Part III

Data-Domain Visualization

6 Continuous Scatterplots

Feature-based vector field visualization techniques described in the previous part of this thesis can be a very effective means to analyze two- or three-dimensional fields. However, today's simulations not only produce large data sets, but also complex ones—usually including several data dimensions. In such cases, the visualization methods presented in previous chapters might not be sufficient to effectively analyze those data sets. Having only those methods at hand, correlations between data dimensions may remain hidden, since only a few dimensions can be evaluated at the same time.

In this chapter, the undisputed visualization power of scatterplots is utilized and extended to display the type of data typically generated in computational sciences. This method fits in the recent trend of combining statistical visualization methods like scatterplots with scientific visualization methods like volume or flow visualization.

The term *continuous scatterplot* was defined in our publication [BW08b] as follows. First, input data is no longer a collection of discrete data points but a field of data values defined on a continuous domain. Typically, the domain is 2D or 3D and has intrinsic spatial embedding; the dimensionality of the domain increases by one if time dependency is included. Data representation often relies on a grid with respective interpolation or approximation schemes. Please note that the data field defined on the continuous domain is not necessarily continuous, i.e., even non-continuous functions can be visualized. The second aspect of a continuous scatterplot is that the output is continuous: instead of a collection of discrete points, a continuous density is drawn on the diagram, providing a continuous frequency description of two data dimensions.

This model of continuous scatterplots and respective computations is generic in the sense that both the spatial domain of the input field and the dimension of the scatterplot can be chosen freely. In practice, the dimensionality of the input-field domain is determined by the data input, which mostly is restricted to dimensions 2–4. From a perceptual point of view, useful output dimensionalities are 1 (i.e., continuous histogram) and 2 (i.e., scatterplot in its original sense). Although there is previous work on 3D discrete scatterplots, their perceptual effectiveness is unclear. To visualize higher-dimensional data,

2D continuous scatterplots can be combined to respective scatterplot matrices. Alternatively, higher-dimensional data can be visualized with parallel coordinates [ID90]. For the case of scientific visualization, parallel coordinates can be computed in a continuous way as well—based on the ideas presented in this section. These continuous parallel coordinates were introduced by Heinrich and Weiskopf [HW09].

6.1 Mathematical Model

This section presents the mathematical model of generic continuous scatterplots. First, required terminology is defined before the generic model is described. Later parts of this section derive specific results for special cases like 2D scatterplots or 1D histograms. In this context, these continuous versions are compared with their well-known discrete counterparts.

6.1.1 Generic Model

The computation of continuous scatterplots needs two different domains: the n-dimensional domain on which the input field is defined, and the m-dimensional domain of the scatterplot onto which the input field is mapped. The first one is denoted as *spatial domain* because it typically describes 2D or 3D spatial positions. Despite this terminology, the spatial domain may also include a time dimension; in fact, any continuous field domain is supported. The second kind of domain is denoted as *data domain* because it represents the multi-attribute data values of the input field. Typical examples are $m = 1$, which corresponds to a histogram for one data component, or $m = 2$, which corresponds to a 2D scatterplot. The input field to be visualized can be represented mathematically by a map from the spatial domain to the data domain: $\tau : \mathbb{R}^n \to \mathbb{R}^m$ The map τ can represent all typical scientific data, including 3D scalar fields, 3D vector fields, or multi-attribute fields. The problem of constructing a continuous scatterplot can then be formulated as finding a density function σ defined on the data domain:

$$\sigma : \mathbb{R}^m \longrightarrow \mathbb{R}, \quad \xi \longmapsto \sigma(\xi) \,, \tag{6.1}$$

which represents continuous frequency and depends on τ. In fact, the mathematical basis of the continuous scatterplot is an operator that maps the function τ to the function σ.

To construct this operator, a continuous description is derived by starting from well-known discrete scatterplots and considering the limit process for infinitely dense data points. This approach is similar to deriving continuum mechanics from systems of discrete mass points (see, e.g., in Goldstein [Gol80, Ch. 12]). Figure 6.1 illustrates the discrete particle model for the example

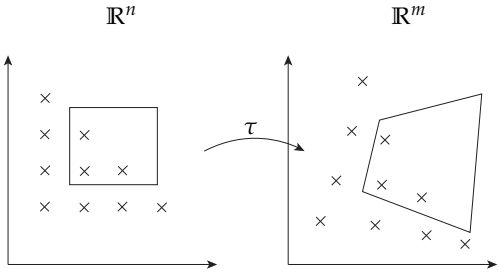


Figure 6.1: Change of point density, by applying the mapping τ from the spatial domain of the input data (left) to the data domain of the scatterplot (right). Sampling density is indicated by dots; the two quadrilaterals visualize the mapping of an extended area. Here, both spatial and data domains have dimension 2.

$n = m = 2$. The derivation is based on two assumptions: First, points in the spatial domain are given according to some kind of density description (typically, uniform density), and second, the mapping τ does not change the number of points. The second assumption is identical to mass conservation if mass points are considered. The limit process for infinitely dense particles leads to a replacement of particle mass by mass density.

According to the first assumption, the mass (i.e., sampling) density s is known in the spatial domain, with $s : \mathbb{R}^n \to \mathbb{R}, x \mapsto s(x)$. The mass M of a covered volume $V \subset \mathbb{R}^n$ is $M = \int_V s(x)\mathrm{d}^n x$. Similarly, the density σ in the data domain can be integrated to compute the mass of the covered volume $\Phi \subset \mathbb{R}^m$. Please note that this notation uses Latin characters for quantities related to the spatial domain and Greek characters for quantities related to the data domain; lower case letters denote scalar or vector values, uppercase letters denote volumes.

If V and Φ are related by $\tau(V) = \Phi$, mass conservation under the transformation τ implies

$$M = \int_V s(x)\,\mathrm{d}^n x = \int_{\Phi=\tau(V)} \sigma(\xi)\,\mathrm{d}^m \xi\,, \tag{6.2}$$

which determines the unknown density function σ for a given input density s because Equation 6.2 has to hold for any volume V in the spatial domain. Rewriting Equation 6.2 leads to the alternative formulation

$$\int_\Phi \sigma(\xi)\,\mathrm{d}^m \xi = \int_{\tau^{-1}(\Phi)} s(x)\,\mathrm{d}^n x\,, \tag{6.3}$$

which equally determines σ because Equation 6.3 has to hold for any volume Φ in the data domain. The inverse map $\tau^{-1}(\Phi)$ is well defined even if τ is

not invertible because, here, this is done on maps of sets, not of single function values.

