PJIM PARSONS JOURNAL FOR INFORMATION MAPPING

THE PARSONS INSTITUTE FOR INFORMATION MAPPING
68 Fifth Avenue New York, NY 10011
212 229 6825 piim.newschool.edu

Preserving Data While Rendering

JOSEPH A. COTTAM, PHD
PETER WANG

KEYWORDS *abstract rendering, alpha composition, color weaving, data transformation, highalpha, homoalpha, overplotting, pixel-level effect, visual mapping,*

ABSTRACT The fundamental premise of visualization is that a useful correspondence between pixels and data can be built. However, visualization programs rarely operate at the pixel level. Instead of the discrete, finite space of pixels, the most common visualization models work with canvases of floating-point coordinates and geometric shapes. Preserving some source data link all the way down to the pixel level provides many opportunities for improving visualization. This paper describes Abstract Rendering (AR), a framework that preserves the data-to-pixel link. Using this pixel-level link, AR is used to provide a unique control over the final visual representation of data sets at all scales and from a variety of visualization technique families.

INTRODUCTION

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

The need to consider pixel-level effects is most apparent when working with point-oriented datasets where the number of items to display exceeds the number of pixels on the screen. For example, the visualizations in FIGURE 2 are based on a dataset that contains one point for every person in the contiguous United States according to the 2010 census[2] (i.e. 306.7 million points). At 300DPI, it would require a sheet of paper approximately five feet square to provide a discrete dot for every person (ignoring actual geographic distribution). Treatments based only on geometric points and alpha composition cannot convey the true dynamic range and spatial distribution of the data (FIGURE 2A). Considering the pixels directly enables a variety of more accurate treatments (FIGURE 2B). AR provides a means of controlling the transition between the geometric representation and the pixel representation.

Succinctly, abstract rendering is the application of data transformations at render time. The fundamental observation is that individual pixels of a visualization

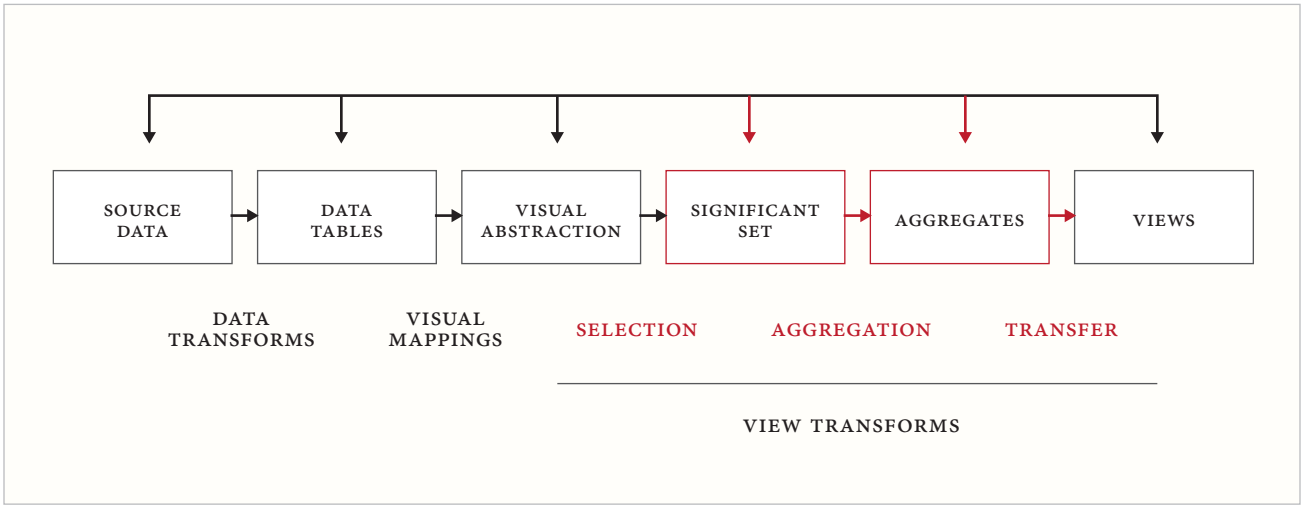


FIGURE 1: *Information visualization reference model Card et al (1999), extended for Abstrat Rendering.*

