VolumeFlies: Illustrative Volume Visualization using Particles

Stef Busking[1,2] Anna Vilanova[1] Jarke J. van Wijk[1]
[1]Technische Universiteit Eindhoven
[2]Technische Universiteit Delft
S.Busking@tudelft.nl A.Vilanova@tue.nl J.J.v.Wijk@tue.nl

Keywords: visualization, non-photorealistic rendering, volume rendering, particle systems

Abstract

Non-photorealistic (NPR) techniques are usually applied to produce stylistic renderings. However, they can also be useful to visualize scientific datasets. NPR techniques are often able to simplify data, producing clearer images than traditional photorealistic methods. We propose a framework for visualizing volume datasets using non-photorealistic techniques. Our framework is based on particle systems, with user-selectable rules affecting properties of the particles such as position and appearance. The techniques presented do not require the generation of explicit intermediary surfaces. Furthermore, the framework is versatile enough to produce a variety of illustrative techniques within the same framework.

1 Introduction

A common problem in visualization research is the visualization of large 3D volumetric datasets. While photorealistic approaches are commonly used, the use of non-photorealistic techniques may provide more insight by creating clearer images. Non-photorealistic rendering (NPR) focuses on increasing the expressiveness of computer graphics by incorporating techniques adapted from traditional art and illustration. By removing unimportant details and visual clutter, the viewer's attention can be directed towards the most important aspects of an image. Alternatively, photorealistic techniques can be combined with NPR methods to show both focus and context respectively in a visualization.

Although the techniques presented in this paper are of general use for volume data, the motivation of this work is to be applied in medical data sets. In figure 1 we have used different non-photorealistic rendering techniques to show both focus (bone) and context (skin and arteries). The sparseness of the selected pen-and-ink styles provides a good alternative to the translucency commonly used in methods like direct volume rendering. The image was generated at interactive speeds, enabling easy exploration of the data.

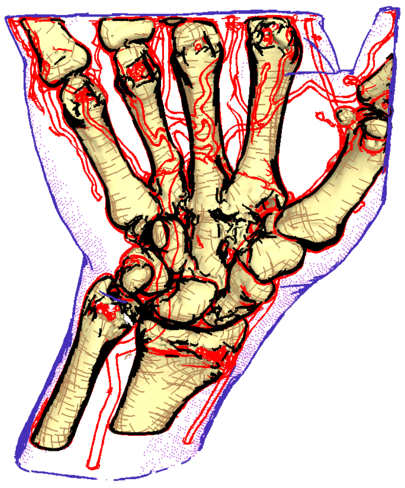


Figure 1: Illustrative volume visualization of a CT hand dataset, combining several NPR techniques.

We propose a framework for visualizing volume data, based on particle systems that operate directly on the dataset. Different non-photorealistic visualization techniques can be implemented within this framework by using different rules that affect properties of the particles such as their position and appearance. These rules use only a small subset of data local to a particle's position, which results in a scalable system that is largely independent of the resolution of the underlying dataset. We aim to show that a particle-based approach to volume visualization results in a flexible system and a unified way of describing various NPR techniques.

In this paper, we first give an overview of research related to our work. Next we present our VolumeFlies framework and shortly discuss the pre-processing steps common to all visualizations. We then present a new hidden surface removal algorithm for particles, followed by the various non-photorealistic techniques that have been implemented in our framework. Finally, we present and discuss results obtained with our prototype implementation of the framework.

2 Related work

Most NPR research focuses on imitating artistic styles and techniques. Lately, however, more and more researchers are exploring possibilities for using NPR for scientific visualization. This is because NPR techniques are suitable for emphasizing certain features or properties while omitting or greatly simplifying other, less important details.

Most research in the area of NPR-based volume visualization either takes an approach based on direct volume rendering (DVR) (e.g., [1]), or applies existing surface-based NPR techniques to polygonal (or otherwise explicitly geometrically defined) surfaces, such as iso-surfaces, extracted from a volume dataset. For example, Interrante [2] recommended strokes oriented in the directions of principal curvature to illustrate the local shape of transparent surfaces. Both approaches have problems however, as most graphics hardware is more suited to dealing with geometric primitives rather than DVR, while working only on extracted surfaces creates a level of abstraction which may lead to loss of information and additional processing costs.

