7 Adaptive Rendering of Continuous Scatterplots

The practical implementation of 2D continuous scatterplots presented in Chapter 6 is subject to a few limitations. The first limitation is related to the interpolation method used for computing the density in the scatterplot domain. The original implementation of continuous scatterplots follows the idea of projected tetrahedra [ST90]. Within a tetrahedron, barycentric interpolation is applied—this linear interpolation leads to a simplified computation of scatterplot density. For regular grids, the hexahedral cells are decomposed into five tetrahedra to compute a continuous scatterplot. The drawback of this approach, however, is that native trilinear interpolation of regular grids is not supported. The second limitation of the original continuous scatterplot approach is related to the time needed for computing the overall density. Splitting a regular grid into tetrahedra introduces additional overhead; finding the density for average-sized data sets (e.g., in the size range of 128^3) is very time-consuming and may require several seconds (cf. Table 6.12) or even minutes for larger data sets. Since the original scatterplot algorithm is based on projected tetrahedra, its run-time behavior is similar and does not scale well for large data sets given on regular grids, even when executed on GPUs using the technique shown in Chapter 6.3.

To overcome the above drawbacks of the tetrahedra-based scatterplot computation, an additional approach [BW09] was created which does not, in contrast to the one presented in Chapter 6.3, rely solely on parallel execution on GPUs to increase rendering performance. The new approach makes use of ideas of adaptive grid subdivision and hierarchical octree structures to efficiently approximate the scatterplot image. In this way, continuous scatterplots can be computed for arbitrary interpolation or reconstruction functions applied to the data set on the spatial grid. In addition, when compared to the original algorithm for computing a continuous scatterplot, the new approach is simpler and easier to implement. The new method natively supports regular grids, i.e., triangulation is no longer necessary. This is one of the reasons why the time needed to compute the density for a continuous scatterplot is greatly reduced. Furthermore, the new approach is adaptive, and the approximation error introduced when estimating the density contribution can be controlled by a single parameter. This parameter enables the user to balance computation time

and scatterplot quality. The direct support for regular grids, which are most popular in scientific visualization, and the high rendering speed, allows one to seamlessly integrate continuous scatterplots into typical interactive visualization systems.

7.1 Mathematical Approximation Model

This section adds the new approximation model for a fast computation of scatterplots. As shown in Section 6.1.4, a continuous scatterplot needs to render a density function σ defined on the data domain. The problem is that this requires a complicated integration of a potentially varying integrand $s(x)/|\mathrm{Vol}(D\tau)(x)|$. Moreover, the integration domain $\tau^{-1}((\xi_1, \xi_2))$ is the intersection of two isosurfaces within the 3D spatial domain (corresponding to isovalues ξ_1 and ξ_2). Both issues can be directly resolved for the special case of barycentric interpolation in tetrahedral cells, as exploited in Section 6.2. However, for general interpolation or reconstruction functions τ, the computation of Equations 6.9 and 6.10 is non-trivial and might not even have an analytic solution. Therefore, the following approximation strategy is applied. This approximation starts with the observation that generic continuous scatterplots can be derived from an abstract version of mass conservation described in Section 6.1.1 and revisited here:

$$M = \int_V s(x)\,\mathrm{d}^n x = \int_{\Phi=\tau(V)} \sigma(\xi)\,\mathrm{d}^m \xi. \tag{7.1}$$

Here, M describes the "mass" of virtual material in either the spatial domain (left integral) or the data domain (right integral). The term V describes any volume in the spatial domain, and $\Phi = \tau(V)$ is the corresponding volume in the data domain. For the special case of 2D scatterplots of 3D data sets, $n = 3$ and $m = 2$.