Please note that σ is only indirectly defined in Equations 6.2 or 6.3 via the effect of integration. For the generic case of scatterplot computation, this indirect definition of σ is required so that not only regular functions are supported but also distributions (generalized functions) like Dirac δ distributions. An introduction to distributions and functional analysis is given in the textbook by Griffel [Gri02]. In fact, the earlier definition of σ as a regular function in Equation 6.1 is extended to allow for distributions as well. Later, it will be shown that δ distributions are useful to build a relationship between continuous and discrete scatterplots, and to allow for scatterplots and histograms of (partly) constant functions. The indirect formulation of Equations 6.2 and 6.3 can be rewritten to directly compute σ for some special, yet important cases.

6.1.2 Case m = n

To begin, the special case of equal dimension $m = n$ is considered in order to compute σ. Since σ is directly based on τ, this discussion is split into several parts that cover different possible subcases depending on the properties of τ. First, it is assumed that τ is differentiable. Here, both subcases are considered, where τ is a diffeomorphism or no diffeomorphism. Later, this discussion is extended to non-differentiable τ.

Assuming that τ is a diffeomorphism from $\mathbb{R}^n \to \mathbb{R}^{m=n}$, the integration variable ξ can be substituted by x in Equation 6.2 according to the transformation theorem for integrals:

$$\int_{\tau(V)} \sigma(\xi)\,\mathrm{d}^{m=n}\xi = \int_V \sigma(\tau(x))|\det(D\tau)(x)|\,\mathrm{d}^n x \stackrel{!}{=} \int_V s(x)\,\mathrm{d}^n x\,, \tag{6.4}$$

where $D\tau$ denotes the derivative of the map τ, i.e., the $n \times n$ Jacobi matrix. Note that the determinant is a volume measure, representing the volume spanned by the partial derivatives of τ. Since the second equality of Equation 6.4 has to hold for any $V \subset \mathbb{R}^n$, the integrands need to be equal, which leads to

$$\sigma(\xi) = \frac{s(\tau^{-1}(\xi))}{|\det(D\tau)(\tau^{-1}(\xi))|}\,. \tag{6.5}$$

Figure 6.2 illustrates the density mapping according to Equation 6.5 for a 1D example $m = n = 1$. Geometrically, the density ratio σ/s in a small neighborhood is given by the ratio of the covered lengths, $V/\tau(V)$, which in turn is computed by the reciprocal of the slope of τ. Please note that a similar approach is followed for the inversion of cumulative distribution functions in order to map probability distributions or derive histogram equalization.

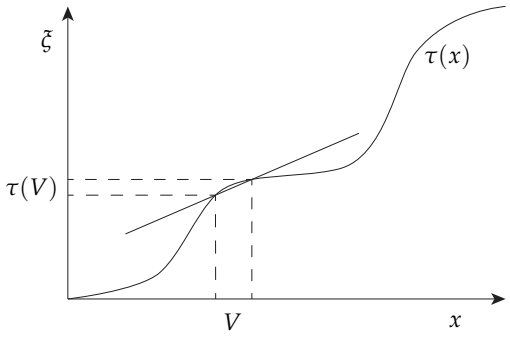


Figure 6.2: Mapping of intervals from the spatial domain x to the data domain ξ. The ratio of the original and mapped intervals, which is given by the reciprocal of the slope of τ (slope is indicated by the straight line), is a measure for the change of density from the spatial to the data domain.

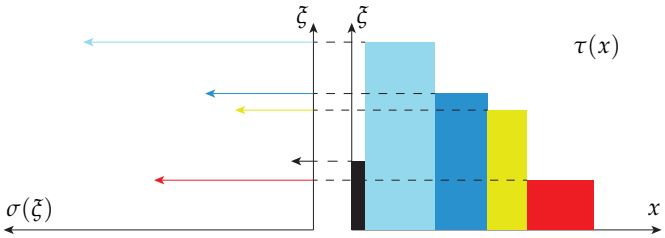


Figure 6.3: Continuous histogram (left) of a piecewise constant function (right). The histogram is rotated by 90 degrees to visualize data correspondence between the function plot and the histogram. δ peaks are indicated by arrows; the length of a δ peak corresponds to the width of the respective interval in the function plot.

When $\det(D\tau) = 0$, then τ is not a diffeomorphism and Equation 6.5 cannot be used to define σ. Such a case can occur, e.g., when τ contains regions of constant value, extremal points, or other points with vanishing $\det(D\tau)$. To begin, the case of constant value is considered. Figure 6.3 illustrates the example of a piecewise constant function for dimensionality $m = n = 1$. In the spatial domain, a piecewise constant function can be modeled as $\tau(x) = \sum_i \tau_i \chi_i(x)$, with the characteristic function

$$\chi_i(x) = \begin{cases} 1 & \text{if } x \in \text{Cell}(i) \\ 0 & \text{else} \end{cases} .$$

It is assumed that the spatial domain is partitioned into cells. Then, τ has constant value τ_i within cell i. Assuming constant density in the spatial domain, $s = 1$, the density in the data domain is:

$$\sigma(\xi) = \sum_i \delta(\xi - \tau_i) \, \text{Size}(\text{Cell}(i)) \, , \tag{6.6}$$

with Dirac delta δ. The correctness of Equation 6.6 can be verified by plugging Equation 6.6 into the mass-conservation Equation 6.3:

$$\begin{aligned} \int_\Phi \sigma(\xi) \, \mathrm{d}^n \xi &= \sum_i \int_\Phi \delta(\xi - \tau_i) \, \text{Size}(\text{Cell}(i)) \, \mathrm{d}^n \xi \\ &= \sum_{\text{for all } i \text{ where } \tau_i \in \Phi} \text{Size}(\text{Cell}(i)) = \int_{\tau^{-1}(\Phi)} 1 \, \mathrm{d}^n x \, . \end{aligned}$$

Since $s = 1$, Equation 6.3 is correctly met. Note that $s = 1$ was chosen to simplify notation; an analogous proof would work for any choice of s, assuming that Equation 6.6 includes s.

A different approach is taken for the other problematic case where $\det(D\tau) = 0$ at isolated parts. To be more precise, in this case, the determinant of the derivative vanishes at isolated null-sets (null-sets with respect to integration in the spatial domain). Here, a two-step approach is performed. First, regions where $\det(D\tau)(x) = 0$ are identified. These regions are denoted $\Gamma = \{x \in \mathbb{R}^n | \det(D\tau)(x) = 0\}$. Second, the null-set Γ and its image under the map τ are removed from the computation of density in Equation 6.5, i.e., σ is not defined at those locations. Since integration over null-sets always yields 0, those isolated locations can be removed without affecting the conservation-of-mass model.