Several researchers have produced NPR-based visualizations directly from volume data. Lu et al. [3] presented a method for previewing medical volume data using a stippling method. The number of stipples drawn in each voxel was carefully adjusted to control density and shading, and to enhance features such as boundaries and silhouettes. The resulting system requires little user interaction and is suitable for quickly previewing volume datasets. Unfortunately, the images often look very noisy, partly due to the lack of hidden-surface removal. Details are hardly visible, and often require parts of the dataset to be visualized using different methods in order to examine them.

Both Nagy et al. [4] and Dong et al. [5] presented methods for generating and rendering hatches in volumetric datasets. The method by Nagy et al. combines hatches with direct volume rendering, in order to enhance these features in the volume. Due to the low density of seed points and the use of long hatches, the resulting images resemble quick sketches. Dong et al., on the other hand, use only hatches and contour lines. Both sketch-like and densely hatched images can be created using their method. However, due to the complexity of the algorithms involved, the process runs at speeds far from interactive, with images taking several minutes to generate and render.

Cornish et al. [6] proposed a novel NPR visualization method for surfaces using a hierarchy of particles, derived from and optionally combined with the polygonal surface. This hierarchy was expanded or collapsed based on the view direction and other parameters such as lighting conditions. By visualizing the particles not only as stipples but also using oriented textured splats, the surfaces could be shown with a number of NPR techniques at interactive speeds.

Our aim is to develop a framework to produce NPR-based visualizations with the speed, versatility and ease-of-use of the method by Cornish et al., but directly from the voxel data. Our framework is inspired by the *Smart Particle* concept by Pang and Smith [7]. This is a combination of particle systems and behavioural animation, allowing particles to be programmed to actively seek out and visualize specific features in a dataset.

3 The VolumeFlies framework

The VolumeFlies framework is based on sets of particles existing within the space of the volume dataset. These particles are controlled by applying specific rules, which affect properties of the particles such as their position or their appearance.

The framework, shown in figure 2, consists of a four-stage pipeline. In the feature location stage, the features that we want to visualize are located and particles are created at locations on those features. Once the features (e.g., some type of surfaces) have been located, the particle manipulation stage prepares the particles for visualization. For instance, the particles will often have to be redistributed in order to cover the entire feature. Afterwards, the filtering stage can be applied. The particles in the system can be filtered based on some criteria, for example, removing particles located at hidden surfaces. Finally, in the rendering stage, geometry is created for each of the remaining particles in order to achieve a desired visual style. Multiple instances of the framework can be used in

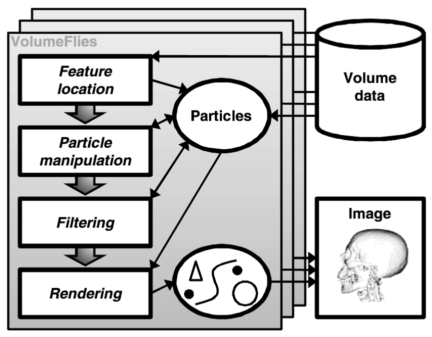


Figure 2: The VolumeFlies framework

a visualization. The geometry resulting from each of these instances is then sent to the rendering pipeline to be projected onto the final image.

Each of these stages gives rise to a class of pluggable modules implementing a set of rules to be applied to the particles. These can be used as building blocks, to adapt the framework to a specific visualization scenario. The first two stages are typically performed in a pre-processing step. During rendering, the modules selected for the third and fourth stages are invoked repeatedly for each change in the viewing direction and / or other visualization parameters.

In the next sections we present several variations for the modules of the framework. An important property of the modules discussed in this paper is that the rules applied to each particle use only information local to that particle's position. This improves scalability of the system to larger datasets. In some modules information about the particle's neighbours is required. In this case we use a spatial binning algorithm for fast access to these neighbours.

4 Position of the particles

The first step is to place a set of particles at some initial position on the features we want to visualize. In medical volume data, the most commonly visualized features are iso-surfaces. Therefore we chose iso-surfaces as the first implementation for the feature location module.

We sample the dataset at a user-defined grid, using interpolation between data points where necessary. In this way we can ensure regular sampling in all directions even if the voxels in the original dataset are not isotropic. Moreover, sampling at a user-defined grid gives the user control over the density of particles created during the feature location stage.

