sciences around the `f3d` data file format. The suite consists of a set of command-line programs which accomplish different preprocessing tasks (Section 2), together with interactive applications for 3D data segmentation (Section 3) and visualization (Section 4).

All these tools are single purpose ones, bound together by the common data format, the `f3d`. However, by a proper tool selection and ordering according to the demands of the given task, one can often achieve very high goals.

2 THE `F3D` FORMAT

At the core of the utilities described in this paper stands the `f3d` format (`f`ormat `3` `d`imensional, [1]) which is, as we hope, both simple and versatile enough to address the demands of labs, groups or individual researchers on representation and manipulation of Cartesian, regular and rectilinear grid

*Commission for Scientific Visualization (VISKOM), Austrian Academy of Sciences, Vienna, Austria. `milos.sramek@oeaw.ac.at`

data. Currently, the format supports properties, which we found to be important from the point of view of our experience. However, `f3d` is distributed as free software, which hopefully will encourage other contributors to add their own features, extensions and tools.

The `f3d` implementation provides a set of simple and flexible C language routines for reading and writing files, which provides for a steep learning curve for novices and enables easy porting to existing applications. Additionally, a set of tools based on a high-level C++ class library is provided for format conversions, geometric transforms, filtering and other operations with volumetric data. These tools are simple and flexible Unix-like filters, which can be concatenated by piping in order to achieve the desired effects:

```
tool1 [sw] in.f3d | tool2 [sw] | ....  | toolN [sw] > out.f3d,
```

where `sw` stands for optional switches. The name of the input file is an optional argument. If not used, the data are read from the standard input. The output data are always written to the standard output.

Format conversion routines are necessary both to enable the import of data in different formats into the `f3d` environment and, occasionally, to export `f3d` data to other environments (although porting of different applications to read/write `f3d` data is quite simple—unfortunately, one needs access to the application's source code, which is not always possible, since many authors tend to hide their code). Currently, input and output filters for conversion of 3D raw data, stacks of `pnm` images and 16-bit slices exist, together with routines for reading the VTK and DICOM formats.

For example, the following command should be used to convert a $256 \times 256 \times 128$ raw data set of 32 bit voxels (the `-l 5` switch) with reversed byte (the `-r` switch) order:

```
raw2f3d -i in.raw -r -l 5 -s 256 256 128 > out.f3d,
```

For any tool, the `-h` or `--help` switch prints a brief description and a list of its switches.

Point operators are programs, which modify values of individual voxels independently of their neighbors. The `f3d2f3d` program converts the type of voxels (e.g. 8 bit integers to floats) and can be also used to change data contrast. Further, `f3dinvert` inverts the image, `f3dthresh` thresholds the data by upper and lower thresholds and `f3dbit` combines each voxel with a specified pattern (operations as addition, subtraction, multiplication, and, or, max and min are possible). The `f3darith` does a similar thing, but between corresponding voxels of two different volumes. A binary mask (e.g., created by the `f3dthreshold` tool) can be used by `f3dmask` to delete (set to 0) large areas of a volume. For example, bone can be deleted from a volume "`in.f3d`" by

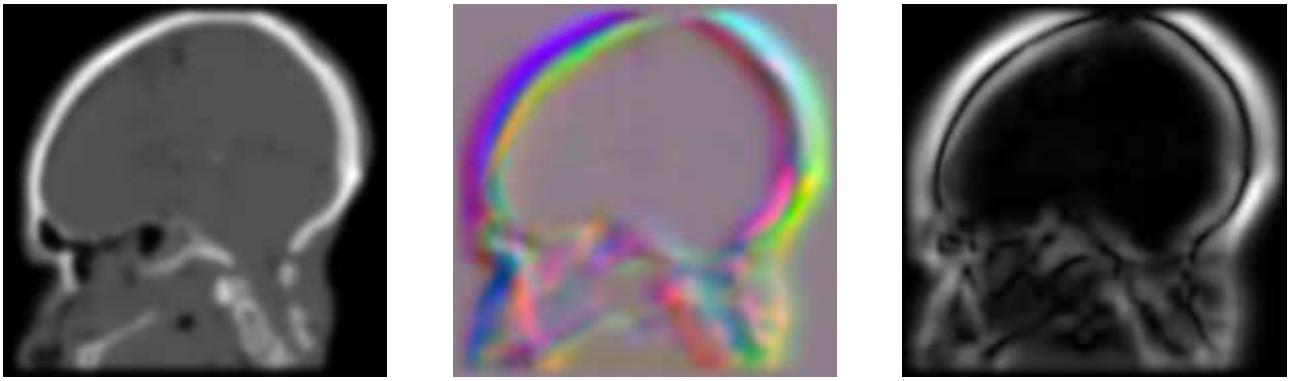
```
f3dthresh -lo 90 in.f3d | f3dmask -i in.f3d > out.f3d.
```