Equation 7.1 is used to approximate the density σ by assuming constant distributions of densities s and σ inside those volumes:

$$M \approx s\,V \approx \sigma\,\Phi. \tag{7.2}$$

This leads to

$$\sigma \approx \frac{s\,V}{\Phi}. \tag{7.3}$$

Typically, s is constant for the whole data set and, thus, can be assumed to be 1. Then, Equation 7.1 is reduced to

$$\sigma \approx \frac{V}{\Phi}. \tag{7.4}$$

With this equation, the density can be directly computed in the data domain. The problem is reduced to determining the volumes V and Φ. The approach is

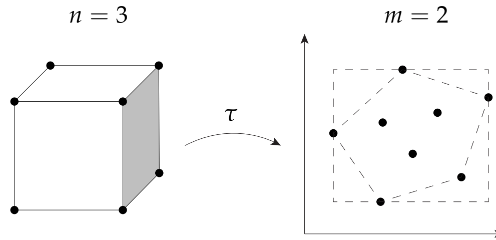


Figure 7.1: Illustration of the approximation of Φ. Projecting a hexahedron from the spatial domain (left) to the data domain (right) results in eight points located in the data domain. The stippled lines indicate the shape that is constructed to represent Φ: an axis-aligned rectangle or the convex hull of the eight points.

to consider a volume V in the spatial domain (i.e., some kind of discretization of 3D space) and apply the transformation τ to the volume V, which yields the volume Φ in the data domain. An illustration of this projection step is shown in Figure 7.1. Please note that a simplified notation is used in which V and Φ denote the actual geometry of a region or its corresponding volume or area, depending on the context.

This approach lends itself to a control of the approximation error. Depending on the extent of Φ in the data domain, a direct measure is available for the maximum error in the data domain (with respect to the error of the projected footprint). If the extent of Φ exceeds a given error threshold, then V needs to be reduced to keep the approximation error within the specified error bounds. Once the approximation error requirement is met, the density contribution of the data values from V can be approximated by calculating the ratio V/Φ. The only remaining issue is to compute the size of V and Φ.

7.2 Algorithm

Two variants of an algorithm were developed that approximate the density in the data domain. Both variants are based on the idea expressed in the previous section: subdivision in the spatial domain controls the approximation error of the density in the data domain. This is done in two steps:

1. the volume V is projected to the data domain,
2. the size of the corresponding volume Φ is estimated.

These two steps are repeated until the extents of Φ are below a user-specified threshold. In contrast to the original continuous scatterplot approach (shown

in Chapter 6.2), both algorithmic variants do not rely on dividing the spatial domain into tetrahedra. Instead, the regular grid of the data set is used to form hexahedra, each of them storing eight multivariate data samples at the corners. As in the original continuous scatterplot approach, the overall density σ in the data domain can be found by linear superposition of all cell contributions. Therefore, this algorithm can construct σ by considering one cell after another.

The two versions of the algorithm differ in the way how the size of the volume Φ in the data domain is computed. The first version uses a convex-hull approach to calculate the volume of Φ accurately, whereas the second version approximates Φ by an axis-aligned rectangle that encompasses the exact shape of Φ.

7.2.1 Subdivision

Following the idea described in Section 7.1, the first step is to project a small volume V to the data domain. The spatial volume V is constructed by creating hexahedra within the regular grid, attaching eight multivariate input data samples to the corners of each hexahedron. Projecting the eight data samples to the data domain results in eight point locations that determine the shape of Φ. Now, the size of Φ is calculated by finding the convex hull of the eight points. The extents of this convex hull can be computed easily.

The subdivision process is triggered when the extent of the convex hull exceeds a user-given limit in the data domain. Possible criteria to measure the extent of Φ include the area of Φ or the maximum extent of Φ in the ξ_1 and ξ_2 dimensions. For this implementation, the latter option is used in order to guarantee that the maximum length of F is bounded. When the subdivision criterion triggers a subdivision step, the current hexahedron is split into eight new hexahedra in a regular fashion. Regular subdivision makes it easy to determine the size of the subdivided volumes V: V covers a relative volume of $2^n 2^n 2^n$ if n is the subdivision level.

For each of the new hexahedra, the attached data values are recomputed using trilinear interpolation. Please note that generic interpolation or reconstruction methods can be applied to find those data values, replacing the trilinear reconstruction filter. The process of projecting the data values of the new hexahedra to the data domain is repeated recursively until the threshold is no longer exceeded. In this case, the resulting convex hull is rendered as a filled polygon with constant density V/Φ, using additive blending. This algorithm is outlined in Figure 7.2 as pseudo code.