The same approach is taken if the underlying data set is not continuous, i.e., τ is not differentiable (e.g., between cells). This results again in null-sets, which can be removed from the computation of the density. Finally, in theory, τ might be non-diffeomorphic on non-nullsets. However, this case is not considered, since any realistic data set will meet the requirement that τ is non-differentiable or has vanishing $\det(D\tau)$ at the most at isolated null-sets. In

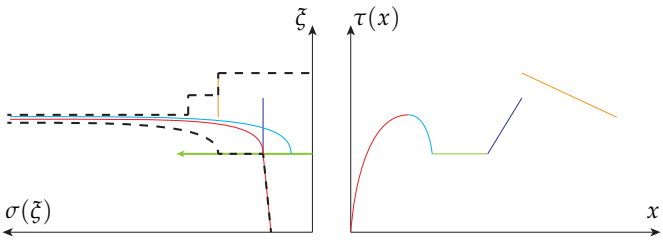


Figure 6.4: General example of a continuous histogram (left) for the function τ (right). The spatial domain of τ is partitioned into intervals with non-vanishing derivative; intervals are color-coded (right image). Respective densities σ are shown in the left image (same color coding). The black dashed curve shows the sum of the intermediate densities σ, which should be overlaid with the δ peak (green) corresponding to the constant part of τ.

particular, any grid-based data set with piecewise cell-oriented interpolation (which is most common in scientific visualization), meets this requirement.

To summarize the special case of $m = n$, σ can be computed by the following schematic algorithm.

1. For an extended volume (i.e., not a null-set) where τ is constant, a δ peak is associated that is weighted by the size of the volume.
2. Isolated null-sets of non-diffeomorphism are identified and used to partition V into volumes where $\det(D\tau) \neq 0$. The null-sets themselves are removed from the computations, and intermediate σ are computed for each element of the partition.
3. The intermediate results from steps one and two are added. This is valid because the defining Equation 6.3 is linear.

Figure 6.4 illustrates the schematic algorithm for an example of a continuous histogram with $m = n = 1$ and with constant density s in the spatial domain.

This construction of continuous histograms allows the reproduction of discrete histograms. Please note that discrete input data has no attached spatial domain in traditional statistical analysis. Discrete data points are identified with arbitrarily positioned cells in the spatial domain; all cells are chosen with equal size. Then, the continuous histogram consists of δ peaks at the data point values (see construction in Equation 6.6), i.e., δ peaks in the continuous histogram correspond to entries in the discrete histogram. Typical bucketing of discrete histograms can be achieved by integrating the continuous histogram with box functions that represent the buckets.

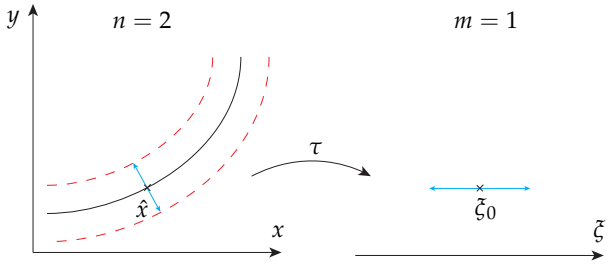


Figure 6.5: Projection issue for the case $m < n$. A small interval (blue) in the data domain around $\tilde{\xi}_0$ (right) corresponds to a bent strip (outlined in red) around the isocontour (solid center line) with isovalue $\tilde{\xi}_0$ in the spatial domain (left). The bent strip is the union of the normal spaces for all points on the isocontour. The blue arrows in the left image mark the normal space at point $\hat{x}$.

6.1.3 Case m < n

Another common case is $m < n$, which arises for example for a 2D scatterplot that shows two scalar attributes of a 3D volume data set. Here, the dimension of the spatial domain is reduced when τ maps to the data domain. Therefore, the transformation theorem for integration, which was used in the previous subsection, does not apply. In particular, $\det(D\tau)$ does not exist, which means that the same approach cannot be used as in Equation 6.4.

Figure 6.5 illustrates the geometry of the underlying problem for the case $n = 2$ and $m = 1$: a single point in the data domain corresponds to an infinite set of points in the spatial domain. In this example, the infinite set is the isoline within the spatial domain that corresponds to the isovalue in the data domain. This imbalance in dimensionality does no longer permit the transformation of differentials as in the previous subsection.

As in the generic discussion of Section 6.1.1, two related volumes are considered in the data and spatial domains: $\Phi \subset \mathbb{R}^m$ and $V = \tau^{-1}(\Phi) \subset \mathbb{R}^n$. To be more specific, only a small neighborhood Φ around a point $\tilde{\xi}_0 \in \mathbb{R}^m$ is considered. To overcome the dimensionality problem, V is split into two parts: first, the inverse image of the point $\tilde{\xi}_0$ (i.e., a generalized isocontour), and second, the perpendicular space around the inverse image of x_0. $\tau^{-1}_{\text{normal}}(p)$ is denoted as the space that is normal to $\tau^{-1}(\tilde{\xi}_0)$ at a point $p \in \tau^{-1}(\tilde{\xi}_0)$ and that is also contained within $\tau^{-1}(\Phi)$. Here, τ is assumed to be a smooth non-constant function, and therefore, the isocontour is smooth as well and the normal space is well defined. If τ is not smooth, then the spatial domain is split into piecewise smooth regions and the method is applied in a piecewise manner (this

approach does not apply to completely non-smooth functions). If τ is (partly) constant, such a constant volume is separated out of the computation similar to the discussion in Section 6.1.2, leading to δ contributions. In the regular case, by construction, the normal space has the same dimension m as the data domain, whereas the isocontour has dimension $n - m$. Figure 6.5 illustrates the isocontour and the normal space. Now, the integration in Equation 6.3 can be split into isocontour and normal parts, similar to the approach of the co-area computation [Fed96]:

$$\int_{\Phi} \sigma(\xi)\,\mathrm{d}^m\xi = \int_{\tau^{-1}(\Phi)} s(x)\,\mathrm{d}^n x = \int_{\tau^{-1}(\xi_0)} \left(\int_{\tau^{-1}_{\text{normal}(\hat{x})}} s(\tilde{x})\,\mathrm{d}^m\tilde{x} \right) \mathrm{d}^{(n-m)}\hat{x}\,. \quad (6.7)$$