Volume filters are filters in the sense of the signal processing theory. They implement local operations on the input data set, i.e., the output voxel value is computed from the values of voxels in a local neighborhood of the corresponding input voxel. The size of the neighborhood depends on the given operation and can span the range from few to many voxels. The following programs fall in this category: `f3dmax`, `f3dmin` and `f3dmedian` take maximum, minimum and median value of a 6-, 18- or 26-neighborhood respectively, `f3dgauss` smoothes the data by the Gaussian filter, (Fig 1a), `f3dgabor` computes a smoothed partial derivative of the image by means of the Gabor filter (derivative of the Gaussian) `f3dgrad` computes gradient of the data (the Gabor filter applied in all 3 directions, the output data has 3 bands, Fig 1b), `f3dgradmag` computes magnitude of the Gabor gradient (Fig 1c) and, finally, `f3dprewitt` detects edges by the Prewitt filter.

Certain operations (e.g., the `f3dgabor` and `f3dgradmag` filters) should not be applied to unsigned data types or types with low precision (8 or 16 bits), either because subtraction of such variables may lead to erroneous results (always nonnegative) or truncation errors deteriorate the result. Therefore, such types should be first converted to signed types with higher precision, preferably to float. E.g., in order to enhance edges of an 8 bit data volume one has to use the following command:

```
f3d2f3d -d 7 in.f3d | f3dgradmag -w 3 > out.f3d,
```

where the `-d 7` switch of the `f3d2f3d` program converts voxels to floats and the `-w 3` switch defines the sigma value of the filter kernel.



(a) (b) (c)

Figure 1: *A slice of a 3D dataset: (a) Gaussian smoothing by `f3dgauss`, (b) gradients computed by `f3dgrad` and (c) edge enhancement by `f3dgradmag`.*

Global operators are represented by tools which modify the whole volume and eventually even change its dimensions. `f3diso` resamples non-Cartesian grids to cubic voxels with the possibility to select nearest neighbor, trilinear or tricubic interpolation. `f3dcrop` can be used to delete outside layers of voxels, while the `f3dpaste` tool can insert one volume into another, with different possibilities to combine their voxels (see the `f3darith` tool). `f3dpaste` can be also used to merge bands of two different files, or, with the help of `f3dnew`, to pad a volume with a layer of voxels.

The `f3drot` and `f3dtrans` programs implement geometric transforms of 3D data sets. While the first one can be used only to rotate a volume around the coordinate axes in 90 degree multiples, in `f3dtrans` all combinations of arbitrary rotations, translations and scalings can be used. Further, similarly to `f3diso`, three types of interpolation techniques can be defined. The `f3ddist`, `f3dlabel` and `f3dffill` work with binary input data, where 0 represents background and nonzero represents objects. `f3ddist` computes distance transforms, both signed (negative distances are assigned to object interiors) and unsigned. Three types of distances can be computed: chessboard, cityblock and the 3-4-5 chamfer distance (Figure 2). As we can see in the figure, these types of distance transforms only roughly approximate the Euclidean distance. A much higher precision can be obtained by the `f3dfmm` tool, which computes the distance by solving the Eikonal equation by the fast marching technique [2]. By means of distance transforms and thresholding one can efficiently implement different morphologic operations (dilation, erosion, opening and closing), even with large structuring elements. For example,

```
f3dthresh -lo 90 in.f3d | f3ddist -chu | f3dthresh -lo 5>out.f3d
```

dilates an object identified by thresholding at level 90, while

```
f3dthresh -lo 90 in.f3d | f3ddist -chu | f3dthresh -lo 5 |
                  | f3ddist -chu | f3dthresh -lo 5>out.f3d
```

computes closing thereof (Figure 3). Here, the unsigned 3-4-5 chamfer distance was used (the `-chu` switch) with a size 5 structuring element. The `f3dlabel` scans the volume and labels the background and all the objects by unique labels, while the `f3dffill` can either keep or delete (the `-d` switch) a single region identified by a seed point.