We identify the location of the iso-surface by comparing the value at each grid point to its direct neighbours. If these values lie on opposite sides of the iso-surface value, the surface must intersect the line between the two points. Linear interpolation is used to approximate the location of this intersection, and a particle is created at that location.

This method usually results in an uneven distribution of particles over the feature surfaces, causing visual artifacts such as the rings in figure 3(a). To solve this, we use a particle distribution method originally developed by Witkin and Heckbert [8], and later improved by Meyer et al. [9] to redistribute the particles over the surface. This is accomplished by having particles locally repel each other, but constraining them to the surface. We adapt Meyer's improved algorithm from the context of rendering implicit surfaces to that of volume datasets and iso-surfaces.

In some situations, creating an even distribution of particles (as shown in figure 3(b)) is not as important, and the goal is merely to remove the patterns created by the feature location algorithm step. In these situations simply moving the particles in random directions over the surface for a number of steps will often produce acceptable results, and the more computationally intensive distribution method may be avoided.

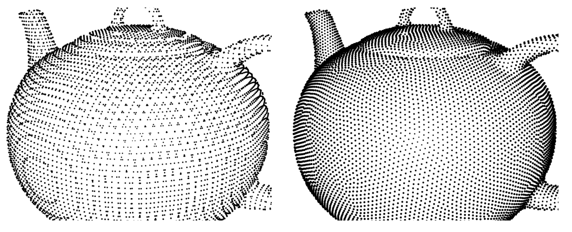

(a) before (b) after

Figure 3: Particles before and after redistribution

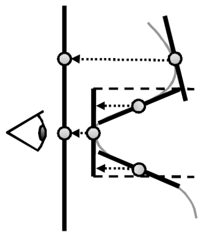
5 Hidden-surface removal

The most obvious issue when dealing with surfaces illustrated by separate particles rather than a complete surface (e.g., a polygonal representation) is that there is no occlusion between the rendered surfaces. While in some cases this may provide additional insight into the structure of the feature, it could clutter the image in other cases. Particles on hidden surfaces should be detected and (optionally) removed in the filtering stage of the framework.

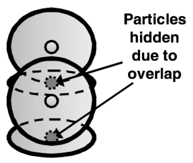
Most surface-based methods solve this problem by first rendering the polygonal representation of the surface to a depth buffer, and subsequently testing each of the particles against this buffer to determine their visibility. The polygonal surface can also simply be rendered using the background colour in order to erase any particles that should not be visible.

An alternative technique, used in [10], first renders the polygonal surface in uniquely coloured patches. Particles are only deemed to be visible if the colour of their corresponding patch is found in the resulting image. While this scanning approach may be slower than the depth-buffer based approach, it has the advantage that the set of visible particles can be determined before they are rendered. This makes it easier to combine different visualizations (sets of particles) in the same image. Also, using this method avoids depth-value precision issues, a common problem with the first approach.

We show that hidden particles can also be removed without an explicit construction of polygonal representations of the surfaces. A common technique for rendering surfaces from separate points (point-based rendering, see for example [11]) is *splatting*. In this technique, discs aligned with the surface are drawn at the position of each of the points. If enough discs are used and the discs are large enough, this results in an



(a) Splats on surface (side view)



(b) Resulting splat image

Figure 4: Overlap between neighbouring splats can cause visible particles to disappear.

approximation of the surface.

If the surface is strongly convex, disks can undesirably mask neighbouring particles as shown in figure 4. This may cause gaps to appear in the surface. Using a scanning approach, this does not matter as long as some part of the discs for these particles is visible. However, in some cases the complete disc is occluded erroneously. This is especially an issue if the splatting image is generated at a lower resolution than the final image, in order to increase performance.

Because of the advantages mentioned above, we use the scanning approach for detecting visible particles. Our goal is therefore to find an algorithm that minimizes overlap problems between neighbours and still works well at reduced resolutions. Our solution is to use cones oriented towards the viewer rather than circular disks. The cones are scaled at their base to match the projection of the original discs (see [12] for details). The 3D nature of their shapes leads to more evenly sized projections for each particle (figure 5). In fact, when the surface is parallel to the screen, the resulting image is a Voronoi diagram of the set of particles. We call this new method *cone-splatting*.

