Interactive Glyph Placement for Tensor Fields

Mario Hlawitschka[1] and Gerik Scheuermann[1] and Bernd Hamann[2]

[1]Image and Signal Processing Group, University of Leipzig, Germany
[2]Institute for Data Analysis and Visualization (IDAV) and
Department of Computer Science, University of California, Davis, USA

Abstract. Visualization of glyphs has a long history in medical imaging but gains much more power when the glyphs are properly placed to fill the screen. Glyph packing is often performed via an iterative approach to improve the location of glyphs. We present an alternative implementation of glyph packing based on a Delaunay triangulation to speed up the clustering process and reduce costs for neighborhood searches. Our approach does not require a re–computation of acceleration structures when a plane is moved through a volume, which can be done interactively. We provide two methods for initial placement of glyphs to improve the convergence of our algorithm for glyphs larger and glyphs smaller than the data set's voxel size. The main contribution of this paper is a novel approach to glyph packing that supports simpler parameterization and can be used easily for highly efficient interactive data exploration, in contrast to previous methods.

1 Introduction

A variety of methods exists for the visualization of tensor fields. In medical imaging applications, for example, tensor data is most commonly visualized via pseudo–colored slices. In the context of diffusion tensor (DT) imaging, the color of a pixel typically depends on the directional information and fractional anisotropy (FA) of a tensor. Due to the human perception of the color space, this method does not provide full coverage of all tensor information. Tensor glyphs are another simple method, widely used especially in engineering that is capable of displaying all intrinsic information of the tensor. However, this visualization is only informative when it is not obscured, i.e., all glyphs can be seen properly. Other methods use the continuous properties of interpolated fields to display structures, such as texture–based methods like line integral convolution (LIC) [1]. While LIC itself can only provide a single directional information without scaling attributes, enhancements of LIC to tensors such as HyperLIC [2] and metric interpretations of tensor fields [3] overcome this problem. Hyperstreamlines [4] show line features in the data and can be combined with additional attributes, but when applied to 2D slices of 3D tensor data, they provide incorrect information when integration is restricted to a plane. Nevertheless, integration–based methods provide a continuous view of the data.

In contrast to vector field visualization, where glyph placement tries to minimize the number of glyphs that describe the vector field [5], for medical tensor

fields, a full coverage of the field is important as features, such as tumor, may be small. This is partly due to boundaries playing an important role in medical imaging data sets and behavior along these boundaries is of interest, and partly because noise changes the data locally. Because of the noise, clustering the data into areas of similar behavior does not work here. Kindlmann and Westin [6] combined this continuous nature of the field as done by LIC or hyperstreamlines with the discrete sampling of glyphs by densely packing the glyphs to strengthen the continuity of the field. The major improvement of this method compared to discrete sampling of the glyphs is that glyphs no longer overlap and visual artifacts induced by the regular sampling of glyphs are reduced. In contrast to glyph–based vector field representations where a reduction of glyphs is the most important method to reduce visual clutter, a dense packing is enforced in measured tensor data to not overlook small features.

We use Kindlmann and Westin's "glyph packing" as a basis for our algorithm but have enhanced his method. Our main contributions are

- the use of a different, parameter–less acceleration structure that does not require good selection of bin sizes like the structure proposed before,
- an improved initial placement of glyphs to reduce overall computation time,
- a re–usable acceleration structure and placement for interactive manipulation of the planes the glyphs are drawn in,
- the exchange of parameters by self–explanatory ones that directly influence the visualization in an intuitive way, and
- an implementation where the glyphs follow a pointer such that a plane in the data set or a region of interest can be chosen interactively by the user while the algorithm provides feedback at interactive frame rates.