Within the normal space, the same approach can be used as in Section 6.1.2 because dimensionalities coincide and a computation based on derivatives is possible. Denoting the intermediate contribution to the density at points $\hat{x}$ by $\tilde{\sigma}_{\hat{x}}$, the inner integration is re-labeled in the right-hand side of Equation 6.7 according to:

$$\int_{\tau^{-1}_{\text{normal}(\hat{x})}} s(\tilde{x})\,\mathrm{d}^m\tilde{x} \stackrel{!}{=} \int_{\Phi} \tilde{\sigma}_{\hat{x}}(\tilde{\xi})\,\mathrm{d}^m\tilde{\xi}\,.$$

Assuming a diffeomorphism in the normal space, Equation 6.5 can be adopted to obtain:

$$\tilde{\sigma}_{\hat{x}}(\tilde{\xi}) = \frac{s(\hat{x})}{|\mathrm{Vol}(D\tau)(\hat{x})|}\,,$$

where $\hat{x}$ and $\tilde{\xi}$ are related by $\tau(\hat{x}) = \tilde{\xi}$. Here, $|\mathrm{Vol}(D\tau)|$ replaces the determinant $|\det(D\tau)|$ in Equation 6.5. The volume measure $|\mathrm{Vol}(D\tau)|$ is defined as the volume spanned by the partial derivatives of τ restricted to variations of parameters in the normal space $\tau^{-1}_{\text{normal}(\hat{x})}$. Figure 6.5 (left) illustrates the reciprocal volume measure $1/|\mathrm{Vol}(D\tau)|$ within the spatial domain. The volume measure is explicitly computed in Section 6.1.4 for the special case $m = 2, n = 3$, and below in this subsection for the case $m = 1, n = 3$. For the final overall density, $\tilde{\sigma}_{\hat{x}}$ is integrated along the complete isocontour:

$$\sigma(\xi_0) = \int_{\tau^{-1}(\xi_0)} \frac{s(\hat{x})}{|\mathrm{Vol}(D\tau)(\hat{x})|}\,\mathrm{d}^{(n-m)}\hat{x}\,, \quad (6.8)$$

which completes the generic discussion of the case $m < n$.

Here, a comparison with the work on isosurface statistics by Carr et al. [CDD06] is included because histograms similar to Carr et al. can be produced by using $m = 1$ and $n = 3$. The work from Carr et al. focuses on analyzing isosurface behavior, whereas continuous scatterplots target visual data analysis in the full domain. This is the reason for different definitions of histograms. Carr et al. define their histogram as the volume of the inverse

image, i.e., the area of the respective isosurface. Using the notation introduced in Section 6.1.1, their histogram would read $\sigma(\xi_0) = \int_{\tau^{-1}(\xi_0)} 1 \, \mathrm{d}^{(n-m)}\hat{x}$ instead of the computation in Equation 6.8. Both approaches use the size of the isosurface (here, via integration over $\tau^{-1}(\xi_0)$)).

One (minor) difference is that continuous scatterplots support a space-variant input density s. The major difference, however, is that in this thesis $1/|\mathrm{Vol}(D\tau)(\hat{x})|$ is taken into account, whereas Carr et al. do not. For $m = 1$ and $n = 3$, $|\mathrm{Vol}(D\tau)(\hat{x})|$ is the magnitude of the gradient at $\hat{x}$. Put differently, the neighborhood of values in the data domain and how they are affected by derivatives of τ is considered by continuous scatterplots, whereas Carr et al. use a point mapping from the data domain to the spatial domain. In this sense, the approach presented there is related to, but no identical with, Legendre transformations that take into account derivatives (see the geometric interpretation of the Legendre transformation in [CH62, pp. 32–39] and its use for Hamiltonian and Hamilton-Jacobi mechanics described by Goldstein [Gol80]). Therefore, this definition of continuous scatterplots takes into account the behavior of τ in its full neighborhood; only in this way, it is possible to represent the transformation of sampling density. Scheidegger et al. [SSD+08] have independently derived the same weighting factor of $1/||\nabla\tau(\hat{x})||$ when revisiting Carr et al. [CDD06]; i.e., for the case of isosurface histograms, both approaches lead to the same result.

A formal, mathematical advantage of this model in Equations 6.2 and 6.3 is its generic applicability to any dimension of the spatial and data domains. In particular, densities s and σ are automatically adapted to the respective integration dimensions n and m—if actual physical units of mass density were used, they would have SI units (Système International d'unités) [kg/m^n] and [kg/m^m], respectively. Moreover, even (partly or completely) constant functions τ are supported in this model; in this case, δ distributions occur in σ. In contrast, the formal derivation of Equation (5) by Carr et al. [CDD06] is problematic because the integration measure in their expression $\int_{f^{-1}(h)} 1 \, \mathrm{d}x$ is not explicitly specified but, from the context of their Equation (3), should be d-dimensional. Typically, $f^{-1}(h)$ is an isosurface, which is a null-set, and therefore the integral vanishes. Alternatively, the integration measure could be adapted to the dimensionality of $f^{-1}(h)$. Then, their integration in Equation (5) would be fine as long as the dimensionality of $f^{-1}(h)$ is constant, i.e., problems would occur when the function is partly constant, leading to a mix of 2D isosurfaces and 3D isovolumes. Scheidegger et al. [SSD+08] resolve these problems by restricting the computation of isosurface histograms to 3D scalar fields with non-vanishing gradient, implying integration on 2D isosurfaces. The above issues with continuous distributions demonstrate the usefulness of the presented density-based definition of generic continuous scatterplots.

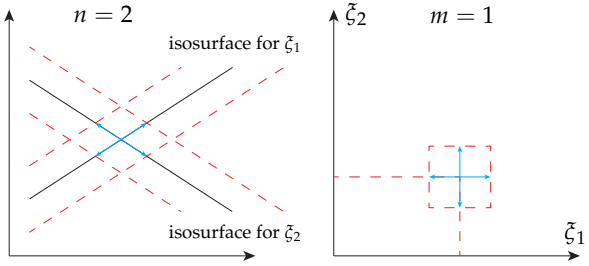


Figure 6.6: In the data domain, two values ξ_1 and ξ_2 are chosen (shown on the right side). These values correspond to isosurfaces represented by two solid lines in the spatial domain (shown on the left side). The areas denoted by the small blue arrows in the two domains correspond to each other. The reciprocal volume $1/|\text{Vol}(D\tau)|$ is the quadrilateral that results from the intersection of the two stippled stripes around the isosurfaces (left).

