Abstract Rendering: Out-of-core Rendering for Information Visualization

Joseph A. Cottam[a] and Andrew Lumsdaine[a] and Peter Wang[b]

[a]CREST/Indiana University, Bloomington, IN, USA;
[b]Continuum Analytics, Austin, TX, USA

ABSTRACT

As visualization is applied to larger data sets residing in more diverse hardware environments, visualization frameworks need to adapt. Rendering techniques are currently a major limiter since they tend to be built around central processing with all of the geometric data present. This is not a fundamental requirement of information visualization. This paper presents Abstract Rendering (AR), a technique for eliminating the centralization requirement while preserving some forms of interactivity. AR is based on the observation that pixels are fundamentally bins, and that rendering is essentially a binning process on a lattice of bins. By providing a more flexible binning process, the majority of rendering can be done with the geometric information stored out-of-core. Only the bin representations need to reside in memory. This approach enables: (1) rendering on large datasets without requiring large amounts of working memory, (2) novel and useful control over image composition, (3) a direct means of distributing the rendering task across processes, and (4) high-performance interaction techniques on large datasets. This paper introduces AR in a theoretical context, provides an overview of an implementation, and discusses how it has been applied to large-scale data visualization problems.

Keywords: Rendering, Out-of-core, Binning, Information Visualization

1. INTRODUCTION

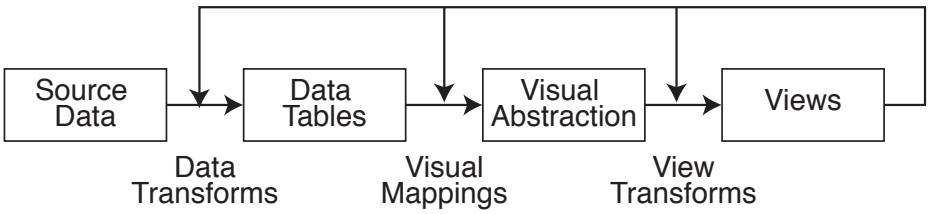
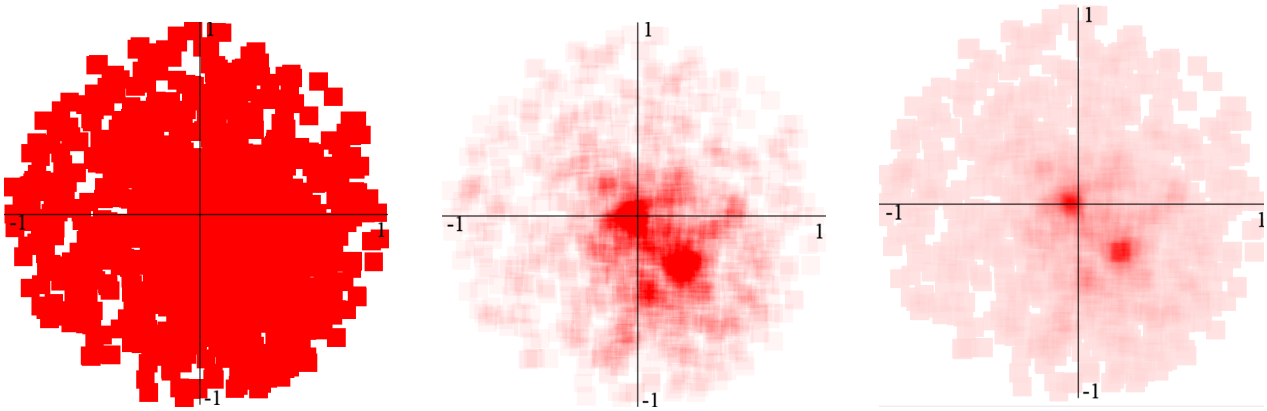


Figure 1. Information visualization reference model.[1]

Visualization transforms source information into a rendered set of pixels. The info-vis reference model (Figure 1) provides a vocabulary for discussing that transformation process.[1] Visualization frameworks tend to focus on the "visual mappings" stage. In this stage raw data is projected into a set of geometrical abstractions, and graphics are represented with high precision on a logical canvas. Conversion to actual pixels is given significantly less attention. Many frameworks simply offload the view transforms and related rasterization to external graphics libraries (such as an SVG renderer, OpenGL or Java2D). Efficient transfer of data to the engine or the raw speed of the underlying engine are often the only consideration discussed. Abstract Rendering (AR) expands control over what occurs in the "view transform" stage of the info-vis reference model, enabling direct discussion of render-time effects pertinent to visualization construction in a simplified manner.

Further author information:
J.A.C.: jcottam@indiana.edu
A.L.: lums@indiana.edu
P.W.: pwangg@continuum.io



(a) Over-plot (b) Standard Alpha (c) High-Def Alpha

Figure 2. A scatter plot of synthetic data under various treatments.

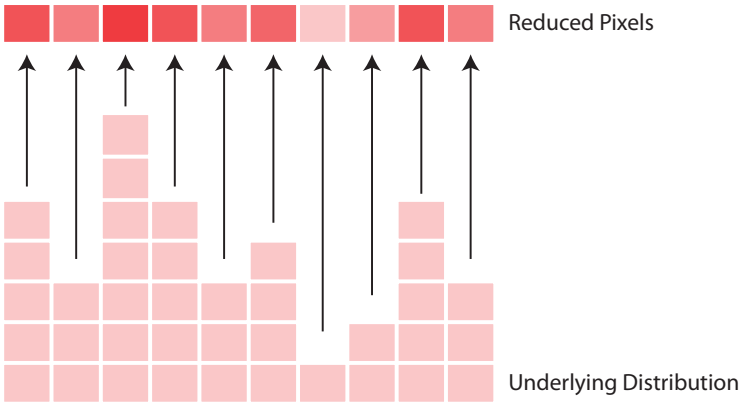


Figure 3. One-dimensional distribution and reduction.