2 Glyph Packing

Kindlmann and Westin introduced glyph packing as an iterative approach to optimize the placement of tensor glyphs. Their method uses a particle model where each particle induces a potential field and therefore, particles attract or repel each other, depending on their distance measured in a metric space distorted by the tensor shape. The movement of particles is described by the differential equation

$$\sum_{a \neq b} f_{ab} - C_{drag} \frac{d\mathbf{p}}{dt} + C_{ext} f_{ext} = \frac{d^2\mathbf{p}}{dt^2}. \tag{1}$$

C_{drag} and C_{ext} are scaling parameters, f_{ab} are the inter–particle forces, and $\mathbf{p}$ is the location of the particle a. These forces are illustrated in Fig. 1. The drag force prevents the field from oscillating while the external force f_{ext} forces some additional layout parameters. Kindlmann and Westin use, e.g., $f_{ext} = \nabla M(\mathbf{p})$ where M is a scalar mask on the data to prevent the glyphs from leaving the domain of the data set.

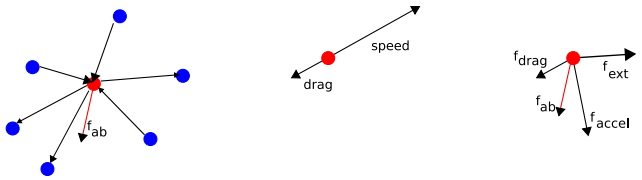

Fig. 1. Illustration of the particle interactions resulting in the force f_{ab} (left) the drag force f_{drag} (middle) and the resulting acceleration f_{accel} (right) from Eq. 1. The forces are computed for a glyph located at the red point. The blue points are positions of glyphs that influence the current point, resulting in an interparticle force. The drag force (middle) always antagonizes the current speed causing fast particles to slow down which prevents oscillation. Note that the particle's speed is the first derivative and the drag force influences the second derivative. Finally, all forces are summed up resulting in an acceleration on the particle (right).

2.1 Computing the Initial Distribution

When analyzing the force function of Kindlmann and Westin, there is a distance of strong repulsion, followed by a strong attraction that decays with increasing distance. Therefore, glyphs that are farther away have less influence than nearby glyphs. In addition, glyphs that have almost the optimal distance to each other have strong forces for keeping that distance fixed. If there is a hole in the data where no glyph is located while the glyphs outside are aligned at an almost optimal distance, there are only weak forces that try to close that hole. This example shows how crucial good initial seeding is for the algorithm. An example of the described effect is shown in Fig. 2. The initial sampling is a uniformly distributed random placement but obviously the same effect can occur when Neumann's rejection method is applied as done by Kindlmann and Westin. To prevent these cases, we compute an improved initial distribution compared to the one proposed by Kindlmann and Westin by using one of the two variants of the following approach, depending on the size of glyphs relative to the cell size of the original data set. The choice of the size of glyphs partly depends on

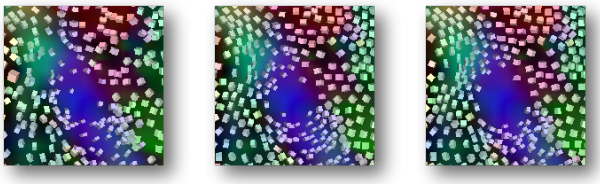

Fig. 2. A worst–case scenario for the initial seeding: The randomly placed initial glyphs (left) do not fill the area completely, but leave a hole at one point. Because of the distance across the hole, there are only week forces trying to close that hole, while there are strong forces to stabilize the layout around that hole. The holes stay stable throughout the optimization process (middle and right image).