6.1.4 Case m = 2 and n = 3

This subsection addresses a special case of the above subsection: $m = 2$ and $n = 3$. This case is important in practical applications because typical data is given on a 3D spatial domain and analyzed by 2D scatterplots. The other important practical application is the computation of 1D histograms for data on 3D spatial domains; this application was covered at the end of the previous subsection.

For simplicity of discussion, it is assumed that τ is a smooth non-constant function so that for any choice of coordinates in the data domain, $\xi = (\xi_1, \xi_2)$, two smooth isosurfaces corresponding to ξ_1 and ξ_2 are obtained (for (partly) constant or non-smooth τ, a special treatment similar to Section 6.1.2 is required). Furthermore, τ is assumed to be non-degenerated so that the intersection of the two isosurfaces yields 1D curves. Figure 6.6 illustrates the geometry of the scenario. Here, a zoomed-in view is shown; therefore, the smooth isosurfaces appear planar. In this case, Equation 6.8 reads

$$\sigma(\xi_0) = \int_{\tau^{-1}((\xi_1,\xi_2))} \frac{s(\hat{x})}{|\text{Vol}(D\tau)(\hat{x})|} \, \mathrm{d}\hat{x} \,, \tag{6.9}$$

where the integration is along the 1D intersection of the two isosurfaces. The 2D area $|\text{Vol}(D\tau)|$ in Equation 6.9 is spanned by the gradients $\partial\xi_1/\partial x$ and $\partial\xi_2/\partial x$. Figure 6.6 illustrates the respective reciprocal volume $1/|\text{Vol}(D\tau)|$ carved out around the two isosurfaces. By using vector computations, the vol-

ume measure is computed as the cross product of the two gradients:

$$|\mathrm{Vol}(D\tau)| = \left|\left| \frac{\partial \xi_1}{\partial x} \times \frac{\partial \xi_2}{\partial x} \right|\right| . \tag{6.10}$$

In summary, the density σ is obtained by integration along the 1D intersection curves of the two isosurfaces, weighted by the reciprocal of the magnitude of the cross product of the two respective gradients.

6.1.5 Case m > n

This subsection briefly discusses the remaining uncovered case $m > n$ in order to complete the description of cases. From a visualization point of view, this case is not very useful because the dimensionality of the scatterplot is higher than the dimensionality of the spatial domain, which adds visual complexity instead of reducing it.

The map τ from the spatial domain $\mathbb{R}^n$ to the data domain $\mathbb{R}^m$ leads to a coverage of the data domain by an n-dimensional subset. For example, a 1D spatial data set would typically result in a 1D curve within a 2D scatterplot, i.e., the support for the density σ would be that curve. The density distribution σ can be computed by applying the mapping from Section 6.1.2 within the support of σ, considering this support as an n-D manifold, and by allowing for δ distributions to obtain finite values when integrating over null-sets in the data domain.

6.2 Scatterplot Algorithm for Tetrahedral Meshes

In practice, the most important examples of continuous scatterplots are the cases $n = 3$, $m = 1$ (i.e., continuous histogram) and $n = 3$, $m = 2$ (continuous 2D scatterplot). Both cases work on a 3D spatial domain, which is common for scientific data. Even for time-dependent 4D data, visualization is often restricted to showing 3D time slices. The case of continuous histograms can be implemented similar to Carr et al. [CDD06]; the only difference is the additional weighting by the reciprocal of the gradient magnitude and by the original density s. The extension of Scheidegger et al. [SSD+08] already includes the weighting by the reciprocal of the gradient magnitude and, thus, their implementation could be directly adopted. Therefore, this section focuses on the other case—the construction of continuous 2D scatterplots.

According to Equations 6.9 and 6.10, the intersection curve of two isosurfaces as well as the two gradients along the intersection curve need to be determined and combined by integration along the curve. The result of this computation depends on the functional behavior of the data field τ. Typically, volumetric

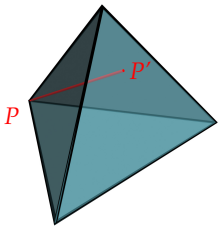


Figure 6.7: This illustration shows how the distance is measured that is used to determine the density. At point P, a depth value that corresponds to the density must be computed. This is done by calculating the distance between P and P' in the spatial domain of the tetrahedron. The point P' is the intersection point between the face opposite to P and the (ξ_1, ξ_2) isoline through P. (Please note that P' does not coincide with a vertex, except for degenerated cases.)

data is given on a grid, and τ is reconstructed by piecewise cell-based interpolation within the grid. The focus lies on tetrahedral grids because they are naturally equipped with linear (barycentric) interpolation.

Since the overall density σ is based on the linear model of Equations 6.2 and 6.3, σ can be constructed by linear superposition of the contributions from tetrahedral cells. Therefore, the remaining question is how to compute σ for a single tetrahedron. Here, the linearity of barycentric interpolation simplifies the computation substantially because of the following reasons: first, isosurfaces within a tetrahedron are planes. Therefore, the intersection between two isosurfaces is a straight line (in the non-degenerate case). Second, the gradient within the cell is constant. Thus, the volume measure from Equation 6.10 is constant as well. In conclusion, σ is obtained by computing the length of the intersection of the two isosurfaces and dividing that value by the constant volume measure. Here, a constant density s is assumed in the spatial domain.

The projected tetrahedra algorithm by Shirley and Tuchman [ST90] is adopted to compute the isosurface intersection. The original algorithm is designed for volume rendering of scalar fields on tetrahedral grids by projecting tetrahedra onto the image plane. However, the image plane is located within the spatial domain, whereas 2D scatterplots need to project the tetrahedra to the data domain. This projection is achieved by interpreting (ξ_1, ξ_2) as coordinates for orthographic projection. The Shirley- Tuchman algorithm partitions the image footprint of the tetrahedron into a collection of up to four triangles, depending on the viewing direction. Within each triangle, parameters are interpolated linearly. The same kind of triangle partitioning is used for scatterplots. Here, the linearly interpolated parameter is the geometric depth of the tetrahedron in the spatial domain, computed at the corresponding data values (ξ_1, ξ_2). Figure 6.7

illustrates the computation of depth. Linear interpolation of depth within the triangle is correct because the underlying 3D geometry is linear as well. The final σ value is obtained by dividing depth by the volume measure from Equation 6.10.

The algorithm consists of the following steps:

1. Classify the tetrahedron based on its silhouette in the data domain. This step yields up to four triangles.
2. Attach data values (ξ_1, ξ_2) as 2D geometric coordinates to the triangle vertices.
3. Determine the volume measure according to Equation 6.10.
4. Compute the Euclidean distance between frontface and backface of the tetrahedron at the vertices. Attach this distance divided by the volume measure as texture coordinate to the vertices.
5. Render triangles. The volume-weighted distance is interpolated during scanline conversion and yields the density at the current fragment. Output the result to the framebuffer.