Two parameters control the size of the cones. The *radius* needs to be large enough to create a closed surface in the projection. However, it should not be larger than the distance between neighbouring particles, as this could cause cones to stick out from the surface. Increasing the *length* of the cones increases the robustness to high-curvature areas such as shown in figure 4. However, if the cones are too long, particles behind the frontmost surface may become visible undesirably.

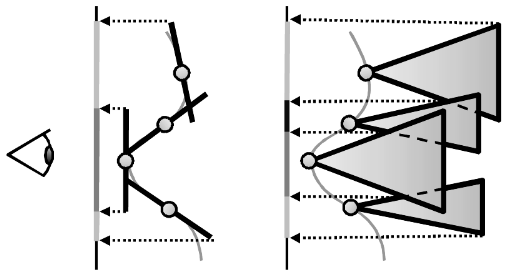


Figure 5: Cone splatting compared to normal splats

6 Rendering

Traditional medical illustrations commonly use pen-and-ink styles. Common elements in these styles include stippling, hatching and contours. We adapt existing techniques and introduce new algorithms to emulate these styles using our framework.

6.1 Stippling

The simplest way to visualize a set of particles is by using point primitives. The set of particles provides us with a set of positions in 3D space, which can be projected onto the image plane using any desired projection method. Additionally, the surface normal – derived from the local gradient – can be used for applying shading, to better illustrate the shape of the surface. There are several options.

In traditional illustration, two techniques are typically used to create shading effects in stipple drawings. One is to vary the scale of the points, using larger points to create darker areas (see figure 6(a)). We have implemented this as a rendering module in our framework, by using the value of the lighting equation as a scaling factor for the size of a particle. Assuming the particles are evenly distributed, they will form an approximately hexagonal pattern over the surface (recall figure 3(b)). Further assuming that particles in the immediate neighbourhood of a particle p_i experience similar lighting conditions as p_i and ignoring overlap between particles, we can derive for the radius S_i of the stipple for p_i,

$$S_i = \sigma\sqrt{\frac{\sqrt{3}}{2\pi}\,(1-L_i)}, \tag{1}$$

where σ is the (average) distance between neighbouring particles and L_i is the computed lighting intensity at the position of particle p_i (see [12] for details).

Another method of shading in stipple drawings is to increase or decrease the density of stipples in certain areas in order to achieve darker or lighter tones respectively (figure 6(b)). We assume that the full set of particles is sufficient to generate a black tone. As all particles are of equal size and evenly distributed, the fraction of particles shown is linearly related to the tone. We therefore first assign to each particle p_i a value v_i from a uniformly random distribution ranging between 0 and 1. During rendering, a particle is drawn only if the value of its lighting equation is less than this value.

A disadvantage of this method is that it may require a very dense set of particles in order to create a detailed image. Very large numbers of particles affect performance as well as accuracy. On the other

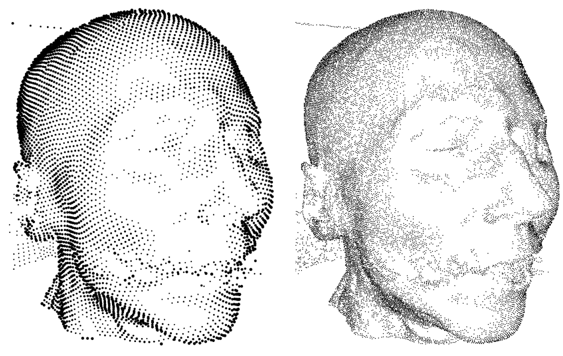

(a) Scale-based shading (b) Density-based shading

Figure 6: Stippling methods

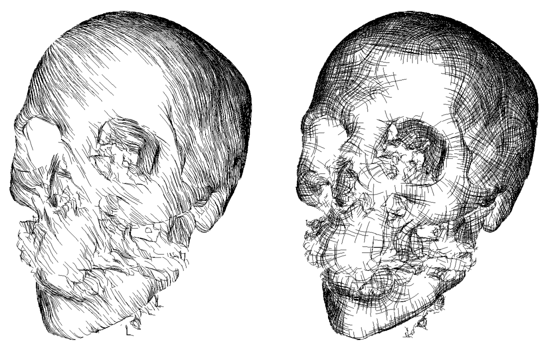

(a) Single direction (b) Principal curvature

Figure 7: Hatching methods

hand, the method is suitable for illustrating surfaces that provide context to a visualization (for example, the skin in figure 1). Transparent surfaces can also be visualized using this method. By placing the light source at the same position as the camera, contours are enhanced while particles are removed from interior areas, reducing clutter.