the available screen resolution. To avoid visual clutter, glyphs have to be large enough to distinguish them and identify their properties, e.g., different scaling along the eigen–directions. On the other hand, they have to be small enough to keep a certain amount of glyphs on the screen to provide the interpolation like effect of dense packing and to provide information on the behavior around the glyph. Therefore, when zooming into one area, there have to be more glyphs per cell than when looking at the data from far away. Considering most commonly used medical data sets, cell size is fixed by the scanner's resolution where every slice of the data set provides a uniform, rectilinear grid containing 128×128 or 256×256 tensor values. When inspecting the data set in a full–screen mode, we found that 1000 glyphs for areas of interest and up to 65000 glyphs for whole data sets are enough for providing all features of dense packing and still avoiding visual clutter. Therefore, less than one glyph per cell needs to be drawn. When inspecting certain areas more closely – especially in printing media, we end up with areas of about 10×10 cells, but still want to draw the same amount of glyphs. In this case more than ten glyphs on the average gather in one cell. This is taken into account in our second sampling approach, where stratified sampling is used to increase the quality of the sampling for more than one glyph per cell. Let v be a cell and let A_v be the (average) area covered by a glyph in the cell. The ratio between the cell size A_v and the estimated glyph area $p_v = \frac{A_v}{a_v}$ provides an estimate of the number of glyphs in the voxel. Obviously, we overestimate the number of glyphs, but this is not a problem. By summing up p_v for all cells, i.e., $P = \sum_v p_v$, we get an estimate P of the number of glyphs we need to cover the whole data set. Furthermore, the probability for a new glyph to be added in cell v will be p_v/P. We then can use the inversion method [7] to get a mapping from a uniform distribution to the distribution of glyphs and estimate the initial placement of glyphs using the following algorithm:

```
Algorithm: Initial Glyph Placement

A[] = cellSizes();
a[v] = estimated size of glyph in cell v
p[v] = A[v]/a[v]
// integrate density
for (v = 1; v < #cells; ++v)
    p[v] = p[v-1]+p[v]
P = p[#cells-1]
for (v = 1; v < #cells; ++v)
    p[v] = p[v]/P
for (i=0; i < P; ++i)    // place P random glyphs
    d = random()         // in [0..1)
    // binary search for cell with p[v-1] <= d < p[v] where p[0]=0
    i = findCellFor(d)
    place glyph randomly in cell i
```

For cells outside the data set or in areas where no glyphs should be drawn, p_v can be set to zero to avoid initial seeding of glyphs. The more glyphs there have to be seeded in a single voxel, the more important becomes a proper placement within the cell. As described before, high–quality close–up views of the data set require many glyphs seeded in a single cell. We address this by placing glyphs

using stratified sampling [8], i.e., we sample the cells independently to guarantee a better distribution within the cells: When there are voxels where it is likely that no glyph is placed, the shown algorithm is used to estimate the number of glyphs placed in every voxel. When we already know, by checking, e.g., the values of p_v for larger than one everywhere, that more than one glyphs will likely be placed in every voxel, we use a rounded value of p_v as the number of glyphs. Now every voxel is filled with a uniform distribution of this amount of glyphs where the location of the glyphs are obtained from a list of strata.These strata can be pre–computed sets of points, e.g., following a Poisson–disc distribution [8] or other randomly created samples of points that have been checked to have low energy between neighboring points for the isotropic case. To avoid regular patterns during the first steps of the optimization, we implemented interleaved sampling [9] using a set of strata and access different sets of points through a hash map using the cell index as key. Instead of using pre-computed strata, pseudo–random sequences can be used and applied with different seeds. Good pseudo–random sequences automatically fill the holes in–between points and therefore can be used incrementally, i.e., if the resolution changes, more points can be added using the same sequence.

2.2 Modified Force Function

While Kindlmann and Westin use a force function based on the ellipsoidal tensor model, we propose the use of an alternative function to improve the packing for different types of glyph representations. We describe the force on a particle directly from its scaled distance to its neighboring glyphs, i.e.:

$$y_{ab} = p_a - p_b \tag{2}$$

$$d_{opt}(a,b) = g_a(-y) + g_b(y) + c_{offset} \tag{3}$$

$$f_{ab} = \phi' \left(\frac{y_{ab}}{d_{opt}(a,b)} \right). \tag{4}$$

Here, c_{offset} is a parameter describing an absolute offset between the glyphs. In most cases $c_{offset} = 0$ is a good choice, but to reduce the number of glyphs while keeping their size constant, an absolute offset may be given here. $g_a(y)$ denotes the surface function of glyph a evaluated at direction y. The sign in the argument is only important if the glyphs would be asymmetric which is not the case in medical data. A simple example of g for ellipsoidal glyphs, i.e., the surface described by the sphere S deformed by the tensor D by DS would be $g(y) = \frac{\alpha\|y\|}{\|D^{-1}y\|}$ which is the distance from the glyph's center to its surface measured along direction y. In contrast to the previous paper [6], in our version the function g can be chosen arbitrarily, therefore glyphs can be painted as boxes, superquadrics or ellipsoids and the force function will always try to avoid glyphs overlapping which is not the case in the previous approach. If g is chosen to be the projection of the glyph on the plane we can overcome the problem of overlapping glyphs in 2D slices that is mentioned in Kindlmann and Westin's