If a data value is constant within the tetrahedron, the corresponding σ is no longer a regular function, but a δ distribution. Such δ distributions are approximated by assigning a very large value (a constant defined within the implementation, e.g., the maximum number which can be represented as a floating point variable). In this way, even degenerate cases can be handled.

The overall density σ is obtained by rendering all tetrahedra with additive blending. Since additive blending is commutative, triangle sorting is not necessary prior to rendering. Finally, color mapping is applied by using a color lookup table in order to generate the final result.

The above algorithm works for any tetrahedral grid. For non-simplicial grids, cells are decomposed into tetrahedra before rendering. In the common case of hexahedral cells, decomposition in five tetrahedra per cell is employed. Since any 3D grid can be approximated by triangulation, any grid-based data set can be processed.

The projected tetrahedra algorithm lends itself to acceleration by graphics hardware because rasterization of triangles and blending are efficiently supported by graphics hardware. Similarly, the 2D scatterplot algorithm can be implemented using graphics hardware. Steps 1–4 of the above algorithm are performed on the CPU, similar to traditional implementations of the Shirley-Tuchman algorithm. The results from all tetrahedra are combined by additive blending within a render-target texture. For appropriate blending quality, the format for the frame-buffer and respective textures is 32-bit floating-point. The render-target texture is used as input to another render process that applies the color table (implemented as 1D dependent texture) to σ to generate the final image. The results of this approach are presented in Section 6.4.1.

6.3 Parallelized Computation of Continuous Scatterplots

The approach presented in Section 6.2 is implemented on the CPU, which, depending on data set size, leads to computation times well beyond interactive frame rates. The time to compute a scatterplot may take up to several minutes—too long to efficiently use continuous scatterplots in a visualization system. To overcome this issue, a modification of the original algorithm was implemented that enables the parallelized construction of continuous scatterplots on the GPU. To perfectly suit the architecture of modern GPUs, tetrahedra are processed by CUDA compute threads.

In order to increase the computational performance of continuous scatterplots, CUDA is used to execute a modified version of the original continuous scatterplot algorithm on the GPU. Directly porting the algorithm from CPU to GPU would be highly inefficient, due to the different architecture and programming model of GPUs. On the GPU, complicated data structures should be avoided and execution paths should not branch in order to achieve optimal performance. The modifications that are necessary to compute continuous scatterplots on the GPU are explained in the following paragraphs. The modified algorithm to compute continuous scatterplots is split into two parts: First, a preprocessing step is performed on the CPU. After this step, the actual computations that are necessary to render the continuous scatterplot are performed on the GPU.

To begin, it is necessary to introduce the notion of classes for tetrahedra. The original classification of tetrahedra was presented by Shirley and Tuchman [ST90]. However, for this approach, a simplification of this classification is used that was proposed by Wylie et al. [WMFC02]. In their work, only classes one and two are used for classification, since the remaining classes are only degenerate cases of these first two classes. In Figure 6.8, the two main classes are shown for reference.

To compute a continuous scatterplot, the density of each tetrahedron needs to be computed—which in turn requires a separate handling of tetrahedra based on their class. To achieve optimal performance for density computations on the GPU, the input tetrahedra are sorted in two lists which only contain tetrahedra of one of the two classes. This is necessary because on the GPU, diverging threads in a CUDA warp force the whole warp to serially execute each branch which results in significant computational overhead. Please note that sorting is a preprocessing step which is necessary only once per data set. After this step, the actual computations for the continuous scatterplot are performed. Depending on the resolution of the data set, not all tetrahedra can be uploaded to the GPU at once. If this is the case, the list of input tetrahedra is split into data subsets that fit into GPU memory. A serialized processing of these data subsets is trivial since individual tetrahedra are rendered using additive blending—therefore, the resulting projected tetrahedra are accumulated

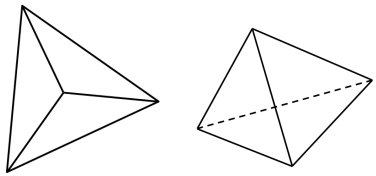


Figure 6.8: The two main classes of projected tetrahedra. Class one is shown in (a), whereas (b) shows class two. Additional classes exist, but are only degenerate cases of these two classes.

in the frame-buffer until all tetrahedra are processed.

Due to the preprocessing step, the input for each CUDA kernel is a single tetrahedron of the same class. The CUDA kernel has to compute a topology for the four input vertices in order to construct triangles for output. This computation of the topology is based on the algorithm by Wylie et al. [WMFC02]. This algorithm computes the triangle topology of the projected tetrahedra based on a series of tests applied to the four input vertices of a tetrahedron. In addition, the density of the input tetrahedron is determined in order to simulate the attenuation of light for the volume rendering. The main difference lies in the computation of this density; here the formula presented in Section 6.1.4 is applied. The resulting density is encoded as a color value and assigned to the corresponding vertices of projected tetrahedra. Resulting triangles of the currently processed chunk are stored in a vertex buffer object, which can be directly rendered using OpenGL in combination with shaders. These shaders are necessary since the color value of the vertices is stored in their z-coordinate—for a 2D continuous scatterplot, only two coordinates are necessary to specify the location in the scatterplot domain, therefore the z-coordinate is free to be used for such a purpose. The shader removes the color value from the z-coordinate and uses it as an input value for a color-lookup table which determines the final appearance of the continuous scatterplot. Results of this approach are shown in Section 6.4.2.

6.4 Application

In this section, examples of both discrete and continuous scatterplots are provided for different visualization examples. Please note that constant input density $s = 1$ is used for all examples. The scatterplot functionality is part of a multi-attribute visualization tool that also supports multiple coordinated views, brushing and linking, and volume visualization. For continuous scatterplots, the implementation described in the previous section was used. For discrete scatterplots, a similar GPU-based implementation was applied; here, points of the scatterplot are rendered via point sprites.