The results of both stippling methods can be improved further by observing that in traditional illustration very bright areas often contain no points at all. We can achieve this effect by removing points altogether, if their brightness is above a certain threshold. Additionally, other parameters can be added, for instance to alter the contrast in the image.

6.2 Hatching

In hatching, lines and curves are used to create an image. Shading is often accomplished by varying line width and/or spacing. The direction of the strokes is used to illustrate the shape or material properties of the 3D surface that is being represented. In traditional illustration, both normal hatching (closely spaced parallel lines) and cross-hatching (two or more sets of lines that may intersect each other) are used.

In our framework, creating hatched images consists of two steps. First, hatch lines are generated from the set of particles in the particle manipulation stage of the framework. Secondly, during rendering, a shading algorithm decides which of these lines should be drawn in order to create the appearance of a shaded surface. For the purposes of shading, we use the density-based method presented in section 6.1.

We use the positions of the particles as seed points for hatch lines traced through the volume. A direction is selected in the local surface tangent plane, after which the position is updated by moving along that direction for some user-selectable distance and repeating the process until a desired number of hatch-segments has been created. This is repeated in the opposite direction, again starting from the particle's position. In order to be able to perform hidden surface removal on these hatches, each segment of the hatch line is linked to the nearest particle in the volume; a segment is drawn only if its linked particle is marked visible.

6.2.1 Direction of the hatches

A simple method for selecting directions is to take a single direction for all hatches and project this direction onto the local surface tangent plane in order to obtain a hatch direction for each individual particle. Due to the uniformity of the hatches, the resulting images look similar to images created using wood engraving (see figure 7(a)). A disadvantage is that in areas where the preferred direction is perpendicular to the surface the hatch field looks messy, as the hatch direction is not well defined.

An alternative, as suggested by Interrante [2], is to use the directions of principal curvatures. We use the curvature estimation method presented by Kindlmann et al. [13] to compute curvature, but perform eigenanalysis on the resulting geometry tensor to obtain the directions of principal curvature as well as the values.

While the directions of principal curvature work well for hatching on smooth surfaces, the iso-surfaces in real-world datasets (such as medical volume data) are not always smooth and often noisy. In order to obtain reliable derivatives, and to ignore unimportant details on the surface, we blur the dataset using a Gaussian kernel. This allows us to calculate curvature properties at a proper scale. To further improve our results, a smoothing algorithm is subsequently applied to the direction field. Figure 7(b) was generated using this method.

6.2.2 Smoothing the direction field

One remaining problem with principal curvature directions is that they are not well defined in areas where the surface is flat or (nearly) spherical. Moreover, Hertzmann and Zorin [14] have noted (based on hatching patterns in traditional illustration) that hatching using principal curvature directions is most effective in areas that are parabolic. That is, areas where one of the principal curvatures (κ_1 and κ_2) is large while the other is near zero. If one of the curvatures is exactly zero the surface is locally cylindrical.

Based on these observations, we have designed a smoothing algorithm which generates direction fields suitable for hatching. Like Hertzmann and Zorin, we use a cross-field consisting of *unordered* pairs of directions, because there are certain cross-hatching patterns that can not be decomposed into two separate single-direction fields. The field is stored as a pair of direction vectors in each particle and is initialized with principal curvature directions. While tracing a hatch, the field from particles near the current position is averaged to obtain an approximate pair of directions for that point. We then select the direction most like the current hatch direction as the direction in which to continue the hatch line.