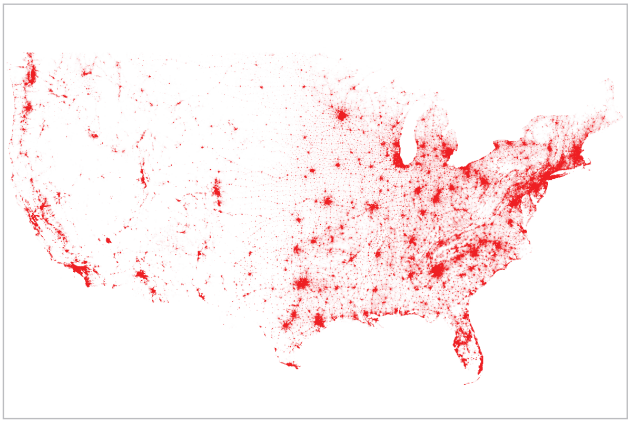
map back to underlying data, mediated by the more common geometric representations. AR provides a means of directly defining that per-pixel transformation. An AR definition consists of two equations (discussed in detail in SECTION 2.1). The first defines per-pixel data summarizations, the second provides transformations of those summarizations (eventually to colors). The system is named abstract rendering because it computes a per-pixel result, much like standard rendering, but the pixel values do not need to be colors and may undergo further transformations (thus, the values remain in a more abstract space). AR enables direct expression of many visualization behaviors, including over-plotting, pixel-oriented techniques, and automated visualization analysis (see SECTION 3).

2 RELATED WORK

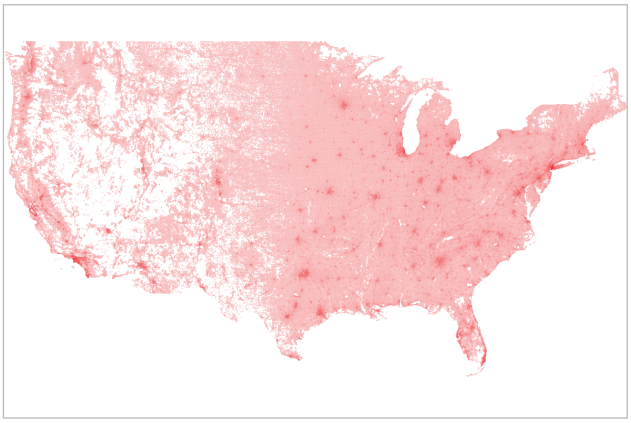
Visualization is most often concerned with pixels in two ways; first, are visualization techniques that are specifically targeted at pixel-level details (so-called "pixel-oriented techniques"). Pixel-oriented techniques try to maximize data preservation by considering individual pixels in the display. 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 to prevent over-plotting.[3]

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. describes a multi-stage technique for high-definition alpha compositing, and Muelder, et al. employs this technique while visualizing MPI communication behavior.[4]

High-definition alpha composition invokes transfer functions more common of scientific visualization.[2] 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 AR follows the pattern of earlier work Muelder et al and Johansson et al., operating on pixels instead of voxels. A high-precision pixel space is created (either explicitly as an



(A) *Standard Alpha Composition*



(B) *Bin-based Counts, Interpolation and Perceptual Corrections*

FIGURE 2: *US Census populations with and without bin-based counts realized.*

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 is a recent technique in this category.[6] Color-weaving presents each relevant color in part of each region it belongs in, rather than blending 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.[7] This is in contrast to standard zoom where only the view's affine transform is modified as screen space is reapportioned. AR can implement semantic zoom by modifying the renderer depending on screen space.[8]

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.[9] These relationships typically override the regular standard visual representations for selected items. It is also similar to Data Shaders and many GPU techniques that use color buffers as data buffers.[10]

Including binning in a visualization pipeline is not unique to abstract rendering; 2013 saw several variations, including imMens, Nanocubes, BigVis, and, to a lesser degree, Scalar.[11] Each of these frameworks shares the core concept of using binning as a fundamental operation, though they differ in when the bins are created and how those bins are used, however, all share an observation that pixel-level data is often useful. Nanocubes, and imMens both focus on in-memory representations and building visual representations off those representations, while BigVis provides additional statistical modeling capabilities over other bin-based frameworks.

Parallelizing rendering is an extensively explored topic. In recent years, Piringer, et al. have described task-parallel rendering based on distinct layers of data and Cottam and Lumsdaine discussed task-and data-parallel rendering techniques for dynamic data.[12] Developments such as OpenGL and OpenCL also add to this discussion. OpenGL is essentially a framework controlling data-parallel analysis related to rendering; the OpenCL framework extends that idea to more general data processing.[13] The framework described in this paper complements much of this research. For example, AR 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).

2.1 ABSTRACT RENDERING

Abstract rendering is done by chaining together functions that fulfill four different roles. The four function-roles are (1) select, (2) info, (3) aggregate, and (4) transfer. These functions are combined in the following fashion:

$$b_{xy} = Aggregate\ (\{Info(g) | g \in Select(G, p_{xy})\})$$
$$c_{xy} = Transfer(b_{xy}, B)$$

The top equation is used to create the pixel-like, discrete representation of the source data from the geometric data. This is essentially a binning process, so the discrete values are referred to as bin values. The "select" function picks geometric items for a given location; the "info" function produces a data value associated with a geometric item; and the "aggregate" function combines lists of info values together to form a bin value. An effective synthetic data space enables further analysis, performed by a "transfer" function. In this formulation 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_{xy} is a geometric representation of a pixel, b_{xy} represents a bin (correspondingly B represents the entire set of bins), and c_{xy} is a final pixel color. Each function fulfilling a particular role may require additional arguments, though the given parameters above are common to most. For ease of expression, multiple bin-transformations stages may be performed. When more than one transfer function is present, intermediate bin-values are numbered and transfers are executed sequentially.