results section. ϕ denotes the force function where ϕ', i.e., its derivative, is chosen to be

$$\phi'(r) = \begin{cases} r-1 & 0 < r < 1 \\ (r-1)(1+\gamma-r)^2/\gamma^2 & 1 \leq r \leq 1+\gamma \\ 0 & r > 1+\gamma \end{cases}, \tag{5}$$

where the optimum would be the zero–crossing at $r = 1$. All glyphs farther away than $r = 1 + \gamma$ do not influence this glyph.

2.3 Acceleration Using a Delaunay Triangulation

Because glyphs that are farther away in the metric distance have no influence on each other, computation of distances between them should be avoided. We do this by introducing a Delaunay triangulation of our sample points that is updated after moving the points. Building such a triangulation for the relevant amount of glyphs is fast. Timing can be found in Table 1. In general, between two steps, the grid does not change much. Therefore, the computational cost can be reduced by reusing the existing data structure which is only useful when a lot of glyphs are drawn and the computational complexity of the Delaunay triangulation surpasses the time of the force computation. The advantange of this data structure is that searches for neighboring points can be performed easily. A breadth–first–search can be performed to get the closest points in terms of edge distance. Usually, these points are the ones with the largest influence on the vertex. While using neighbors with a two edge distance seems to have slightly better results, direct edge neighbors are enough to make our algorithm converge.

2.4 User Parameters

Our implementation has two modes: First, it is possible to distribute a certain amount of glyphs in the data set and let the algorithm spread them in a large enough area, and second, one can select an area, where glyphs should be placed and the algorithm selects the number of glyphs needed by itself. This is an advantage to the approach previously presented, as in both cases, the user only has to select visual properties while the previous approach only lets you select the number of glyphs and the parameters for the forces, but does not provide a reasonable mean to compute one parameter from the others.

2.5 Addition and Removal of Glyphs

If a certain area of the data set needs to be filled with glyphs, and the pre–estimated amount of glyphs is not enough to fill this area, we need to add glyphs. A glyph is added when there is an area, where the distance between neighboring glyphs is large. These areas can be found by looking for triangles that have at least two long edges. In contrast, in areas where too many glyphs are present, glyphs need to be removed. We mark glyphs for removal that only have high repelling forces on their neighbors. As in dense areas, we do not want to remove

all glyphs at once, we remove a certain set of glyphs randomly. Re-evaluation of the edge forces in this step provides a hint of the quality after removing the glyphs which may be used to improve the selection.

2.6 Adding Group Movement for Interactive Glyph Clouds

In addition to the mode presented in 2.4, an interactive mode is supported in our implementation, where a set of glyphs is placed around a pointing device and will follow the pointer loosely. Instead of having fixed positions related to the pointing device, the glyphs will be in motion to reorganize for the best visualization. Here, the user selects the number of glyphs to have displayed. We initially place the glyphs randomly around the cursor as described before. Then the grid is optimized while the glyphs are already displayed, which gives the user interactive feedback. When the cursor is moved, all points are moved the same way as the cursor and the optimization process keeps on moving them relatively to each other to avoid overlapping. By fixing one glyph to the cursor, we prevent the cloud to diffuse randomly into the data set, but the cloud may become asymmetric. To prevent this, additional small forces can be applied pointing to the cursor.