We define a measure of *field reliability*, ρ, which essentially states how suitable for hatching the principal directions are at a given point. We base this measure on the *shape index s*, defined by Koenderink and Van Doorn [15],

$$s = \frac{2}{\pi} \arctan \frac{\kappa_2 + \kappa_1}{\kappa_2 - \kappa_1} \quad (\kappa_1 \geq \kappa_2). \tag{2}$$

The shape index is a number between -1 and 1 indicating the shape of the surface. We transform s into our reliability measure by taking $\rho = 1 - |2(|s| - 1/2)|$. The value of ρ ranges from 0 (spherical or saddle-shaped) to 1 (cylindrical). This way, when $\rho = 1$, the principal curvature directions are most suitable for hatching, while $\rho = 0$ means the directions are unreliable. The shape index does not indicate whether a surface is flat, that is, if both κ_1 and κ_2 are (nearly) 0. It is, however, important to detect flat areas as the directions of principal curvature are not well defined in those areas, therefore we set ρ to 0 in these cases.

We iteratively replace the directions in in each particle with the average taken over that particle's neighbourhood, using the values of ρ in each particle as weights. This leads to blurring in areas of low reliability, while reliable directions are preserved. Differences in the orientation of the surface at neighbouring particles may cause the averaged directions to be outside of the surface tangent plane. To prevent this, directions are rotated according to the minimal rotation between the surface normals at the particles before they are averaged.

6.3 Contouring

Contouring is one of the most useful techniques in non-photorealisitic rendering. By tracing the silhouettes of objects, these objects are emphasized in the

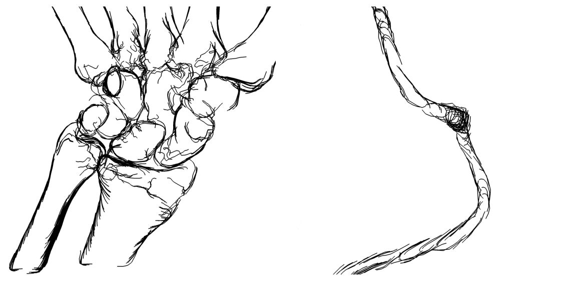

(a) Bones of the hand (b) Blood vessel

Figure 8: Iso-depth contours

visualization without cluttering their interior. Creating contours consists of two steps. First, a subset of particles near the contour is selected. These can be found by placing a threshold on the dot product of surface normal $\vec{n}$ and viewing direction $\vec{e}$. Next, we trace contours in a way similar to hatching. Unlike hatching, tracing of contours can not be performed in a pre-processing step, because contours are dependent on the viewing direction.

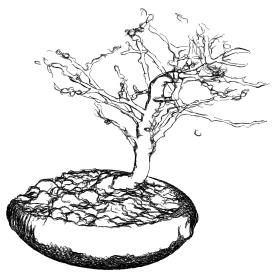
In order to follow the silhouette, the direction of contours should keep the surface normal perpendicular to the viewing direction. One option is to draw iso-depth lines instead of true contours. We obtain these by following the direction of $\vec{n} \times \vec{e}$. This method results in a decent approximation to contours for objects for which the silhouettes lie in or close to planes perpendicular to the viewing direction (figure 8(a)). In areas where this is not the case, the result is often a sketch-like effect (see the lower bones in figure 8(a)). The method fails, however, on small cylindrical structures, such as the blood vessel in figure 8(b). The iso-depth lines can also work as an effective hatching pattern.

In order to obtain more accurate contours (see figure 1), we note that the local surface curvature describes the local behaviour of the normal. It can therefore be used to determine the silhouette direction. By considering a local coordinate frame at point **p** consisting of the principal curvature direction vectors $\vec{k}_1$ and $\vec{k}_2$ combined with the local surface normal $\vec{n}$, and using Rodrigues' formula [16], we derive an approximation for the contour line in this frame (see [12] for the complete derivation). From this we obtain the direction,

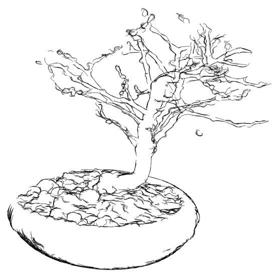
$$\vec{d} = -\kappa_2 \left(\vec{k}_2 \cdot \vec{e}\right) \vec{k}_1 + \kappa_1 \left(\vec{k}_1 \cdot \vec{e}\right) \vec{k}_2. \tag{3}$$

Blurring is applied to improve the robustness to noise of the curvature calculation.

