8 Progressive Splatting of Continuous Scatterplots

The preceding chapters presented solutions to speed up the computation of continuous scatterplots. Parallel processing of grid cells shown in Section 6.3 achieves this goal using specialized GPU hardware, whereas the subdivision algorithm of Chapter 7 employs a different computation scheme to reduce necessary computations. Our alternative approach [HBW11], presented in this chapter, is similar to the latter, as it offers a trade-off between output quality and rendering performance, depending on the needs of the user. In this way, these two algorithms create approximations of increasing quality with each iteration.

In this chapter, a forward-mapping technique inspired by splatting for direct volume rendering is employed to compute the influence of samples to the final image and successively approximate the true representation of the plot. High frame rates are the expected outcome for interactive data exploration using continuous scatterplots. In this case, splatted continuous plots allow trading accuracy for rendering performance by resampling the data at a lower sampling rate. This may be implemented by skipping some of the input data points or by resampling at a few, freely chosen positions in the spatial domain. In contrast, for very small data sets, additional samples may be distributed over the spatial domain to improve image quality. A key observation is that the continuous plots are based on samples that are completely independent from the number and positions of the grid points of the data set.

To this end, a novel closed-form analytic description of the splatted footprint of Gaussian input kernels is presented for scatterplots. A new progressive refinement algorithm is introduced that allows obtaining initial results extremely fast and hence complements a very useful property of continuous data plots: for many data sets, a small subsample of the full data may provide a good approximation to the final density distribution, making progressive refinement the ideal algorithm for rendering.

This new construction algorithm relies on splatting. The splatting method by Westover et al. [Wes89] is adopted, modified, and extended. The original method was introduced as a forward-mapping algorithm for direct volume rendering. In contrast to ray tracing techniques where pixel intensities are computed by mapping the image plane to the data space, volumetric splatting computes the contribution of samples from the data space to the image plane. Since ray integration within a sample's reconstruction kernel is independent of its density, the integral can be precomputed for a given view direction. The resulting image plane footprint of the kernel is then used to compute the projected image of all data space samples and only has to be recomputed if the viewing direction changes. For continuous plots, however, the viewing transformation depends on the data, such that footprints have to be computed for every sample individually.

An approach similar to splatting densities has recently been presented by Feng et. al. [FKLT10] for the visualization of uncertain data samples. In their work, a probability density function is estimated using normally distributed, uncorrelated kernels. Hence, all samples in the scatterplot are represented by scaled Gaussian footprints. In contrast, the model presented in this work uses gradient information contained in the data to transform the density distribution of scatterplots. Furthermore, densities are derived analytically to allow for the precomputation of splat footprints.

8.1 Footprint Computation

This section describes the computation of density footprints for reconstruction kernels in the original spatial domain of the data and the respective mapping to scatterplots. Please note that terminology from measure theory is used and thereby weighted, non-normalized densities are used (as opposed to the probability density which is typical for statistics literature).

As introduced in Section 6.1.1, the two main ingredients for the mathematical model of the statistical plots are:

- Density function $s(\vec{x}) : \mathbb{R}^3 \rightarrow \mathbb{R},\ \vec{x} \mapsto s(\vec{x})$, which describes the scalar-valued density over the *spatial domain* of the data set, i.e., the importance of the respective spatial part of the data.
- Map $\tau : \mathbb{R}^3 \longrightarrow \mathbb{R}^n,\ \vec{x} \mapsto \tau(\vec{x})$, which represents the multivariate data set defined on the same spatial domain as above.

The mathematical problem setting can now be formulated as follows: what is the transformed density function in the data domain? This density function explicitly depends on the density s in the spatial domain and is implicitly affected by the map τ, which connects from the spatial domain to the data domains. Figure 8.1 illustrates the domains and the example of mapping a typical splatting kernel.