2.7 The Integration Step

The actual optimization is done in an Euler iteration [10] where a stepsize of 0.2 has empirically proven to be appropriate for all data sets for which we have tested our method.

```
Algorithm: Iteration

while not converged or maximal stepsize reached
    sample tensor values at particle positions
    compute all particle forces
    update particle speed
    move particles
    recompute Delaunay triangulation
    if( change in number of glyphs is allowed by user )
        check grid for triangles with large distances and
            attracting forces and place glyphs in those cells
        check grid for vertices with large repelling forces on all
           edges and mark these vertices for removal
        remove random subset of these vertices
```

3 Evaluation and Results

We implemented the proposed algorithms in our visualization system and used CGAL [11] for computing the Delaunay triangulation. We applied our algorithm to data acquired with a three–Tesla Siemens scanner on a healthy volunteer. 60 diffusion weighted images were acquired using three times averaging and 21

baseline images. The data has been converted to second order tensors using linear least squares fit [12]. Data acquisition takes about 20 minutes with an in–slice resolution of 256×256 voxel and 40 slices on a $1.7 \times 1.7 \times 3.0mm^3$ grid.

Fig 3 shows the Delaunay triangulation used for neighborhood calculation, the resulting packing of glyphs and an overlay on a pseudo–colored slice. While the grid only shows direct neighbors, all neighbors having an edge distance of two are taken into account in the calculation to improve stability when edges of the triangulation flip. The evolution from the initial seeding throughout the first 30 steps is shown in Fig. 4. The energy function between glyphs decreases tremendously during the first steps which results in an almost stable packing after about 15 steps. Fig. 5 shows a sequence of images out of an interactive sweep from the singuli to the corpus callosum.

We performed timing experiments on different scenarios in a single threaded environment. It turns out that up to 8000 glyphs can be optimized using the interactive approach on a desktop PC. For complete slices, a calculation time of about one minute is needed to get first results but up to four minutes are required until they become stable. A comparison of times can be found in Table 1. We used 10000 iterations for small amounts of glyphs to get comparable times, but far less than 100 iterations are required to obtain stable results there.

4 Conclusions

The main contribution of this paper is the development of a modified glyph packing algorithm that overcomes several drawbacks we found in the algorithm presented by Kindlmann and Westin [6]. First, we introduced a parameter–less acceleration structure that has small computational complexity and allows interactive tensor visualization using glyph placement. Second, we implemented an improved initial seeding that reduces the number of steps required by the iterative optimization and makes the algorithm more likely to converge to visually good results.To provide the user, i.e., the radiologist or surgeon, with an easy to use user interface usable in surgery, we reduced the number of parameters to two, namely the scaling of the glyphs and the distance between neighboring glyphs. Furthermore, we provided an interactive user interface where areas of interest can be selected interactively using a pointer, e.g., the mouse or a tracked 3D

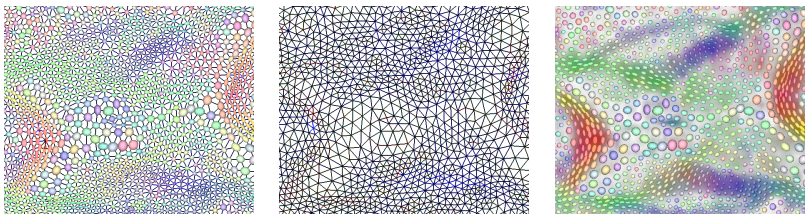

Fig. 3. Result of glyph packing of 2000 glyphs with grid (left), Delaunay triangulation only (middle) and glyphs shown on a color–mapped slice (right).

Fig. 4. Different steps of the iterative algorithm using 2000 glyphs. An initial random placements of seeds and steps 20 and 30 are presented from top left to bottom right.

Glyphs	Iterations	Depth	Time[s]	Delaunay[s/step]	Force[s/step]	Interpol.[s/step]
100	10000	2	15	-	-	-
200	10000	2	35	-	-	-
400	10000	2	80	-	-	-
1000	100	2	2	-	0.03	-
2000	100	2	5	-	0.06	-
4000	100	2	15	-	0.14	-
10000	100	2	55	0.08	0.46	0.03
10000	10	2	5	0.02	0.21	0.02
20000	10	2	17	0.34	1.44	0.06
40000	10	2	61	1.36	4.4	0.1

Table 1. Timings of different configurations as absolute time, time for constructing the Delaunay triangulation in every step, time for computing the forces in every step and time for interpolation of the tensor values in every step. Cells marked with a dash are below the threshold of 0.01 seconds. The glyph packing usually becomes stable after several hundred steps, more than one thousand steps have never been required in our experiments. The detailed timings show that most of the time is spend in computing the inter-particle forces.