Consider a scatter plot, such as the one shown in Figure 2. Each of the 1,500 points is a 10-pixel-wide square, resulting in 150,000 pixels minimum to show everything. At 200dpi and 2 inches, even a highly regular tiling can barely show all of the items without overlap. However, most data is not very regular, and over-plotting naively leads to data loss. That data loss is directly encountered when no attempt is made to mediate it and over-plotting is accepted (Figure 2a). When over-plotting is present, the rendered plot conveys presence/absence distribution information, which is a bound on the actual distribution and not a full description. This is not necessarily undesirable, but it is a silent transition of plot type that is not represented in the visual mappings. *The issue of silent transitions is compounded by the fact that the visual mappings are often presented as the definition of a visualization* .[2–6] A common approach to mediating data loss due to over-plotting is to use alpha-compositing, providing more complete distribution information on a per-pixel basis as an emergent effect (Figure 2b). Alpha-compositing is used as an implicit and emergent means of transitioning from a standard 2D scatter plot to a heat-map style plot. It is an implicit request to perform a reduction on the glyphs being rendered on a per-pixel basis (see Figure 3). Alpha-compositing is usually indicated by setting the alpha channel during visual mappings. However, the composition algorithm is usually deferred to the rendering system (e.g., OpenGL, Java2D, etc.). Rendering systems have their own limitations that directly influence the display of a plot and its interpretation. For example, most hardware limits alpha composition to a maximum of 255 items per pixel (using common 32-bit color encodings and the "over" rule from Porter and Duff[7]). This maximum overlay can be achieved only if the minimum non-transparent alpha setting is used; the actual maximum is proportionally reduced when more common alpha-values are employed. For example, in Figure 2b, alpha is set to 10%, so only 25 items can be composed before a pixel is saturated with 32-bit color encoding. Though the image in Figure 2b is more detailed than the one in Figure 2a, it hides a great deal of the structure due to the limits of standard

alpha composition. Gaining explicit control over the plot-type transitions and related render-time effects is the goal of the AR framework. Figure 2c provides a high-definition alpha composition[8] and reveals more subtle structures in the over-plotted region. Control over alpha composition, over-plotting and many more render-time regimes can be efficiently realized as render-time effects when supported by the conceptual abstractions of AR.

Succinctly, AR is constructing a synthetic data space informed by both the geometric constructions and raw data. It logically proceeds in two phases. The first phase is the creation of a synthetic data space that conceptually sits between between the geometric representation and the raw pixels of an image. The second phase is application of transformations to that synthetic space to create an actual image.

This first phase is analogous to traditional rendering but no restriction is placed on the topology or content of the rendering (where traditional rendering produces a regular rectangular grid topology of colors). To be effective, the synthetic space must provide new tools to analysis that neither the raw data, pixels, or geometric constructions do by themselves. The ability to compute the full range of overlap, as seen in high-definition alpha composition (see Figure 2), is one such tool.

The fundamental observation of AR is that individual pixels are bins that represent raw data by way of a synthetic space. The synthetic data space is built out of (but not identical to) the geometric representation. AR materializes this synthetic space and provides tools for working with that space. It is defined in terms of four functions (discussed in detail in Section 3). The system is named AR because it computes a discretized result, much like standard rendering, but the discrete values do not need to be colors (and the process is thus more abstract than traditional rendering).

2. RELATED WORK

Visualization is most often concerned with pixels in two ways. First, are visualization techniques that are specifically targeted at pixel-level details (so-called "pixel-oriented techniques"). Pixel-oriented techniques try to maximize data preservation by considering individual pixels in the display. Often pixels are placed in 1:1 correspondence with data items. Common features include space-filling layouts and distortions oriented at packing and spreading data so over-plotting does not occur.[9]

The second way visualizations often deal with pixel-level effects is emergent effects, as occur with alpha composition. Standard alpha composition has known limitations for handling large amounts of over-plotting. Many alpha channels are limited to eight bits, capping over-plot at 256 items if alpha is set to the minimum visible level. In most circumstances, such low alpha values are unacceptable, since areas with few values would be practically indistinguishable from the background. Johansson, et al.[10] describes a multi-stage technique for high-definition alpha compositing. Muelder, et al.[8] employs this technique while visualizing MPI communication behavior.

High-definition alpha composition invokes transfer functions more common of scientific visualization.[11] A traditional transfer function takes a 3D information space, discretized into a set of 'voxels', and maps each voxel to a color. This mapping can take into consideration information beyond the properties of the immediate voxel including neighboring voxel values and viewing angle. The application of transfer function in Abstract Rendering follows the pattern of earlier work,[8,10] operating on pixels instead of voxels. A high-precision pixel space is created (either explicitly as an off-screen buffer or implicitly by functional processes on a scene-graph) for the source data to the transfer function. The transfer function processes this high-precision space in the context of the view transformation to determine the actual rendered pixels. This process may include reference to neighboring pixels, the view transformation, or underlying source data. The transfer function provides opportunities to perform render-time optimizations such as ensuring visibility of particularly important features and emphasizing specified range in a statistical distribution.

Some techniques do consider pixel-level issues in ways distinct to the two main methods described above (explicit pixel-oriented techniques and emergent controls). Color-weaving[12,13] is a recent technique in this category. Color-weaving presents each relevant color in part of each region it belongs in, rather than blend each color in a pixel. The colors in the region can be made proportional to the underlying data, similar to how opacity scaling enables proportional contribution at the pixel level. Though developed for applying multiple factors to a single region, the concept can be extended to emergent regions formed by overlapping polyhedra. Semantic

zoom techniques change the visual mapping of information based on the screen space allotted to them.[14] This is in contrast to standard zoom where only the view's affine transform is modified as screen space is reapportioned. Abstract Rendering can implement semantic zoom by modifying the renderer depending on screen space.[15]

Blending geometric and data components into a rendering system is part of most coordinated multiple views systems (CMV) where visual linkages are determined by underlying data relationships.[16] These relationships typically override the regular standard visual representations for selected items. It is also similar to Data Shaders[17] and many GPU techniques that use color buffers as data buffers.[18]

Parallelizing rendering is an extensively explored topic. Piringer, et. al[19] describes task-parallel rendering based on distinct layers of data. Cottam and Lumsdaine[20] discusses task- and data-parallel rendering techniques for dynamic data. OpenGL[21] is essentially a framework controlling data-parallel analysis related to rendering; the OpenCL[22] framework extends that idea to more general data processing. The framework described in this paper complements much of this research. For example, Abstract Rendering can provide a per-layer and cross-layer composition framework for layer-based techniques. It also provides a structure for exploiting parallelism implicit in single-layer techniques (such as color-weaving and semantic zoom).