The 3D fast Fourier transform is implemented in `f3dfft`. In combination with masks produced by the `vxt` tools it enables the implementation of various filtering schemes with large filtering kernels. `vxt` is another package working with the `f3d` format, aimed at voxelization of geometric objects [3].

(a) (b) (c)

Figure 2: *A slice of the 3D distance transform of a skull (identified by thresholding at level 90 by means of `f3dthresh`: (a) chessboard, (b) cityblock and (c) 3-4-5 chamfer distance.*

(a) (b) (c)

Figure 3: *Morphologic operations: (b) dilation of (a) using distance transforms, equivalent to using a* $5 \times 5 \times 5$ *approximation of a spherical structuring element. (c) closing of (a) using the same parameters.*

The Watershed Transform: the aim of the watershed based image segmentation technique is to split images into spatially homogeneous regions, which can be further processed by different image analysis tools [4, 5]. The watershed segmentation technique is based on the interpretation of an image

as a topographic relief and on the simulation of flow of water along steepest descent paths called downstreams. Thus, for each local minimum of the image, a drainage region is defined, which, if computed for a gradient image, represents an area with approximately constant properties.

The computation of the watershed transform in this case consists of 3 steps: it is necessary to compute the gradient magnitude of the input volume, to identify the local minima, and to label them by unique labels:

```
f3dgradmag -w 3.0 in.f3d | tee grad.f3d |
   | f3dlocmin | f3d2f3d -d 5 | f3dlabel > labels.f3d
```

The `f3d2f3d -d 5` changes the volume representation to the `int` type, since the expected number of regions (minima) is very high. The gradient magnitude volume will be used in the second step and therefore it is stored in an auxiliary file `grad.f3d` by the standard Unix `tee` command. In the second step, the `f3dwatershed` program computes the watershed transform and sends the result, where each watershed region is assigned the same label as the corresponding local minimum, to its standard output. In this form it can be used by other programs for further analysis [5]. In order to show the results in a form suitable for viewing by a human, is is useful to detect the edges of the watershed region (`f3dedge`) and overlay them over the original image by pasting the edge volume into a new band (the `-b` switch) of the original:

```
f3dwatershed -m labels.f3d grad.f3d | f3dedge |
   | f3dpaste -b in.f3d > out.f3d
```

(a) (b) (c)

Figure 4: *The watershed transform: (a) a slice of the original data set, (b) detected regions and (c) region edges overlayed over the original.*

3 3D DATA SEGMENTATION BY `ISEG`

Segmentation of volumetric data sets is often a prerequisite for the subsequent visualization step. Here, the scope of segmentation tasks extends from simple ones, as e.g. recognition of the whole object in the background or segmentation of the bone, which has much higher density that soft tissue in CT tomograms, to tasks, which are even hard to formulate as e.g. the segmentation of small brain structures in MR images.

The interactive segmentation method implemented in `iseg` is based on thresholding and repeated application of morphologic and connected component labeling operations. Since different tissues can have densities from overlapping intervals, a situation when more than the desired object is marked is possible. Separation of such regions can be done on the presumption of *object connectivity*. The connectivity analysis can be implemented by a region filling algorithm (say *flood fill*). Since connectivity is a spatial property, which means that we cannot decide whether the given voxel belongs to the object or not in one slice, a 3D version of this algorithm is used.

The backbone of the segmentation program consists of three 3D buffers: an input buffer for the original data, a work buffer and an output buffer for storing of up to 256 non-overlapping objects. The work buffer is used for the morphological, labeling and manual editing operations as well as for logical operations between objects. The user has the possibility to view the contents of the buffers directly (3 consecutive slices or 3 mutually perpendicular cuts) and to render either some or all objects stored in the output buffer (6 orthographic projections).