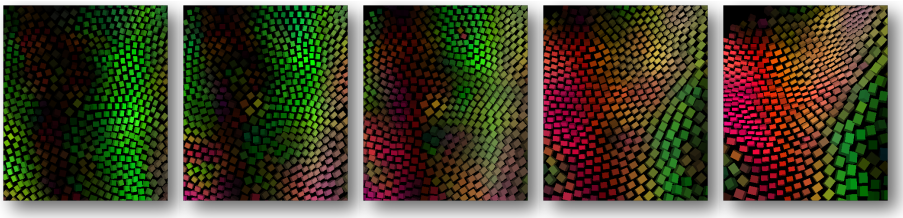

Fig. 5. An interactive sweep through a data set. The glyphs realign at interactive frame rates to prevent overlapping.

pointing device as used in surgery, and the glyphs automatically align around this pointer in a way that reduces visual clutter. Because our approach does not require any pre–computation or pre–processing, it can be used as an interactive probe to inspect areas of interest and can be used supplementary to existing visualization techniques.

5 Acknowledgments

We thank our cooperation partners from the Max Plack Institute for Human Cognitive and Brain Science in Leipzig, especially Harald E. Möller, Marc Tittgemeyer, Alfred Anwander and Thomas Knösche for discussion, evaluation of our results and for providing the data sets. We thank Christian Heine for his ideas for optimizing the algorithm.

References

1. Cabral, B., Leedom, L.C.: Imaging Vector Fields Using Line Integral Convolution. In: SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, ACM Press (1993) 263–270
2. Zheng, X., Pang, A.: HyperLIC. In: Proceedings of IEEE Visualization '03, Los Alamitos, CA, IEEE Computer Society (2003) 249–256
3. Hotz, I., Feng, L., Hagen, H., Hamann, B., Joy, K.I.: Tensor field visualization using a metric interpretation. In Weickert, J., Hagen, H., eds.: Visualization and Processing of Tensor Fields, Springer–Verlag Berlin Heidelberg (2006) 269–281
4. Delmarcelle, T., Hesselink, L.: Visualizing second-order tensor fields with hyperstreamlines. IEEE Computer Graphics and Application **13 (4)** (1993) 25–33
5. Griebel, M., Preusser, T., Rumpf, M., Schweitzer, M.A., Telea, A.: Flow field clustering via algebraic multigrid. In Rushmeier, H., Turk, G., van Wijk, J.J., eds.: Proceedings of IEEE Visualization '04, Los Alamitos, CA, IEEE Computer Society (2004) 35–42
6. Kindlmann, G., Westin, C.F.: Diffusion tensor visualization with glyph packing. In Gröller, E., Pang, A., Silva, C.T., Stasko, J., van Wijk, J., eds.: Proceedings of IEEE Visualization'06, Los Alamitos, CA, USA, IEEE Computer Society Press (2006) 1329–1335
7. Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. ninth Dover printing, tenth GPO printing edn. Dover, New York (1964)
8. Sobol, I.M., Messer, R.: Monte Carlo Method (Popular Lectures in Mathematics). University of Chicago Press (1975)
9. Keller, A., Heidrich, W.: Interleaved sampling. Rendering Techniques (2001) 269–276
10. Bronstein, I., Semendjajew, K., Musiol, G., Mühlig, H.: Taschenbuch der Mathematik — 5., überarbeitete und erweiterte Auflage. Verlag Harri Deutsch, Thun und Frankfurt am Main (2001)
11. The CGAL Consortium: CGAL, Computational Geometry Algorithms Library (2007) http://www.cgal.org.
12. Basser, P., Mattiello, J., LeBihan, D.: Estimation of the effective self–diffusion tensor from the NMR spin echo. Journal of Magnetic Resonance **3** (1994) 247–254