3. ABSTRACT RENDERING

Abstract rendering is done by chaining together four different function types. The four function types are (1) select, (2) info, (3) aggregate, and (4) transfer. The first three functions are used to create a synthetic data space. This synthetic data space is derived from the source data and the geometric data. The select function picks geometry, the info function produces a data value associated with a geometric item, and the aggregate function combines info values together. An effective synthetic data space enables efficient analysis, performed by the transfer function. These four functions will be described in greater detail in turn through the evolution of an example visualization.

The example dataset comes from a social network analysis of the open source community surrounding Sourceforge.net circa 2007. Each project is a node in the network and each developer is a link. The largest connected component (27,500 projects) was laid out using VxOrd.[23] This layout was used to visually assess the interactions between the social network and various project attributes as part of an exploratory analysis.[24]

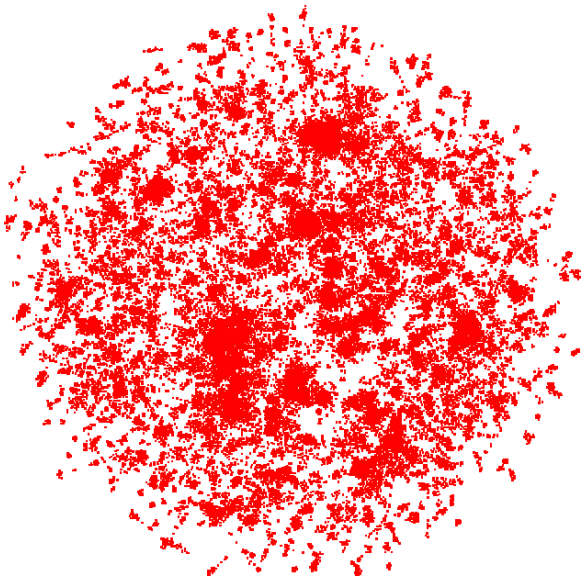
Figure 4 shows various renderings of just the project nodes, encoded as sub-pixel-sized squares. The naive treatment of 100% opaque squares (Figure 4a) demonstrates the rough distribution of the clusters in the data. However, it does not show density inside of the clusters. Figure 4b shows a simple alpha-treatment (alpha set to 1%). In this minimal treatment, the highest peaks are still over-saturated and the lowest troughs are indistinguishable from empty. Shifting to 25% alpha on each node yields improved definition in the low-population areas, but at the cost of significant over-saturation in the highest peaks. AR enables an accurate treatment of this data and provides diagnostic tools to guide decisions.

3.1 Simple Example: Select, Aggregate, Transfer

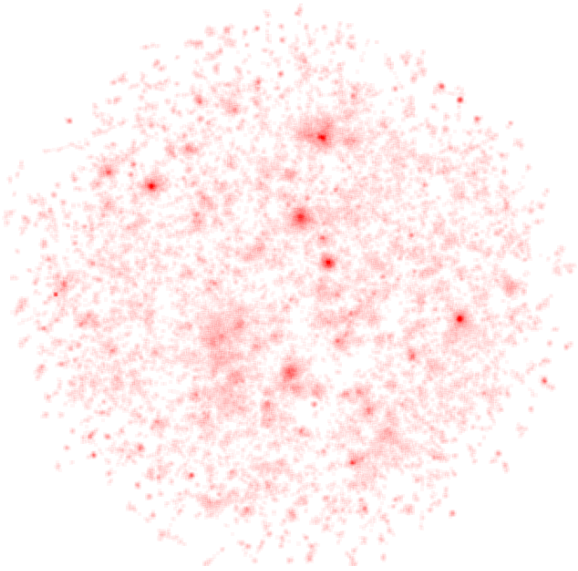
As mentioned, the original dataset is being represented as colored squares. Abstract rendering accepts these squares as input. The basic task for an accurate transparency-based representation of the data is to count how many items land in each pixel, then to create a color ramp that handles the entire range of counts. This pixel-level analysis and scale creation is known as High-Definition Alpha Composition (HDAlpha).[8]

For HDAlpha, the synthetic data space is a pixel-level grid of the number of items that land on each pixel. This grid of counts is directly constructed by (1) selecting all the items that land on a pixel, and (2) counting the number of items that are selected. In terms of the four AR functions, the selector is the "intersects" function, which selects all of the items that intersect a given pixel. The aggregator is the "length" function that counts the number of items in a list. (The "info" function will be addressed later.) Applying $length(intersects(Glyphs, pixel))$ for each pixel in the image will result in the required synthetic data space. (Assuming "Glyphs" is a list off all of the source rectangles, "pixel" is a rectangle representing a single screen pixel projected into the logical canvas that the glyphs occupy.)