Please note that in contrast to the previous chapters, vector fields are not vi-

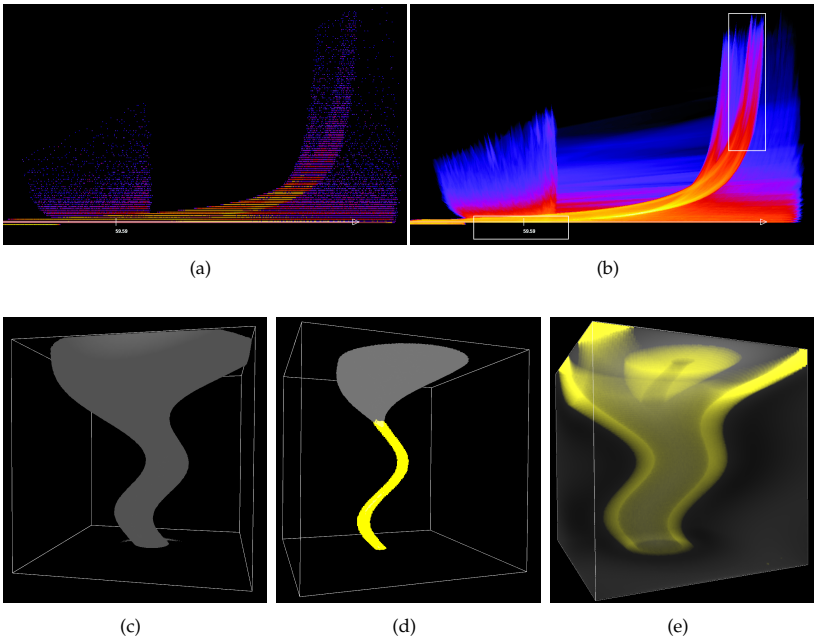
sualized directly with continuous scatterplots. Instead, multi-attribute fields are analyzed, i.e., vectors contained in the data may be decomposed to their individual data dimensions. An example of this approach can be seen in Section 6.4.1, where the "Tornado" data set is analyzed by mapping the z-component of the velocity data to the vertical axis of the scatterplot.

6.4.1 Qualitative Results

The first example in Figure 6.9 shows the "Tornado" data set. This data set is commonly used in flow visualization as a benchmark data set. It represents the 3D velocity field of air flow of a simplified tornado. Data is given on a uniform grid of resolution 128^3. The two resulting scatterplot variants are compared in the upper part of Figure 6.9. For this data set, several data dimensions are available for visualization in the scatterplot. In Figure 6.9, the magnitude of the velocity is mapped to the horizontal axis and the velocity in z-direction to the vertical axis. In this way, different features of the "Tornado" can be extracted, e.g., the inner part of the vortex region (Figure 6.9b)) or the outer boundary of the vortical structure (Figure 6.9c)). Furthermore, brushing and linking is demonstrated for the "Tornado" data set. During this process, features in the scatterplot are identified and selected using a selection rectangle. In the volume visualization, voxels are highlighted if they correspond to the selected area defined in the scatterplot. Two different selections were made and their results are shown in the lower part of Figure 6.9.

The next example is the IEEE Visualization 2004 Contest data set "Hurricane Isabel", depicted in Figure 6.10. This data set is shown in two different resolutions—a down-sampled version with a size of $128 \times 128 \times 30$ and the original data set with a size of $500 \times 500 \times 100$. Air temperature is mapped to the horizontal axis, whereas air pressure is mapped to the vertical axis. This example shows that continuous scatterplots are structurally independent of the resolution of the data set. In particular, the discrete scatterplot of the low-resolution data set induces misleading structures (i.e., the slanted, nearly vertically aligned clusters of points), which are not part of the data but due to the low sampling resolution. Moreover, while the visual result of a discrete scatterplot depends on the size of the individual points, continuous scatterplots are parameter-free.

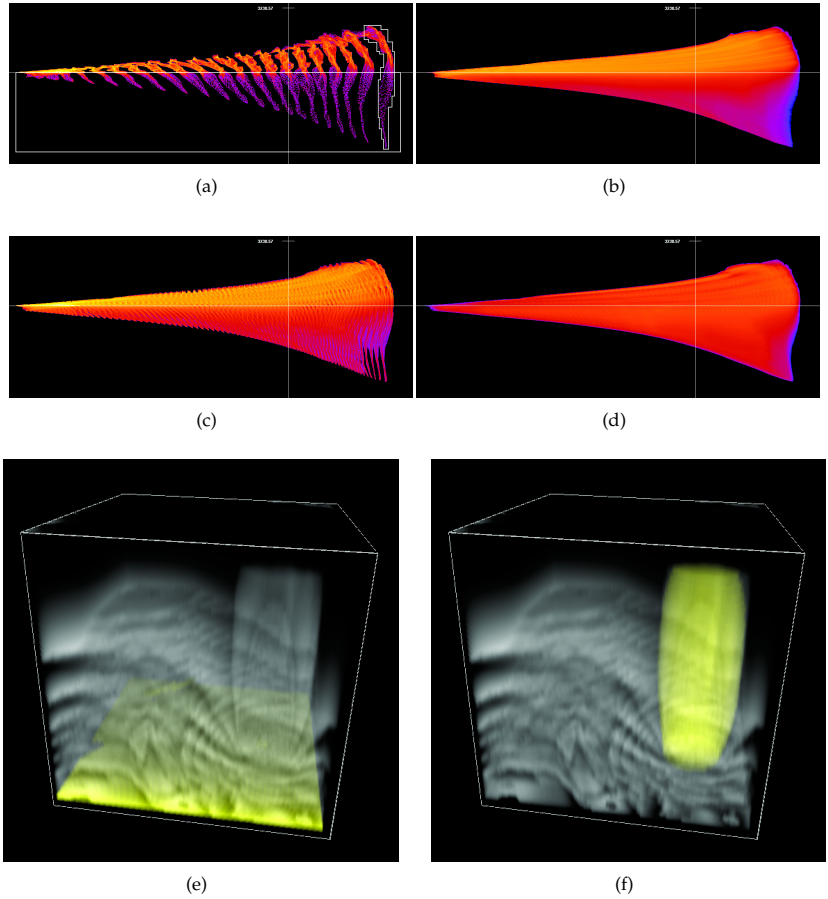
The third example in Figure 6.11 shows scatterplots that support the user-guided specification of 2D transfer functions for volume rendering [KKH02]. The scatterplots visualize the "Blunt Fin" data set, which is given on an unstructured grid derived from a curvilinear grid of resolution $40 \times 32 \times 32$. Since this data set contains only one scalar value representing the velocity of the flow, the second data dimension is obtained by computing the magnitude of the gradient of the scalar velocity (a more detailed description is given in Section 1.6). Although both types of scatterplots show arc-like patterns, differ-



(a) (b)