We need to determine which particles to draw contours from. Using a threshold on $\vec{n} \cdot \vec{e}$ results in wide contours in areas of low curvature, while contours in areas of high curvature are smaller or may be missed altogether if no particles are in the contour area (fig-



(a) Contours based on $\vec{n} \cdot \vec{e}$



(b) Constant-width contours

Figure 9: Controlling the width of the contours

ure 9(a)). Kindlmann et al. [13] observed a similar problem when drawing contours using transfer functions in direct volume rendering. They proposed using a 2D transfer function dependent on not only the $\vec{n} \cdot \vec{e}$ value, but also on the local surface curvature. Their method requires the surface curvature in the viewing direction, which they derive from the geometry tensor matrix for each point whenever the viewpoint changes.

We can avoid additional expensive computations by re-using our earlier approximation of the contour. The distance of a particle to the contour can be derived from the distance of this line to the origin in the $(\vec{k}_1, \vec{k}_2)$-plane,

$$\tau = \frac{\vec{n} \cdot \vec{e}}{\sqrt{\left(\kappa_1 \left(\vec{k}_1 \cdot \vec{e}\right)\right)^2 + \left(\kappa_2 \left(\vec{k}_2 \cdot \vec{e}\right)\right)^2}}. \quad (4)$$

Assuming orthogonal projection, the distance of the particle to the contour in the image plane is $T = \tau(\vec{n} \cdot \vec{e})$. We can therefore place a threshold on the value of T in order to obtain contours of approximately constant width (see figure 9(b)). The only disadvantage is that principal curvature information has to be computed for all particles. However, as this information is independent of the viewpoint this can be performed in a pre-processing step.

7 Results

The algorithms described in this paper have been implemented in C++ using the OpenGL 3D graphics API. The resulting system allows a user to easily configure a number of sets of particles within a volume dataset, each of which can have its own visualization technique and parameters.

Different techniques can be combined to visualize multiple features in a volume dataset. Figure 1 shows a CT dataset of a hand visualized using our framework. The focus of the image, the bones of the hand, are visualized using an iso-surface. This is combined with principal-curvature directed hatching and contours in order to better illustrate the shape of this surface and emphasize it in the image. Context is provided by a visualization of the skin (using stippling and contours) and of the arteries (using accurate contours based on curvature).

In comparison, figure 10 shows a photorealistic visualization (Direct Volume Rendering) of a similar dataset. The DVR techniques allow for visualization of ranges of values in the data rather than only selected iso-surfaces, for example, the muscle tissue and tendons (shown in red). However, the result often looks fuzzy and may clutter the image, especially when one is mainly interested in the bones. In these cases, the sparseness of our NPR techniques may offer a better alternative. Because only simple geometric primitives were used to render our results (points and lines), these two types of visualizations may also be easily combined, for instance in a focus / context type of visualization.

Our framework is flexible enough to produce visualizations similar to the work by Nagy et al. [4] and Dong et al. [5]. The current implementation of the system is limited to visualizing iso-surfaces, therefore we can not create volumetric visualizations as used by Lu et al. [3]. However, the methods presented can be extended to other types of surfaces if the required derivative properties (surface normal and in some cases curvature) are available.

The system runs at interactive speeds (with the exception of the pre-processing steps), allowing a user to immediately observe the result of changing most of the visualization parameters. Table 1 gives an overview of the performance of the (unoptimized) prototype implementation, for some of the images presented in this paper. All datasets used were in the range of 256^3 voxels. Stages 1 and 2 are pre-processing steps (executed once per dataset) while stages 3 and 4 are executed during interactive rendering when certain parameters are changed.

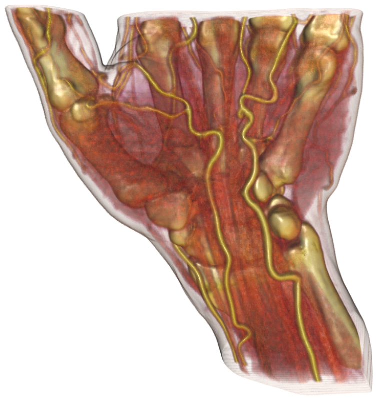


Figure 10: DVR visualization of a CT hand dataset, created using VolumeShop by Bruckner et al. [1]

Image	particles	stage 1&2	stage 3&4
Figure 1	449529	120 s	2 fps
Figure 6(a)	34162	20 s	30 fps
Figure 6(b)	218432	70 s	6 fps
Figure 7(a)	58367	35 s	19 fps
Figure 7(b)	58367	60 s	17 fps
Figure 8(a)	27070	15 s	19 fps
Figure 9(a)	29481	20 s	4 fps
Figure 9(b)	29481	20 s	2 fps