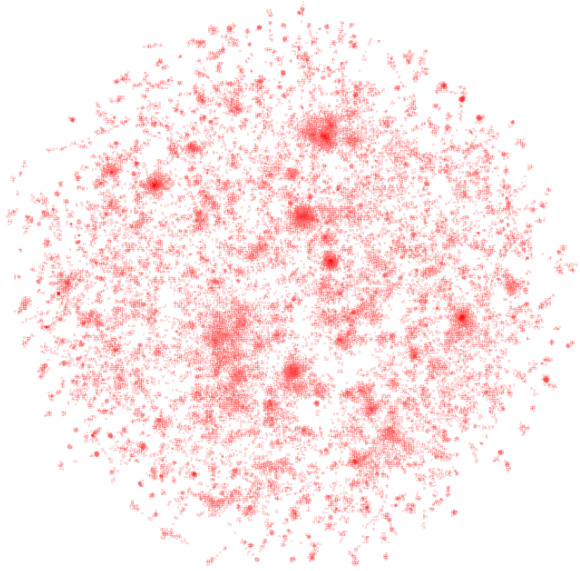
The synthetic data space can be directly analyzed to find the minimum and maximum intersection values. A color ramp can be built between those two values and directly applied to the synthetic data space. Building and



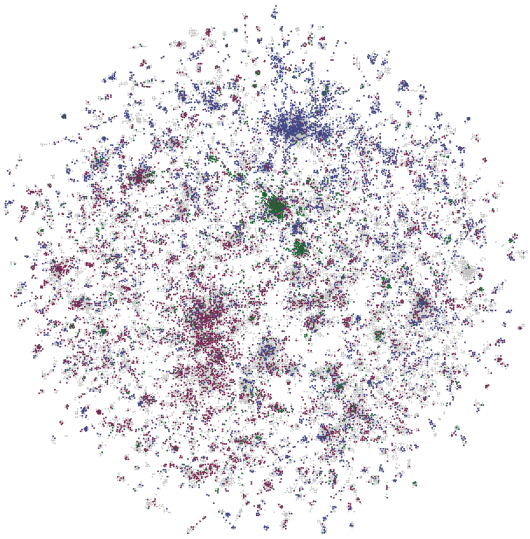
(a) Full Saturation Nodes



(b) 10% Alpha Per-Node



(c) High-Definition Alpha



(d) Category Integrated

Figure 4. Various treatments of the social network of software development.

applying this color ramp is the task of "transfer," the fourth AR function. $interpolate(s, Red.10, Red, min(S), max(S))$ therefore supplies the color for each pixel ("Red.10" is red with 10% alpha, S is the synthetic data space and s is an item in S). Applying AR yields the image shown in Figure 4c. This avoids over-saturation in the highest peaks, while guaranteeing visibility of the lowest troughs (shown at 10% saturation). The result is a correct image of the distribution of the nodes of the social network.

3.2 The Info Function

In the preceding example, the "info" function was not specified because it was not required. The count of the values was the point of concern. Trivially, the same encoding can be created by using an info function of "identity" that returns whatever it is passed. Using "identity" for the info function, the synthetic space creation is done by $length(identity(g)|g \in intersects(Glyphs, pixel))$. Recalling the original problem, each project has various attributes associated with it. The "info" function slot enables a more detailed representation that includes those attributes. Assuming the attribute encodes the programing language and has values "Python", "C/C++", "Java" and "Other", a more detailed synthetic data space can be made with $countCategories(att(g)|g \in intersects(Glyphs, pixel))$. In this phrasing, "att" returns the attribute value and "countCategories" creates a list of the unique values seen paired with how often they are seen. Applying a category-aware transfer function, the image in Figure 4d results.

3.3 Formalization

Abstract Rendering can be compactly described as the application of four functions, combined in the following fashion:

$$s_{xy} = Aggregate(\{Info(g)|g \in Select(G, P_{xy})\})$$
$$c_{xy} = Transfer(s_{xy})$$

In this formulation all (x,y) values refer to positions on the screen and must match up between the two equations to color a single image pixel. Furthermore, G represents all glyphs with $g \in G$, $P_{x,y}$ is a geometric representation of a pixel, s_{xy} represents a value in the synthetic data space and $c_{x,y}$ is a final pixel color. Each of the component functions may require additional arguments (such as "interpolate" needing the high color, low color, min value and max value). When required in further discussion, these will be included in line. In general, the first of these equations will be referred to as the synthesis step, while the second will be called the transfer step.

The final Sourceforge image as seen in Figure 4d is produced with the following abstract rendering equations:

$$s_{xy} = countCategories(att(g)|g \in intersects(Glyphs, pixel))$$
$$s1_{xy} = ReKey(s_{xy}, \{Python : Green, C/C++ : Red, Java : Blue, Other : Gray\})$$
$$c2_{xy} = HDAlpha(s1_{xy}, S1)$$

where $ReKey$ replaces the keys of a dictionary with the keys found in a new dictionary. In this case, the old dictionary was produced by "countCategories" and included programming languages as keys. $ReKey$ uses its second argument to associate the counts with colors instead of languages. $HDAlpha$ expects a set of colors and quantities for each pixel, implementing the procedure described earlier.[8] When more than one transfer function is present, they are executed sequentially.

The division between the synthesis and transfer is not essential for expressiveness. For example, conversion to actual colors can be done by the aggregate function. However, a separate transfer function provides two important benefits. First, the division reflects a conversion of data types. The synthesis step converts geometry into the discretized synthetic data space. In contrast, the transfer step starts and ends with a discretized space. The second reason for separating the reduction and transfer function is that the division provides many optimization opportunities. These optimizations are discussed further in Section 4, but in brief, the division provides (1) a natural break between client and server execution, (2) a natural break between source-data aware and oblivious operations (reducing the working set), (3) an expectation that neighboring aggregate values have already been calculated when performing transfer (without which, AR definitions were often recursive and required greater

care to implement without infinite loops), and (4) the prior three combine to make efficient execution strategies for both reduction and transfer more direct. The difference between the aggregates and the eventual pixels is similar in magnitude and benefit to the difference between Data Tables and Visual Abstractions in the info-vis reference model. Theoretically, "Data Tables" in the standard model could be directly transformed into pixels. However, the "Abstract Geometry" of visual abstractions is useful for conceptual and practical purposes.[1,15]

Conventions and Details

To simplify further discussion, a few auxiliary items need to be defined. These definitions are not implementation requirements, but will focus discussion to the essential points. Many operationally equivalent options exist. When discussing sets, all sets will be of homogeneous type. Types under consideration are numbers, colors, glyphs, and tuples. Glyphs will be treated as a geometric shape and a fill color. Tuples will be modeled as sorted dictionaries where all fields must be assigned. In this way, the fields of a tuple have both a stable name and index for all tuples of the same type. Practically speaking, if a value of type $tuple(c : color, Z : int)$ is described, it is a pair with fields 'c' of type *color* and 'Z' of type *int*. All tuples of that type will have color as the first field and Z as the second. The flexibility of either name or position-based reference makes some functions easier to describe. By convention, functions that return more than one value will return tuples. Glyphs and colors will be treated as tuples with convenient fields. With these data types defined, the function types are as follows: Select functions will return sets of glyphs; Info functions will take glyphs and return tuples; Aggregate functions will operate on sets of tuples; and Transfer functions will take and return tuples. Details on specific functions will be provided at their site of use.