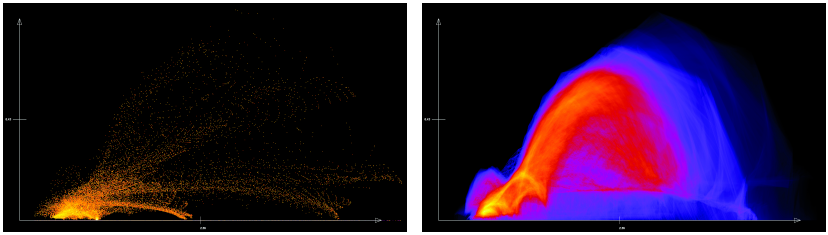
(c) (d) (e)

Figure 6.9: In the upper part, both types of scatterplots are shown for the "Tornado" data set. The discrete scatterplot is shown in (a), the continuous version in (b). The lower part shows three volume visualizations of the data set. The lower-left image (c) shows the tornado visualized by a representative isosurface of velocity magnitude. The image in the middle (d) shows highlighted voxels (yellow) that were marked in the continuous scatterplot. This highlighting corresponds to the upper-right selection rectangle in the continuous scatterplot. The other selection rectangle in the lower-mid part of the continuous scatterplot highlights different voxels, as shown in the lower-right volume-visualization image (e). In image (e), highlighted voxels (yellow) and the velocity magnitude are simultaneously visualized by rather transparent volume rendering in order to show selected features at different depths. Therefore, we can see that different voxels than in (d) are highlighted, especially not the ones in the center of the tornado.



(a) (b) (c) (d) (e) (f)

Figure 6.10: Comparison of discrete (a, c) and continuous (b, d) scatterplots for the "Hurricane Isabel" data set. The upper row shows scatterplots for the low-resolution version of size $128 \times 128 \times 30$. The middle row shows the scatterplots for the data set in its original size of $500 \times 500 \times 100$. The left image in the lower row shows highlighted voxels (yellow) that were marked in the upper left scatterplot by the non-rectangular selection area. The highlighted voxels cover the bottom slice of the volume, showing that the prominent structures of the discrete scatterplot are simply related to the low resolution in z-direction. The lower right image shows the selection of all negative air pressure samples (selected by the large white box in the upper left scatterplot). The highlighted voxels lie exclusively in the center of the hurricane.



(a) (b)

Figure 6.11: The discrete scatterplot of the "Blunt Fin" data set is shown in (a), whereas (b) is the continuous version. Both types of scatterplots visualize the scalar data value along the horizontal axis and the magnitude of the gradient along the vertical axis. Choosing these data dimensions, material and boundary identification is possible by finding arc-like structures.

ences between discrete and continuous scatterplots are clearly visible. Unlike the discrete scatterplot, the continuous version provides a dense visualization that allows spotting of interesting features more easily than in the discrete representation. The discrete scatterplot just uses the data at the grid points and ignores the underlying grid structure, whereas the continuous scatterplot takes into account the varying size and shape of grid cells by computing gradients within cells. Therefore, differences between discrete and continuous scatterplots may be particularly pronounced for unstructured or curvilinear grids compared to uniform grids with their constant cell size.

For all three examples, continuous scatterplots show better visual quality than discrete scatterplots. In particular, discrete scatterplots tend to miss visual information in plots; those visual gaps require extra mental work by the user in order to close those gaps. In addition, some features are glossed over or are completely missing in discrete scatterplots. In contrast, continuous scatterplots provide guaranteed coverage of the relevant parts of the scatterplot domain and, thus, cannot miss important features.

6.4.2 Performance Results

The main goal of Section 6.3 was to speed up the computation of continuous scatterplots. In this section, the original CPU implementation is compared with the GPU implementation. Both approaches are comparable since exactly the same data is processed, and the same result is produced. For the time measurements a computer was used with an Intel CPU running at 2.4 GHz and an NVIDIA GeForce 8800 GTX graphics card. All continuous scatterplots were created for a viewport size of 1024×768 pixels. The measurements were per-

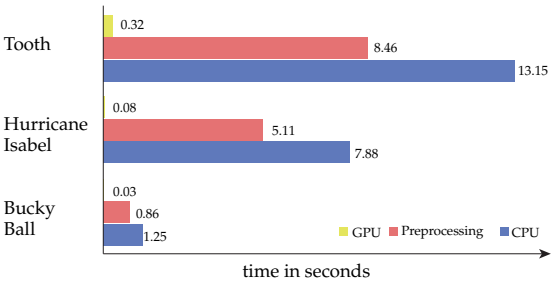


Figure 6.12: Comparison of CPU and GPU implementation for three data sets. The time which is needed to compute a continuous scatterplot is measured in seconds with identical parameters for both approaches.

formed for three different data sets and the results are shown in Figure 6.12.

The first data set is the "Bucky Ball", an artificial volume data set with a resolution of 32^3. The data set itself contains scalar values, which are mapped to the horizontal axis of the continuous scatterplot. The second data dimension is derived according to the approach described in Section 1.6. The "Bucky Ball" data set is very small, resulting in short computation times for the continuous scatterplot. The CPU implementation needs 1.25 s to finish, whereas the GPU version takes only 0.03 s. The preprocessing step for the GPU version needs 0.86 s for this data set. Taking this into account, the GPU version is 1.4 times faster. However, once the pre-sorting is done, the GPU version is approximately 41 times faster.

The second data set is "Hurricane Isabel", where several data dimensions are available at a resolution of $100 \times 100 \times 20$. As in Section 6.4, air temperature is mapped to the horizontal axis and air pressure to the vertical axis. For this data set, the CPU version takes 7.88 s to compute a continuous scatterplot. The GPU version needs 0.08 s to perform the same task, after a precomputation step which takes 5.11 s. Taking the precomputation step not into account, the GPU version is approximately 98 times faster than the CPU version.

The third and last data set is a CT scan of a human tooth. This data set has a resolution of $64 \times 64 \times 80$. Again, the scalar values of the data set are mapped to the horizontal axis, whereas the magnitude of the gradient of these scalar values is mapped to the vertical axis. For this data set the CPU version needs 13.15 s to produce the final result. The GPU version needs 8.46 s for the precomputation step and 0.32 s for the actual computation of the continuous scatterplot. Without considering the precomputation step, the GPU version is approximately 41 times faster than the CPU version.

When taking the preprocessing step into account, the algorithm performed on the GPU with CUDA is up to 35 percent faster than the CPU implementation. This shows that exploiting the massively parallel execution possibilities of modern GPUs pay off for even small data sets. Considering that a user of continuous scatterplots commonly has to recompute the final image several times to finish her tasks, the speed-up is even greater once the preprocessing step is performed. After this step, the computation of a continuous scatterplot is performed with interactive frame rates.