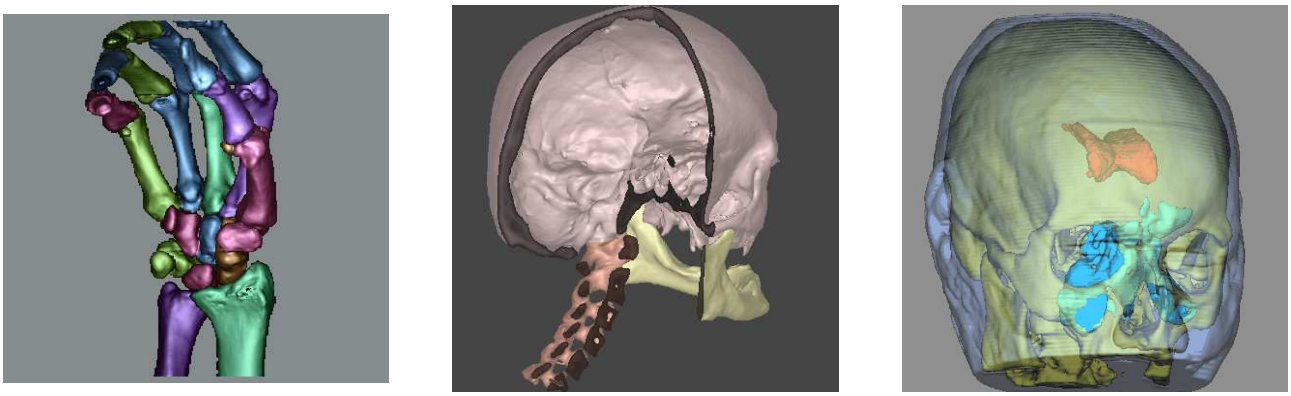
4 3D DATA VISUALIZATION

`fd3view`: A Slice Viewer. The `f3dview` viewer serves as the basic tool for visualization of `f3d` data. It is a slice viewer, i.e. it enables the display of only orthogonal 2D cuts through the data. However, in spite of this limitation it is the most useful visualization tool in development of 3D data processing applications, since it enables the access of individual voxels, which helps the developer in debugging a misbehaving code. The `f3dview` viewer, besides its basic functionality, allows the user to modify the data contrast via lookup tables in order to improve the visibility of details of interest. The zoom-in function is not implemented, since it can be acieved easily by external tools as, e.g., `xmag`.

`f3drender`: A simple command-line renderer. The `f3drender` tool is the simplest of the 3D rendering programs working with the `f3d` data. It is a command line program which can render data only in 6 axis aligned directions with no magnification—the rendered image has exactly the same size as the corresponding volume crossection—which can be used with advantage in data inspection and algorithm testing. Several rendering modes are supported: reprojection, maximum/minimum intensity projection and surface rendering of thresholded and segmented data.

`vrnd`: Software-Based Rendering of Segmented Data. `vrnd` is a versatile interactive renderer based on volumetric ray tracing with a text command interface and scripting possibilities. It is predominantly oriented towards visualization of segmented data (the `iseg` segmentator can even output a `vrnd` script tailored to the actual data and therein segmented objects) and implements several surface and volume rendering techniques (Fig. 5). To its numerous features belong different volume and surface interpolation modes (voxel/subvoxel rendering), different normal estimation techniques (graylevel gradients, centroidal shading of binary data[1], shadows, reflections, multiple lights, shaded cutplanes and parallel and perspective rendering. In `vrnd` up to 64 different objects can be defined, each with its own color, transparency and a set of mutually orthogonal cut planes. For all volume rendering modes (shaded/unshaded blending, maximum intensity projection) piecewise linear lookup tables are used to set color and transparency of the data.

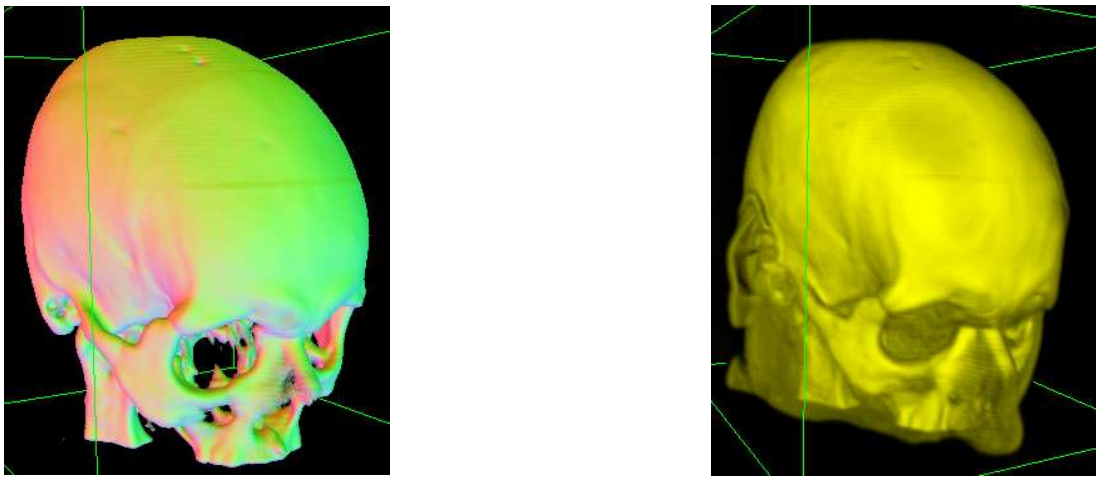
The `vrnd` tool can be controlled by a simple command line interface. However, it is also equipped with scripting possibilities, which enable the rendering of animations with hundreds of images easily.