4. IMPLEMENTATION

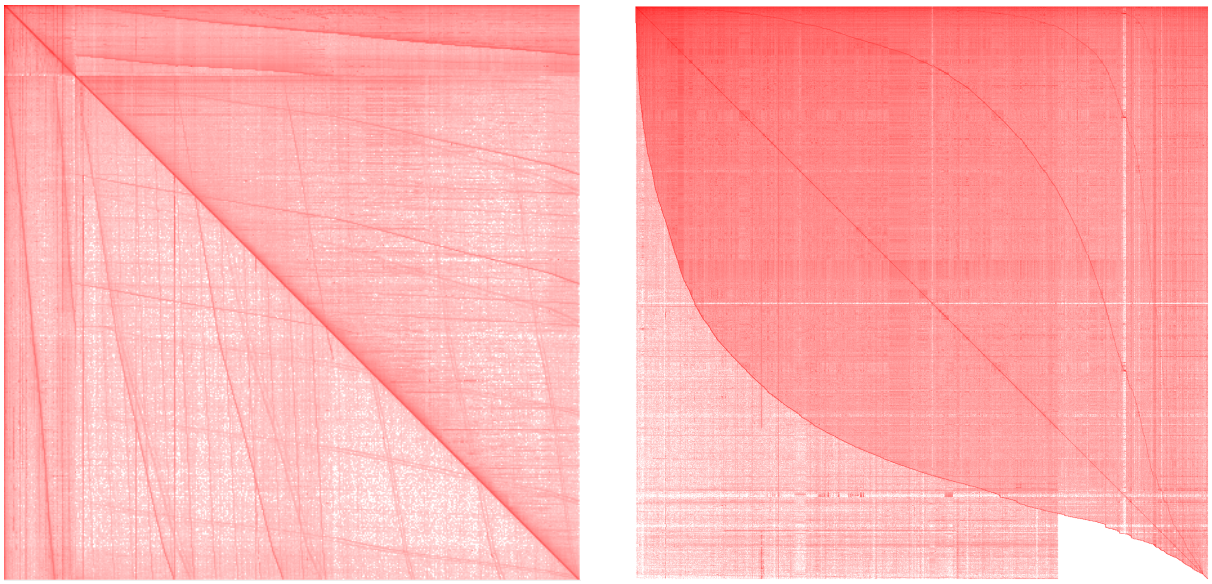
Abstract rendering has been implemented in Java and Python. This section describes the implementation in Java. The Python implementation is similarly structured, but differs in the parallelization strategy (relying heavily on vectorization through NumPy[25]).

The Java implementation generally follows the definition provided in Section 3. All four function classes are directly represented as interfaces. However, aggregate functions are expressed at a more fine-grained level and have a few restrictions to provide efficient execution in parallel and out-of-core environments. In addition to the standard functions, the Java implementation introduces a "Renderer" class to control execution and data access order. The most efficient execution strategy encountered thus far is glyph-oriented during aggregation and pixel-oriented during transfer. The pixel-oriented transfer executes exactly as described in Section 3. It can be efficiently parallelized by dividing the aggregate values into contiguous blocks.

The glyph-oriented aggregation strategy diverges from the execution implied by Section 3, but provides significant opportunities for optimization and execution in out-of-core environments. In the glyph-oriented execution, each glyph is visited exactly once, and each aggregate value may be updated multiple times. This allows source data (often large relative to the number of aggregation bins/pixels) to be visited in whatever order is efficient for its containing data structure. However, since the pixels may be updated multiple times, the aggregate function never receives a list of info function results (as described in Section 3). It instead receives one info result and a pre-existing aggregate value. The implications of this different iteration behavior are that the aggregate function must have a "zero" value and must be commutative-associative if deterministic behavior is desired. The zero is an indicator that the synthetic space location is empty and should behave like an identity in that $op(x, zero) = x$ for all x. The zero value is used to initialize the synthetic data space and often simplifies the coding of the actual operator by allowing to assume both arguments are always valid. In the pixel-oriented description of AR in Section 3, the aggregate operator receives a list of all relevant info results at the same time. Therefore, the operator does not need to be commutative-associative. This glyph-oriented implementation receives info results as a temporally sequence, and thus needs to handle that sequence in some way. Requiring aggregate functions to be commutative-associative is a solution that has not restricted actual use.

When presented with a list of info results, the operator receives all of its operands "at once" and is thus does not need to be commutative associative.

This implementation architecture provides for efficient rendering for a wide variety of data sizes and enables mixing of serial, task-parallel, data parallel and vectorized execution strategies. The parallel Java renderer uses



(a) Kiva (b) Wikipedia

Figure 5. Adjacency matrices used in performance analysis.

$$s_{xy} = length(identity(g)|g \in intersects(Glyphs, Pixel))$$
$$c_{xy} = interpolate(log(s), Red.10, Red, min(S), max(S))$$

Figure 6. AR equations for the test images. Interpolate performs linear interpolation between the first two parameters, with the maximum and minimum values provided by the last two parameters and the actual value by the third.

the fork/join framework for both aggregation and transfer execution. The modified aggregate function enables blocks of input data to be processed independently. The resulting aggregate results of each block are combined to form the final result using an additional "rollup" function on the aggregator class.