In the following, splats are first defined in the spatial domain and the map τ is discussed in the context of sampling at the center points of splats. Then, the discussion is restricted to the transformation of a single splat template,

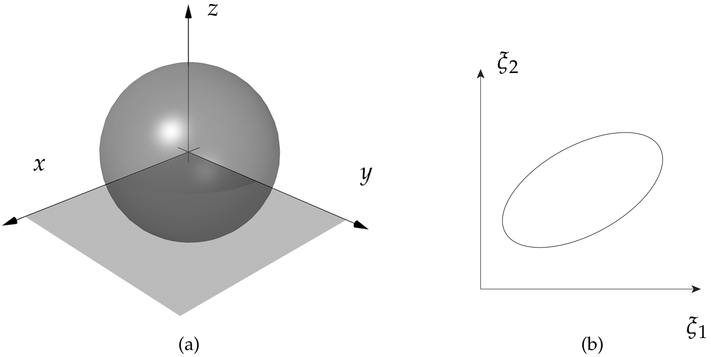

Figure 8.1: (a) The spherical reconstruction kernel in the spatial domain maps to (b) an ellipse in the scatterplot domain.

applying this model to continuous 2D scatterplots.

8.1.1 Spatial Domain and Data-Set Function

Following the splatting approach for direct volume visualization [Wes89], the density in the spatial domain is represented by a weighted sum of kernels at various spatial locations. Then, the overall density on $\mathbb{R}^3$ is given by

$$s_{\text{overall}}(\vec{x}) = \sum_i w_i \, s_i(\vec{x})$$

with scalar-valued weights w_i and kernels s_i. Similar to splatting in volume rendering, the kernels s_i are assumed to be derived from a single template function that is just shifted to different positions. Furthermore, such a template is usually assumed to be spherically symmetric due to reasons of isotropy.

The main idea of splatting is that the template is transformed in a preprocessing step to form a splat. The transformation of the overall density s_{overall} is then computed by overlaying the splats with the same weights w_i. This approach requires that transformation and weighted summation are commutative, which is true for continuous scatterplots because they are computed by a linear operator.

The sample positions $\vec{x}_i$ may be arbitrarily chosen. In particular, they are independent from positions of grid points of the data set. Typically, the sample positions $\vec{x}_i$ are evenly distributed in space, e.g., by putting them on a regular sampling grid or by applying low-discrepancy point sets (see Section 8.3).

The usual case of constant overall density s_{overall} can be implemented by even distribution of sample positions and constant weights w_i.

The kernel s_i is centered at the point $\vec{x}_i$ with relative "radius" k. The Gaussian kernel is popular in volume rendering due to its fast fall-off behavior with increasing distance from the kernel center. Therefore, it is used here as well.

8.1.2 Scatterplot Domain

The density function described by the kernel $s(\vec{x})$ in the spatial domain is now transformed to the corresponding density function $\sigma(\vec{\xi})$ at position $\xi = (\xi_1, \xi_2)^T$ in the 2D data domain. $\sigma(\vec{\xi})$ is also called footprint of the splat or, in short, just splat or footprint.

The footprint can be computed according to Equation 6.9—since τ is linear, the partial derivatives are constant and the volume measure can be moved outside the integral. Heinrich et al. [HBW11] describe in detail how such a footprint of any sample can be transformed to a generic template footprint using a scaling matrix multiplied by a matrix for rotation around the z-axis with a specific angle.

Although the splat computation for scatterplots resembles the one for volume rendering, there is an important difference: the scatterplot-domain splat additionally depends on τ, which is not the case in volume rendering.

8.2 Sampling and Progressive Refinement

The previous section provided the basis for rendering a single splat in the continuous statistical plot. The overall plot is obtained by applying this process to many different splats that densely cover the spatial domain, leading to an appropriate representation of the overall density by accumulating the splats with additive blending. Besides the data values, the partial derivatives of the data are reconstructed at the splat sample so that the position, size, and orientation of the splat can be determined. In this way, the generic footprint template is transformed according to Section 8.1. Using the inverse transformation, a splat is rendered as a rectangle with the correct texture coordinates and density values. The generic template footprints for scatterplots are discretized in a 2D texture during pre-processing. The 2D standard Gaussian $e^{-\|\vec{x}\|^2}$ is sampled on the uniform grid of the texture, which then serves as a lookup table during rendering.