(a) (b) (c)

Figure 5: *Examples of tomographic data segmented by* `iseg` *and rendered by* `vrnd`*: (a) skeleton of a human hand, all bones were identified and labeled, (b) a skull with cut planes, (c) a human head with different objects identified and rendered with transparency.*

`fdvr`: Interactive Rendering by Texture Mapping. The last in the list of `f3d` data visualization tools, the `fd3vr`, is an interactive hardware-based tool, which takes advantage of the capabilities provided by the modern programmable graphics accelerators [6]. Different techniques are implemented—2D and 3D texture-based rendering, color and transparency lookup tables and fragment program based illumination, shading and classification (Fig. 6). As such, `fd3vr` serves two purposes. Firstly, it is a data exploration tool, which can be used for visualization of any kind of scalar volumetric data. Secondly, due to the large number of implemented techniques, it is used to demonstrate different volume visualization techniques in education.



(a) (b)

Figure 6: *Shaded images rendered by* `fd3vr`*. (a) colors modulated by gradient vector components, (b) shaded surfaces enhanced by gradient magnitude. Cg programming was used to obtain these effects.*

[1] In a $3 \times 3 \times 3$ or $5 \times 5 \times 5$ voxel neighborhood centroids C_o of object voxels and C_b of background voxels are computed. Then the surface normal is estimated as $||C_b - C_o||$.

5 Conclusion and Future Work

We have introduced the the `f3d` suite for processing, segmentation and visualization of 3D data. Although most of the examples were taken from the area of medical imaging, the presented tools are by no means specialized and can be used to accomplish similar tasks with data coming from arbitrary sources.

Not all existing tools working with `f3d` data were described here. The `vxt` package allows to voxelize (i.e., to convert from analytic to volumetric form) different kinds of geometric objects [3]. Such volumetric object models can be subsequently simply used for object rendering, or can play the role of artificial phantoms in development and testing of new algorithms for 3D data processing. The `f3d` format and the `f3d` C++ class library are used in the core of the AngioVis software, which is aimed at visualization and evaluation of pathological lower extremity vessels by means of CT Angiography data [7]. Another rendering package, `VORTEX` [8], is aimed at visualization and mapping of unstructured sampled data onto surfaces defined by 3D grids. Further, the `f3d` suite is used in education at universities in Bratislava, Slovakia and Vienna, Austria. Btw, some tools were written as student projects and master theses.

The `f3d` suite is free software [2], and as such can be used by everybody and everybody has access to its source code[3]. In this sense, we invite anybody to cooperate by using our software, reporting bugs, improving it or even creating new tools.

BIBLIOGRAPHY

[1] M. Šrámek and L. I. Dimitrov. `f3d` —a file format and tools for storage and manipulation of volumetric data sets. In G. M. Cortelazzo and C. Guerra, editors, *1st International Symposium on 3D Data Processing, Visualization and Transmission*, pages 368–371. IEEE Computer Society, Padova, Italy, June 2002.

[2] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[3] M. Šrámek and A. Kaufman. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):251–266, 1999.

[4] M. Šrámek and L. I. Dimitrov. Segmentation of tomographic data by hierarchical watershed transform. *Journal of Medical Informatics and Technologies*, 3:MI–161–MI–169, 2002.

[5] M. Straka, A. La Cruz, A. Köchl, M. Šrámek, E. Gröller, and D. Fleischmann. 3D Watershed Transform Combined with a Probabilistic Atlas for Medical Image Segmentation. *Journal of Medical Informatics & Technologies*, 6:IT 69–IT 78, 2003.

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

[7] M. Straka, M. Cervenansky, A. La Cruz, A. Köchl, M. Šrámek, E. Gröller, and D. Fleischmann. The vesselglyph: Focus & context visualization in CT-angiography. In *Proceedings of the Visualization'04 Conference*, October 2004. accepted.

[8] L.I. Dimitrov. Texturing 3D-reconstructions of the human brain with EEG-activity maps. *Human Brain Mapping*, 6(4):–, 1998.

[2] http://www.fsf.org, http://www.opensource.org

[3] http://www.viskom.oeaw.ac.at/~milos