In addition to parallelization, this architecture supports efficient out-of-core data processing. Just as partial results can be combined between threads, the "rollup" function on the aggregator can be used to update partial results with new input values. This type of streaming update is extremely difficult with the pixel-oriented phrasing of AR from Section 3, but straightforward using the glyph-oriented implementation described here.

5. DATASETS

The SourceForge.net visualizations used in Section 3 were used in social network analysis. The original analysis predates AR, but AR enabled a more direct resolution of issues encountered in that original analysis. The dataset only includes a few thousand data points, and thus dosen't directly require the out-of-core capabilities. However, the ability to interactively change the transfer function was beneficial in tuning representations without needing to refer back to the source data (only the aggregate values). This interactive style was developed for out-of-core processing to provide interactivity at a level that does not require re-iteration of the raw data. The conclusion of the network analysis was that the emergent structure of the open source software community was aligned to the technical structure of the solutions, *not* the problem space. This is evident in the presence of clusters based around programming languages, not problem domains. This observation is elaborated and backed up in other work.[24]

To characterize performance, we use adjacency matrices derived from two datasets, show in Figure 5. These datasets were used because the data volume is sufficient to require out-of-core processing but require simple analysis to create a visual representation.

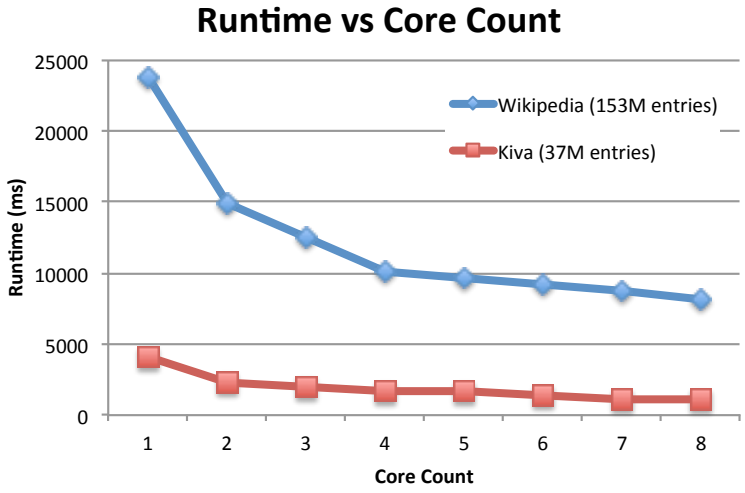


Figure 7. Scaling behavior of AR as processor core count and task count increase.

The main dataset is a collection of 37 million transactions from the Kiva micro-finance site. Each transaction represents a monetary transfer between a lender, borrower, or intermediary. Each actor was given an identifier, with senders placed on the x-axis and receivers placed on the y-axis. Each transaction is represented as a point at the intersection of the sender and receiver. The coloring was done by log-transforming the counts of items contained in each pixel. Figure 6 has the formal phrasing of AR treatment of the data.

The Kiva dataset was part of a data analysis "sprint" under DARPA's XDATA program. Prior visualizations of the dataset relied on off-line pre-processing of the data and provided interactive graphics through a tile-server. AR's out-of-core processing enabled comparable visualizations to be produced in an interactive fashion on simpler hardware. The density-based visualization in Figure 5a shows the resulting structure. Prominent features are the diagonal, representing transactions to self, which is how Kiva records payment of transaction fees for its intermediaries. The spine-like shape is also indicative of a generally depth-first treatment of the data,[26] in this case partially based on temporal ordering. (To be fair, the tile-server provides more responsive zoom and pan navigation, a problem discussed in Section 7.)

The second dataset is an adjacency-list of links found on Wikipedia. This data set includes 153 million edges, representing the links of the largest connected cluster if the category system is ignored. Links were numbered by topological sort ordering. Because links are directed, the resulting adjacency matrix is not symmetrical. The AR encoding is the same as that for the Kiva data set.

The Wikipedia dataset was brought to this team to help triage invalid results produced from an analysis pipeline. The topological-sort order of the data was suspect, but prior attempts to visualize the ordering had failed. For example, an R-based approach was abandoned after it executed for more than 12 hours without results. The high memory overhead of retained-mode graphics frameworks lead impractical memory utilization in another attempt. AR's out-of-core capabilities directly address memory consumption, enabling the data to stream off of disk. The resulting image immediately revealed at least one problem with the analysis the team was pursing. They had assumed the topological sort ordering was based on undirected edges, but the asymmetry and presence of points outside of the "envelope" invalidated that assumption.[26] AR's ability to scale by out-of-core processing made this result accessible on a notebook-class computer, where more powerful desktop-class machines had failed with other frameworks.

6. OUT-OF-CORE RENDERING PERFORMANCE

To characterize performance, we use adjacency matrices derived from two datasets: Kiva and Wikipedia. These two datasets were described in Section 5, along with the actual test visualizations (Figure 5). For the tests

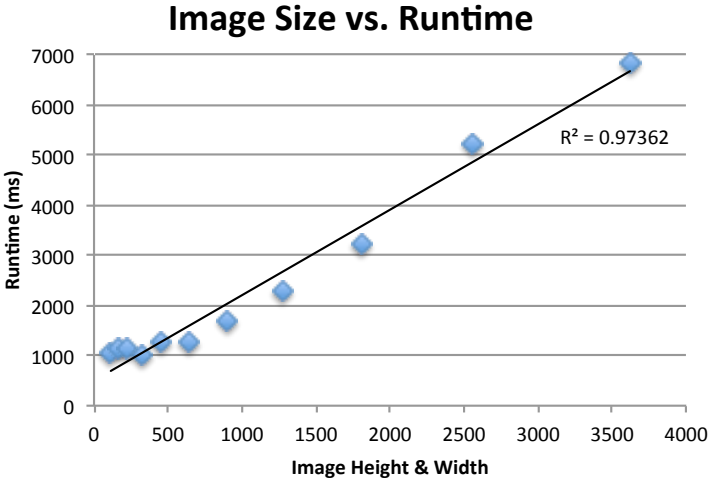


Figure 8. AR scales linearly as image size increases.

presented in this Section, each data set was binary-encoded as an adjacency-lists and stored in a file. The file contents were memory mapped, streamed off disk and rendered in a single pass over the data. The Kiva dataset contained 37 million edges, and the Wikipedia dataset has 153 million.