An even faster approach to approximate the size of Φ uses an axis-aligned rectangle in the data domain that forms a tight fit around the eight projected points. Since only the lower-leftmost and upper-rightmost points have to be found, the computational effort is reduced when compared to finding the con-

```
// main loop:
for each Cell in data set
{
  Process (Cell);
}

// -----------------------------------------------------------
function Process (IN: CurrentCell)
{
  project CurrentCell to data domain;

  if (Size (ProjectedCell) > threshold)
  {
    // split into eight new cells
    // generic interpolation possible!
    Split CurrentCell;

    for each NewCell do // recursion
      Process (NewCell);
  }
  else // size of Phi small enough
  {
    // draw either convex hull
    // or axis-aligned rectangle
    create triangles for CurrentCell;
  }
}
```

Figure 7.2: Pseudo code for the subdivision approach. For the subdivision step, trilinear interpolation is used. The area Φ can be represented by the convex hull of the projected points or by an axis-aligned rectangle.

```
// precomputation step:
BuildOctree;

// Invoke rendering:
TraverseOctree (OctreeRoot);

// -----------------------------------------------------------
function TraverseOctree (IN: OctreeNode)
{
  if (OctreeNode.SizeOfPhi > threshold)
  {
    if (OctreeNode has children)
    {
      for all children  // recursion
        TraverseOctree (child)
    }
    else // reached a leaf
    {
      // perform subdivison
      Process (OctreeNode.Cell)
    }
  else // size of Phi small enough
  {
    create triangles for OctreeNode.Cell;
  }
}
```

Figure 7.3: Pseudo code that traverses the octree. Once a leaf is reached, the same subdivision is used as in Fig. 7.2.

vex hull and computing its extents. In addition, the rendering is based on simple rectangles instead of more complex filled polygons. Otherwise, this algorithmic variant is identical to the first one.

7.2.2 Octree Hierarchy

Until now, all cells of the original data set are taken into account equally, regardless of their contribution to the final density in the data domain. However, many data sets contain large homogeneous regions. To reduce the amount of cells that have to be considered, an additional octree hierarchy is applied that quickly processes those homogeneous regions in the input data set. Since the computational overhead of the octree should be as small as possible, subdivision is done in a regular fashion only (i.e., cells of the same level in the octree all have the same size). For each node of the octree, the smallest and largest data values of each data dimension are stored. With these values, the extents of the previously mentioned axis-aligned rectangle are known. While traversing the octree, these extents are simply compared with the user-given threshold.

Descending further down along the octree is done as long as the extents exceed the threshold. Once the octree is traversed to the lowest level, each leaf contains a single hexahedron of the original data set. If the size of Φ of that hexahedron still exceeds the threshold, the hexahedron is subdivided as described in Section 7.2.1. For this approach, filled rectangles are rendered with constant density as a representation of Φ. The summary of this algorithm is presented as pseudo code in Figure 7.3.

7.3 Generalization

The basic idea of this approach can be extended to other kinds of grids than regular grids. Alternative grids may lead to modifications of the computation of V or Φ, while the basic idea of subdivision remains. For example, generic structured grids, such as curvilinear grids, have the same topology as regular grids. Therefore, the structure and shape of Φ are not affected by this kind of generalization. Only the computation of V needs to incorporate a more complicated volume formula for deformed hexahedral cells. For other kinds of primitive cells—such as tetrahedra or prisms—the shape of Φ might differ from the hexahedral case. However, the concept of axis-aligned bounding volumes or convex bounding cells in the data domain is not affected. In other words, the main issue with more complex grid structures is not the subdivision process, but to construct an adequate replacement of the octree hierarchy that is put on top of the original data set.

Another kind of generalization replaces the trilinear interpolant. The only required change is that, during subdivision, the more general reconstruction function is invoked to compute the new data values at newly inserted grid points. If the generic reconstruction function is convex, then the convex-hull approach still computes an accurate volume Φ. A function is denoted as convex when the function values stay within the min-max interval of the input values (i.e., the data values at grid points). This is the case for trilinear interpolation and for the example of on-the-fly gradients used for 2D transfer functions (see example in Section 7.5). Even if the generic reconstruction function is not convex, this approach may still provide a good approximation as long as the function values do not reach too far outside the min-max interval of the input values.