The splatting technique is further improved by extending it to progressive rendering. Due to the linear superposition of splats, progressively sampled intermediate images can be combined by linear superposition as well. More specifically, a sequence of several, independent continuous plots of low sampling resolution are generated that are then accumulated in a separate image (e.g., an offline rendering target on the GPU). Mass conservation, as described in Section 6.1.1, is guaranteed for the transformed densities by compositing single images I_1 and I_2 using the over operator [PD84] and a fixed value for α:

$$I = \alpha I_1 + (1 - \alpha) I_2 \quad \text{with } 0 \leq \alpha \leq 1.$$

If both intermediate image observe mass conservation (i.e., each has the same overall mass M as accumulated from the densities or all pixels), then the blended image is guaranteed to have identical mass as well because mass is also subject to alpha blending.

The footprints further depend on the choice of the smoothing parameter k, which is a measure for the "radius" or bandwidth of the Gaussian kernel in the spatial domain. This parameter is transported to the data domain, where it affects the extents of the footprints to be drawn. On the one hand, large kernels provide a better coverage of, and higher overdraw on, the reconstructed spatial area and thus allow reducing the sampling resolution. On the other hand, large splats introduce a large error to the overall density, resulting in potentially highly blurred images. Analogously, small kernels produce a more accurate, but potentially non-smooth sampling of the density distribution.

8.3 Results

In this section, the results obtained with splatting are compared with discrete and previous non-splatting continuous plots with respect to rendering performance and visual quality. First, results of all of the three rendering techniques are compared with respect to their visual appearance. Then, some results are shown which were obtained by using resampling and progressive refinement. The relation of splat size and sampling resolution is also investigated with respect to rendering performance and image coarseness. All measurements were produced with an implementation based on C++ and GLSL. The implementation was tested on a PC with an Intel Core 2 Quad CPU running at 2.4 GHz and an NVIDIA GeForce 8800 GTX graphics card.

As discussed in Section 8.2, increasing splat size allows reducing the sampling resolution, while the level of coarseness of the resulting image is approximately maintained. The relationship between these parameters is illustrated in Figure 8.2. Here, every plot was rendered without progressive refinement but with different sampling resolutions and reconstruction kernel sizes. As expected, the images become smoother with increasing number of samples and with increasing splat size. Furthermore, coarseness is approximately maintained when doubling the number of samples with half of the kernel size and vice versa.