The test machine had eight physical cores with hyper-threading to provide sixteen logical processors. The installed ram was 144 GB, though the Java Virtual Machine was allocated a maximum and initial heap space of 1GB. (Due to memory mapping, the total process memory could exceeded this amount.) The machine ran Linux (CentOS 2.6) and OpenJDK Java (1.7.0). Prior experience showed that two tasks per core often outperformed one task per core, but otherwise no specific tuning was done for this machine. The Linux "taskset" command was used to limit the number of processors used by specific runs and the "parallelism" level of the fork/join pool was set to match that core count. Each configuration was tested 10 times, and conclusions here are based on the average execution time.

Figure 7 presents the performance results. In general, more physical processors are more helpful. On average, an additional core provided about 12% speedup, though the first core provided closer to 40%. Hyper-threading (not shown) also improved performance by a small amount, but including all eight hyper-thread cores only improved runtime by 10%. Execution times for the Wikipedia data set were recorded under the same circumstances as the Kiva data set. These results can also be seen in Figure 7. The scaling characteristic is similar to that of the Kiva dataset. The Wikipedia data set is four times larger than the Kiva dataset, and on average took five times longer to render with matching test configurations. This source of this additional time is not known but may be related to the data layout or percentage of the pixels containing non-zero values.

The variance for each configuration was less than 6% of the mean for all configurations. In all configurations, the first execution was the slowest. Subsequent executions were typically 25% faster than the first execution, and the third execution 15% faster than the second. Executions 3-10 varied by less than 4% on average. These differences likely reflect additional just-in-time compilation and improved caching of the memory mapped file. Using a particular iteration instead of the average (i.e., first, worst or last) does not change the trends or percentage variations by much, but does change the absolute values involved. All test images were 800 by 800 pixels.

Switching from spinning media to a solid-state-drive does not significantly impact the runtime (first execution time in each configuration improved by an average of 5%, subsequent executions by less than 1%). This leads us to conclude that the execution performance is not latency limited, though it may be bandwidth or CPU limited.

Using VisualVM, a standard Java instrumentation package, total heap space was monitored during the execution. For both data sets, total allocated heap space never exceeded 700 MB. This is despite the fact that on

Figure 9. Representation of sparse features in the Kiva adjacency matrix.

disk the Kiva data set occupies 1.6 GB and the Wikipedia data set occupies 2.9 GB. Forcing garbage collection after execution showed that than neither data set needed more than 100 MB to store the aggregate set, result image and test application. Further investigation showed that AR could complete either visualization with a heap size of 150 MB by using a larger set of smaller tasks. Using more, smaller tasks significantly increased runtime. This illustrates that the memory requirements for AR are driven principally by the task size, and not by the overall data size; though the resulting image size has some impact as well. Execution with larger maximum heap space yielded a small decrease in execution time, probably driven by less garbage collection activity.

In our tests, the granularity of the synthetic data space matches that of the screen size. So, image size influences performance significantly. Figure 8 shows the influence of image size on performance using the Kiva dataset and four cores of the test machine. To accommodate the larger image sizes, the heap-space was increased to 2GB. The results show that render speed scales linearly with the number of pixels. The actual scaling factor (e.g., slope of the trend line) varied by how many pixels were occupied by data, but all data sets tested scaled linearly.

7. FUTURE WORK & CONCLUSIONS

The architecture described in Section 4 satisfies many of the requirements for operating in a distributed memory environment. Notably, the interaction style of changing transfer functions can provide a limited form of interactivity without requiring additional client/server communication. However, work on client/server protocols is required. Additional work and distributed memory environments is a high priority for AR.

Current work on AR focuses on controlling the representation of data. AR provides opportunities to automatically evaluate visualizations by taking into account both data and visual properties. Initial forays into this field have included automatic detection of over- and under-plotting, and detection/display of features that combine both presence and absence of information. Figure 9 represents visualization in this last category. The experimental transfer function used for this figure presents the *ratio* of values in a neighborhood. Darker pixels

are higher ratios. This is in contrast to Figure 5a, which is based on absolute values. This new treatment provides greater visual saliency to features such as the sparse rows on the right-hand-size and the asymmetry between the upper and lower triangles. Producing this image and AR requires only changing the transfer function. It is one example of novel techniques that are directly accessible in AR, but not directly related to out-of-core opportunities. This avenue of research is closely related to scagnostics,[27] but provides promising new opportunities based on the integration of data and visual spaces.

Current work on parallelization in AR focused on the aggregation step. This is largely due to the simplicity (and related speed) of the current library of transfer functions. However, more complex transfer functions are being explored. These include ISO contours and aggregates-based analysis of a visualization. With these more complex functions, parallelization will be more important. The general structure of the transfer function matches GPU pixel shaders, which will be investigated as the need arises.

A significant decision in current AR implementations and descriptions are to closely tie the bin-elements to the screen resolution and display region. This decision introduces view-dependent effects where there were not before. View-dependent effects are used advantageously in HDAlpha composition, but may not always be desirable. Developing techniques for avoiding these effects and guidelines for their usage is an ongoing effort.

Relatedly, a significant limitation of AR is the inclusion of the selection function in the synthesis step. The net result is that operations such as zoom and pan include recalculating the (relatively expensive) synthesis step because selection associates source data with pixels. Preliminary work has shown that some time/space tradeoffs can be made can that significantly improve the performance. Brushing/linking, dynamic queries, and picking are similarly not supported in a way that is both general-purpose and efficient, though they may be expressed through AR by applying appropriate aggregation and transfer functions.

Current work on AR focuses on pixel-oriented tessellations. Work on non-rectangular tessellations has been pursued by systems such as imMens.[28] Additionally non-tessellation based binning strategies may lead to novel visualization metrics.