Table 1: Performance of the framework on a modern computer (Athlon 2 GHz)

8 Conclusions

We have discussed a framework for particle-based non-photorealistic volume visualization. Our main contributions are:

- **The VolumeFlies framework,** a framework for non- photorealistic rendering of volume data, based on particle systems and operating directly on the voxel data. The framework is flexible, scalable and largely independent of data resolution. It supports various styles in a unified way, including stippling, hatching and contouring.
- **A new hidden surface detection algorithm** for surfaces rendered using particles. This method does not require the construction of an explicit geometric representation of the surface and can be applied at interactive speeds.
- **New techniques for NPR and visualization**, including a simple density-based shading algorithm, contour approximation methods and a direction field smoothing algorithm for hatching based on principal curvature directions, which uses the shape index to indicate the local suitability for hatching.

Non-photorealistic methods for data visualization, such as the ones presented in this paper, may be used to simplify visualizations in cases where "realistic" methods would clutter the image. They are by no means a replacement for photorealistic methods but rather a useful addition. We expect that non-photorealistic methods are particularly suitable to provide context for more realistic visualizations, as their inherent simplicity will serve not to distract the viewer from the focus of such an image.

We also conclude that the use of particles presents a useful alternative to traditional surface-based methods. The flexibility presented by particles combined with the advantages of working directly on the data may present new possibilities for data visualization. While such techniques are currently non-accelerated, recent developments in graphics hardware may well be used in order to improve performance.

References

[1] S. Bruckner and E. Gröller. Volumeshop: An interactive system for direct volume illustration. In *Proc. IEEE VIS*, pages 671–678, 2005.

[2] V. Interrante, H. Fuchs, and S. Pizer. Conveying the 3d shape of smoothly curving transparent surfaces via texture. In *IEEE Visualization and Computer Graphics*, pages 98–117, 1997.

[3] A. Lu, D.S. Ebert, C. Hansen, M. Hartner, C.J. Morris, P. Rheingans, and J. Taylor. Illustrative interactive stipple rendering. *IEEE Visualization and Computer Graphics*, 9(2):127–138, 2003.

[4] Z. Nagy, J. Schneider, and R. Westermann. Interactive volume illustration. In *Proc. Vision, Modeling and Visualization Workshop*, 2002.

[5] F. Dong, G.J. Clapworthy, H. Lin, and M.A. Krokos. Nonphotorealistic rendering of medical volume data. *IEEE Computer Graphics and Applications*, 23(4):44–52, 2003.

[6] D. Cornish, A. Rowan, and D. Luebke. View-dependent particles for interactive non-photorealistic rendering. In *Proc. Graphics Interface*, pages 151–158, 2001.

[7] A. Pang and K. Smith. Spray rendering: Visualization using smart particles. In *Proc. IEEE VIS*, pages 283–290, 1993.

[8] A.P. Witkin and P.S. Heckbert. Using particles to sample and control implicit surfaces. *Computer graphics and interactive techniques*, 28:269–277, 1994.

[9] M.D. Meyer, P. Georgel, and R.T. Whitaker. Robust particle systems for curvature dependent sampling of implicit surfaces. In *Shape Modeling and Applications*, pages 124–133, 2005.

[10] O.M. Pastor and T. Strotthote. Graph-based point relaxation for 3d stippling. In *Proc. ENC Computer Science*, pages 141–150, 2004.

[11] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH*, pages 343–352, 2000.

[12] S. Busking. Volumeflies - a smart-particle-inspired framework for illustrative volume rendering. Master's thesis, Technische Universiteit Eindhoven, July 2006.

[13] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Moller. Curvature-based transfer functions for direct volume-rendering: methods and applications. In *IEEE VIS*, pages 513–520, 2003.

[14] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proc. Computer graphics and interactive techniques*, pages 517–526, 2000.

[15] J.J. Koenderink and A.J. van Doorn. Surface shape and curvature scales. *Image and Vision Computing*, 10(8):557–565, 1992.

[16] D.J. Struik. *Lectures on Classical Differential Geometry*. Courier Dover Publications, 1988.