7.4 Implementation

The implementation to compute the density in the data domain is written in C++ and executed on the CPU. In order to keep the implementation simple, the code is running single-threaded. The approach that uses the convex hull to compute the size of Φ employs the algorithm described by Jarvis [Jar73].

The octree needed for acceleration of this algorithm is built in a preprocessing step and kept in main memory for all following traversal steps. All generated triangles are rendered with OpenGL using triangle strips. The final result is stored in a floating point texture—by doing this, the continuous scatterplot must be recomputed only if the user changes the viewing parameters. For other user inputs, e.g., brushing areas of interest, the texture is drawn that stores the continuous scatterplot.

7.5 Results

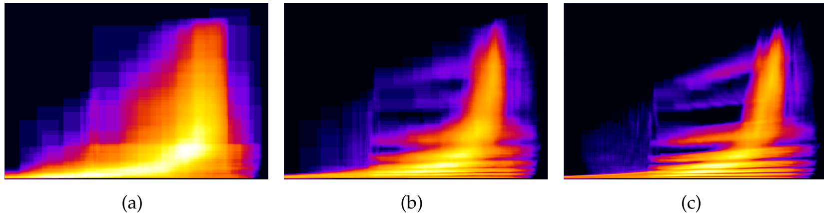
There are two aspects that are of main interest: scatterplot quality and computational speed. The quality of a scatterplot created by the proposed approximation algorithms directly depends on the user-specified threshold. Raising the threshold increases the approximation error, but decreases the time necessary to compute a continuous scatterplot and vice versa. Please note that this threshold is intuitively specified in terms of pixels in the scatterplot—it defines the maximum extents of a projected cell in the data domain.

The scatterplots for the following analysis were created on a personal computer equipped with an Intel CPU with 2.4 GHz and 4 GB of RAM. The computer's GPU is an NVIDIA GeForce 8800 GTX with 768 MB of texture memory. All scatterplot images have a resolution of 1024×768.

The method is first applied to the "Tornado" data set which has dimensions 128^3. This data set contains simulated wind speeds stored as 3D vectors. The horizontal axis shows the magnitude of the velocity, whereas the vertical axis is mapped to the z-dimension of the wind speed to gain the same results as in Section 6.4.1. The resulting images are shown in Figure 7.4. The maximum approximation error is reduced for each image from left to right which results in increased image quality as expected.

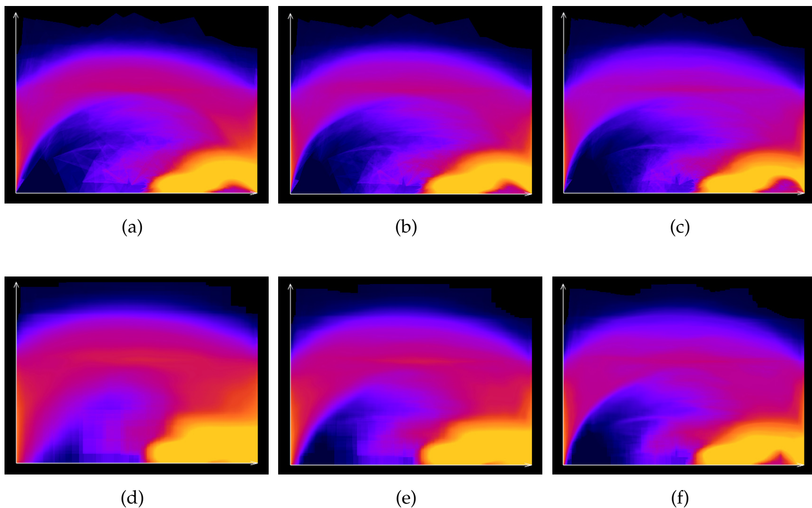
As a second example, the CT scan of a human tooth is analyzed. The spatial extent of this data set is $128 \times 128 \times 160$. The second data dimension is derived as described in Section 1.6. Computing the gradients on-the-fly for the second data dimension is an example of a generic non-linear reconstruction function that can be used with this approach. First, a series of continuous scatterplots is shown which are created with the subdivision approach using a convex hull to represent Φ in the data domain. To examine the effect of the user-specified threshold, the maximum approximation error is decreased step by step.