This paper is focused on bin-based strategies for large data visualization via out-of-core techniques. Some of these techniques may be equally applicable to more traditional data sizes and yield new capabilities for rendering in more traditional applications. However, it is likely that different aggregation and transfer functions will be of higher utility.

Abstract Rendering presents a powerful extension to the common rendering paradigm. It provides techniques for controlling traditionally emergent properties of the visualization. The same abstractions also open up novel opportunities for efficient implementations of rendering in nontraditional execution environments. This paper demonstrated that AR can be efficiently implemented, and applied to out-of-core rendering problems with large data sets.

REFERENCES

[1] Card, S. K., Mackinlay, J., and Shneiderman, B., [*Readings in Information Visualization: Using Vision to Think*], Morgan Kaufman (1999).

[2] Bostock, M. and Heer, J., "Protovis: A graphical toolkit for visualization," *IEEE Transactions on Visualization and Computer Graphics* **15**(6), 1121–1128 (2009).

[3] Bostock, M., Ogievetsky, V., and Heer, J., "D3: Data-Driven Documents," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011).

[4] Cottam, J. A., *Design and Implementation of a Stream-Based Visualization Language*, PhD thesis, Indiana University (2011).

[5] Wilkinson, L., [*The Grammar of Graphics*], Springer-Verlag, New York, 2nd ed. (2005).

[6] Wickham, H., "A layered grammar of graphics," *Journal of Computational and Graphical Statistics* **19**, 3–28 (March 2010).

[7] Porter, T. and Duff, T., "Compositing digital images," *SIGGRAPH Comput. Graph.* **18**, 253–259 (Jan. 1984).

[8] Muelder, C., Gygi, F., and Ma, K.-L., "Visual analysis of inter-process communication for large-scale parallel computing," *IEEE Transactions on Visualization and Computer Graphics* **15**, 1129–1136 (Nov. 2009).

[9] Keim, D. A., "Designing pixel-oriented visualization techniques: Theory and applications," *IEEE Transactions on Visualization and Computer Graphics* **6**, 59–78 (Jan. 2000).

[10] Johansson, J., Ljung, P., Jern, M., and Cooper, M., "Revealing structure within clustered parallel coordinates displays," in [*Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*], *INFOVIS '05*, 17–, IEEE Computer Society, Washington, DC, USA (2005).

[11] Fang, S., Biddlecome, T., and Tuceryan, M., "Image-based transfer function design for data exploration in volume visualization," in [*Proceedings of the conference on Visualization '98*], *VIS '98*, 319–326, IEEE Computer Society Press, Los Alamitos, CA, USA (1998).

[12] Urness, T., Interrante, V., Marusic, I., Longmire, E., and Ganapathisubramani, B., "Effectively visualizing multi-valued flow data using color and texture," in [*Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*], *VIS '03*, 16–, IEEE Computer Society, Washington, DC, USA (2003).

[13] Hagh-Shenas, H., Interrante, V., Healey, C., and Kim, S., "Weaving versus blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color," in [*Proceedings of the 3rd symposium on Applied perception in graphics and visualization*], *APGV '06*, 164–164, ACM, New York, NY, USA (2006).

[14] Bederson, B. B., Grosjean, J., and Meyer, J., "Toolkit design for interactive structures and graphics," *IEEE Transactions on Software Engineering* **30**(8), 535–546 (2004).

[15] Heer, J. and Agrawala, M., "Software design patterns for information visualization.," *IEEE transactions on visualization and computer graphics* **12**(5), 853–60 (2006).

[16] North, C., "Multiple views and tight coupling in visualization: A language, taxonomy and system," in [*Workshop of Fundamental Issues in Visualization*], 626–632 (June 2001).

[17] Corrie, B. and Mackerras, P., "Data shaders," in [*Proceedings of the 4th conference on Visualization '93*], *VIS '93*, 275–282, IEEE Computer Society, Washington, DC, USA (1993).

[18] McDonnel, B. and Elmqvist, N., "Towards utilizing gpus in information visualization: A model and implementation of image-space operations," *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis 2009)* **15**(6), 1105–1112 (2009).

[19] Piringer, H., Tominski, C., Muigg, P., and Berger, W., "A multi-threading architecture to support interactive visual exploration," *IEEE Transactions on Visualization and Computer Graphics* **15**(6), 1113–1120 (2009).

[20] Cottam, J. A. and Lumsdaine, A., "Automatic application of the data-state model in data-flow contexts," in [*IV '10: Proceedings of the 2010 14th International Conference Information Visualisation*], IEEE Computer Society, Washington, DC, USA (2010).

[21] Board, O. A. R., Shreiner, D., Woo, M., Neider, J., and Davis, T., [*OpenGL Programming Guide: The Official Guide to Learning OpenGL*], Addison-Wesley Professional (2007).

[22] Khronos OpenCL Working Group, [*The OpenCL Specification, version 1.1*] (September 2010).

[23] Davidson, G. S., Hendrickson, B., Johnson, D. K., Meyers, C. E., and Wylie, B. N., "Knowledge mining with VxInsight: Discovery through interaction," *Journal of Intelligent Information Systems* **11**(3), 259–285 (1998).

[24] Cottam, J. A. and Lumsdaine, A., "Extended assortitivity and the structure in the open source development community," in [*International Sunbelt Social Network Conference*], International Network for Social Network Analysis (January 2008).

[25] Oliphant, T. E., *Guide to NumPy*. Provo, UT (Mar. 2006).

[26] Mueller, C., Martin, B., and Lumsdaine, A., "Interpreting large visual similarity matrices," in [*Asia-Pacific Symposium on Visualization*], (02/2007 2007).

[27] Wilkinson, L., Anand, A., and Grossman, R., "Graph-theoretic scagnostics," in [*Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)*], 157– 164, IEEE Computer Society, Los Alamitos, CA, USA (2005).

[28] Liu, Z., Jiang, B., and Heer, J., "imMens: Real-time visual querying of big data," *Computer Graphics Forum (Proc. EuroVis)* **32** (2013).