Figure 8.3 shows a performance analysis for different splat sizes and sampling

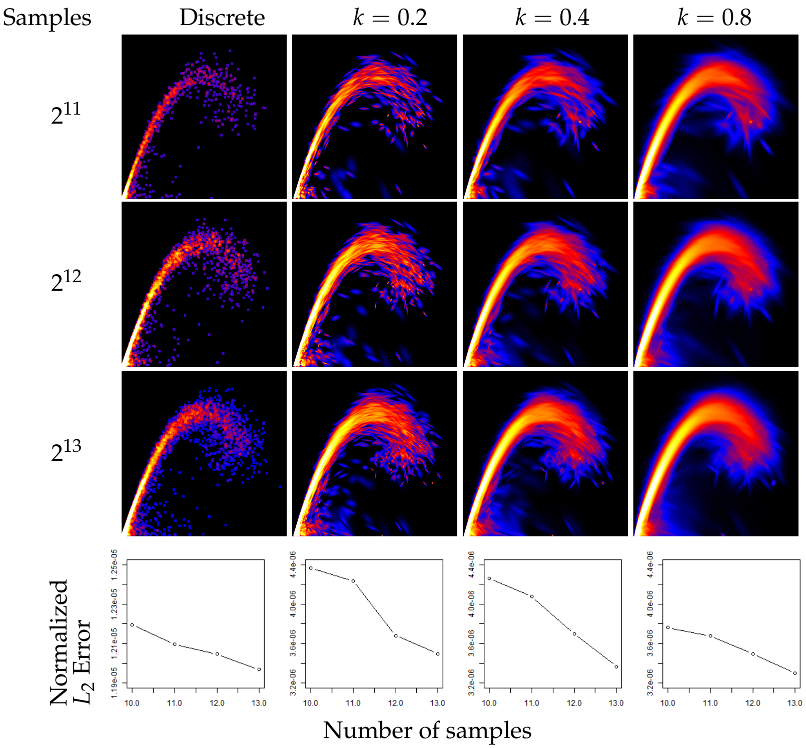

Figure 8.2: The effect of the sampling resolution and the kernel size parameter with respect to the overall coarseness of the plot. In the matrix shown above, the number of samples increases from top to bottom while the splat size increases from left to right. In both cases, the corresponding images become smoother. At the same time, however, increasing splat sizes result in less accurate plots, as can be seen in the rightmost column. Here, the blur introduced by larger splats makes the image appear "wider". The bottom row shows the L_2 error of densities from the splatted plots with respect to a traditional, discrete scatterplot rendered with 10^7 samples. While the convergence behavior is similar for all columns, the total error decreases with increasing smoothing factor k.

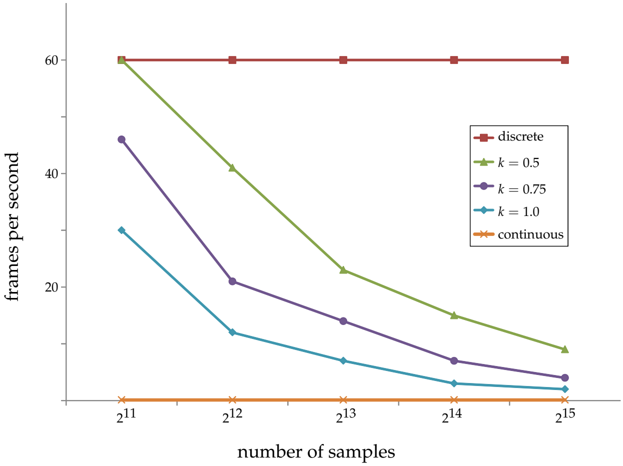

Figure 8.3: Rendering performance for splatted continuous scatterplots of the “Bucky Ball” dataset depending on splat size and sampling frequency. From the plot, a linear dependency between sampling resolution and frames per second can be concluded. Note that, for comparison, the performance of traditional discrete scatterplots and of continuous scatterplots rendered with the projected tetrahedra algorithm are included. The traditional scatterplots are limited to 60 frames per second by the frame refresh rate of the test-hardware. In contrast, the frame rate of continuous scatterplots lies well below one frame per second.

rates, compared with traditional discrete scatterplots and continuous scatterplots using the original projected tetrahedra algorithm. From the measurement data, a linear dependency between sampling rate, splat size, and rendering performance can be concluded (note the $\log_2$ scale of the x axis). Although performance decreases with the number of splats, the progressive refinement algorithm introduced in the previous section can still be used to achieve interactive frame rates, as the total number of splats is divided into successive rendering frames. As a consequence, the algorithm scales well with data set size and can easily be adopted for streaming data, as only a fixed number of samples is required for the individual rendering steps. This also includes large simulation data or time-dependent data, where visualization may be required in real-time. Finally, efficient rendering of statistical plots naturally facilitates the implementation of stacked displays or small multiples, where many instances of the same plotting technique are used to visualize different parts, projections, or subareas of the data set. For example, the scatterplot matrix may be extended using progressive refinement.