Figure 7.5 (upper row) shows this series of scatterplots. Despite the significant differences in the threshold, the continuous scatterplots differ only marginally. As it turns out, density values within a cell do not vary strongly, therefore the approximation that uses constant density does not deviate too far from the true values. Differences to the original continuous scatterplot can be explained with the improved interpolation method.



(a) (b) (c)

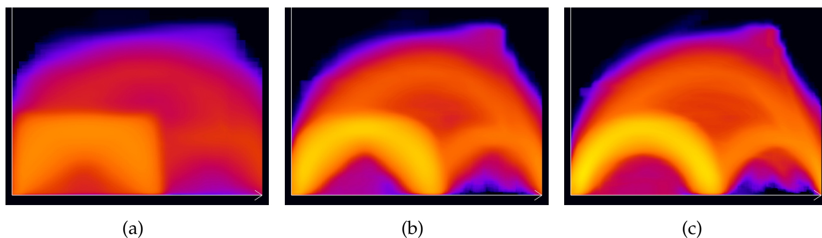
Figure 7.4: These continuous scatterplots were created with the octree approach for the “Tornado” data set. The effect of different thresholds is shown. A coarse approximation is used in (a) with a threshold that allows Φ to extend up to 100 pixels in each dimension, (b) uses a decreased threshold of 50 pixels, and (c) is drawn with a threshold of 25 pixels.



(a) (b) (c)

(d) (e) (f)

Figure 7.5: These continuous scatterplots were created with the subdivision approach for the “Tooth” data set. Pictures (a), (b), and (c) show results of the subdivision approach using a convex hull to represent Φ in the data domain. In (a), Φ is allowed to span up to 200 pixels in each dimension, (b) is created with a threshold of 100 pixels, whereas (c) uses a threshold of 50 pixels. Pictures (d), (e), and (f) show the same data set, but this time the subdivision approach uses axis-aligned rectangles to represent Φ. The same thresholds as for the upper row are applied.



(a) (b) (c)

Figure 7.6: These continuous scatterplots were created with the octree approach for the "Engine" data set. The effect of different thresholds is shown. A coarse approximation is used in (a) with a threshold that allows Φ to extend up to 200 pixels in each dimension, (b) uses a decreased threshold of 100 pixels, and (c) shows a good approximation since the threshold is lowered to 50 pixels.

The same subdivision approach can be used in combination with axis-aligned rectangles to approximate Φ. In Figure 7.5, this series in shown in the lower row. In contrast to the previous example, the effect of the threshold is clearly visible. Using a high threshold and therefore allowing a high approximation error results in a coarse scatterplot. When compared with the subdivision approach that uses the convex hull, this approach yields scatterplots of lower quality since the approximation of Φ tends to deviate much more from the correct solution. On the other hand, scatterplots computed with this approach are created faster, due to simpler computations.

An additional data set was analyzed which was created with a CT scan of an engine block. The spatial extent of this data set is $256 \times 256 \times 110$. As for the first data set, the second data dimension is generated by computing the magnitude of the gradient of the data samples.

With this "Engine" data set, a third series of continuous scatterplots was created. Here, an octree is used in combination with axis-aligned rectangles to approximate the density contribution of a cell in the data domain. The resulting pictures are shown in Figure 7.6. Due to the octree hierarchy, a speed-up of up to two orders of magnitude is achieved compared to the original continuous scatterplot approach. The octree is created in a precomputation step which needs less than one second to prepare for the "Engine" data set. Since memory consumption of the octree is very low, the overhead introduced by the tree structure can be neglected. Depending on the given error threshold, this approach allows one to draw very coarse representations of the scatterplot, since an arbitrary number of cells can be combined in higher levels of the hierarchy. Therefore, this approach is completely independent from the input resolution

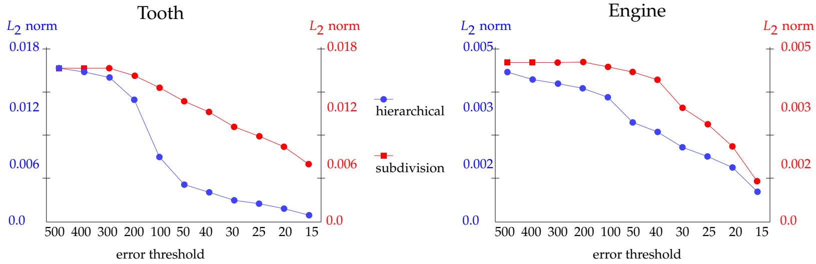


Figure 7.7: The left plot shows the L_2 norm for the "Tooth" data set, the right plot for the "Engine" data set. For both data sets, the hierarchical octree approach ("hierarchical") and the subdivion approach that uses the convex hull ("subdivision") are analyzed. Please note that the scale of x-axis is not uniform. Also, the vertical axes use different scalings for the two different approaches.

of the data set. The computational speed directly depends on the error threshold; however, faster computation leads to lower scatterplot quality.

In order to quantify the effects of the threshold with regard to scatterplot quality, error plots are shown in Figure 7.7. The L_2 norm was used to measure the error between scatterplots. In order to obtain an L_2 norm that can be interpreted easily, the density values of the scatterplots are normed. By doing this, the scale of the scatterplot changes in such a way, that a density value of one corresponds to the arithmetic mean of all density values. Since an analytic solution of a continuous scatterplot is not available, the convergence behavior is analyzed by comparison to a numeric solution with a low error threshold of only 10 pixels. Both error plots for the subdivision approach show a similar behavior: first, the L_2 norm stays on a constant plateau before it drops with decreasing error threshold. High error thresholds do not trigger the subdivision process, therefore the same values are returned by the L_2 norm. For lower error thresholds, the subdivision approach converges in an expected way. Please note that the error values are at a very low level at all times (below 0.2 percent for the "Tooth" data set and 0.1 percent for the "Engine" data set), since the convex hull provides a good approximation. The error plots for the hierarchical octree approach do not have an upper bound as the error plots for the subdivision approach. This is explained with the fact that the octree hierarchy allows very coarse representations for the size of Φ. However, the same behavior as for the subdivision approach can be observed for lower thresholds: the error converges similarly to zero for low error thresholds. However, the absolute scale of the L_2 values is higher due to the coarser approximation by axis-aligned rectangles.

Continuous scatterplots are designed to be included in existing or future inter-

Table 7.1: Time in seconds needed to compute a scatterplot depending on the chosen approach and error threshold. For conventional and continuous scatterplots, there is no threshold that can be set, therefore, only one result is recorded. “Convex Hull” is the subdivision approach using the convex hull to represent Φ. “Octree” is the hierarchical approach that uses axis-aligned rectangles in combination with an octree. Different thresholds were used for the performance measurements; these thresholds are listed in the top row.

Tooth		200	100	50
Conventional Scatterplot	0.04	-	-	-
Continuous Scatterplot	39.8	-	-	-
Convex Hull	-	17.17	17.68	23.88
Octree	-	0.25	2.92	23.19
Engine		**200**	**100**	**50**
Conventional Scatterplot	0.14	-	-	-
Continuous Scatterplot	106	-	-	-
Convex Hull	-	46.1	48	54.5
Octree	-	7.1	22.3	166

active visualization systems. Therefore, their run-time behavior with regard to computation time is of interest as well. Table 7.1 lists time measurements of different methods that compute a continuous scatterplot of the “Tooth” and “Engine” data set. There are some interesting conclusions that can be drawn from these measurements. First of all, the original continuous scatterplot approach is not very user-friendly, since it needs a very long time to compute. This is overcome by using the approaches presented in this chapter. However, the user has to take care that the approximation-error threshold is appropriate. If this is not the case, the presented methods may even need more time to compute a continuous scatterplot than in the original approach. On the other hand, if a good trade-off between accuracy and speed is found, these algorithms compute a continuous scatterplot much faster while the scatterplot is still approximated fairly well.