These four functions-roles are populated with specific implementations to produce images. For example, FIGURE 2B is derived from the 2010 US Census. The glyph-input (G) includes 306.7 million data points, one for each

person in the contiguous United States. Points are placed so each census block contains the same number of points as people and are annotated with the race information to match the distributions in the block. This data was originally produced in support of the Racial Dot Map and shared by that project's directory Cable (2013).[14] The bin-space is built by counting the number of input points that land on each pixel, then transforming and interpolating over the full range of those counts. The AR phrasing is:

$$b1_{xy} = size(\{identity(g) | g \in intersects(G, p_{xy})\})$$
$$b2_{xy} = exp(b1_{xy}, 1/3)$$
$$c_{xy} = interpolate(b2_{xy}, red.10, red, min(B2), max(B2))$$

A two-phase transformation is made to do perceptual correction that more closely approximates human response to changes in luminance Stone (2003).[15] The first step (creating the $b1_{xy}$ values) breaks the input data into a bin-grid based on the pixel that the input points land on through the "selector" function intersects. Since only the presence/absence is of interest, the "info" function is identity and the aggregator "size" counts the number of items in a list. The creation of $b2_{xy}$ with the "transfer" function *exp* values is for perceptual correction. The transformed counts are then analyzed to find the minimum and maximum values. A color ramp can be built between those two values and directly applied to the bin values. Therefore, $interpolate(b2_{xy}, red.10, red, min(B2), max(B2))$ supplies the color for each pixel, operating as a second transfer function (*red.10* is red at 10% saturation, $b2_{xy}$ is a single bin value and *B2* is all bin-values from the earlier transformation). 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 census.

The four function-role system can be employed to create more complex representations as well. Extending the census plot to represent race/ethnic-group information in the color requires application of the "info" function. For example, the image of FIGURE 4 is derived from racial annotations on each of the input points. It is phrased in AR:

HIGHALPHA

$$b1_{xy} = countCategories(\{race(g) | g \in intersects(G, p_{xy})\})$$
$$b2_{xy} = reKey(b1_{xy}, \{African: Green, Asian: Red, Caucasian: Blue, ...\})$$
$$c_{xy} = HDAlpha(b2_{xy}, B2)$$

Race is the "info" function, and retrieves a race annotation on each data point. The function *countCategories* is the "aggregator" and creates a list of the unique values seen paired with the sum of all occurrences of that category. *reKey* is a "transfer"function and 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 race/ethnic-group. *reKey* uses its second argument to associate the counts with colors instead. *HDAlpha* is also a "transfer" function , and expects a set of colors and quantities for each pixel.[16]

The four function-role formalization of AR is the basis of two implementations of AR (one in Java, one in Python). Both implementations directly represent all four functions. Despite the relatively high-level representation, we have implemented dozens of bin-based algorithms, and achieved efficient execution in a wide variety of environments.

3 AR APPLICATIONS

The general abstract rendering form can be used to describe many different visualization techniques. This section provides the equations for several existing techniques. Again, example figures provided in this section are derived from the 2010 US Census data.

3.1 OVERPLOTTING

OVERPLOT

$$b1_{xy} = countCategories(\{race(g) | g \in intersects(G, p_{xy})\})$$
$$c_{xy} = HDAlpha(b2_{xy}, B2)$$

Basic overplotting occurs when pixels colors are assigned on a last-write-wins basis. To achieve overplotting in the AR framework, an ordering basis must be established. In the equation shown above, the ordering basis is the z-value of the glyphs. The information function selects the color and the z from each glyph, the reduction picks the color of the glyph with the largest z value. The resulting aggregate-set is a list of colors.

3.2 HOMOGENEOUS ALPHA COMPOSITING

HOMOALPHA

$$b1_{xy} = count(\{id(g) | g \in intersects\ (G, P_{xy})\})$$
$$c_{xy} = interpolate(b_{xy}, red.10, red, min(B), max(B))$$

Instead of selecting which item to render on a particular pixel, blending the items of a pixel is a common technique. Alpha composition is the most common expression of blending.

Homogeneous alpha composition occurs whenever all rendered items have the same visual representation. The synthetics-set is made by counting the glyphs that intersect each pixel (using the count function). The interpolate function then interpolates between two colors (*red.10* being 10% saturated red). The full bin-set (B)is used to establish the low and high input values. Working with the counts directly enables more fine-grained control over the interpolation, avoiding issues of unknowingly over-saturating the alpha-buffer. Therefore,the AR representation is trivially able to implement high-definition alpha composition.[17]

Homogeneous alpha composition is used to produce density representations, like those in FIGURES 2B and 3. The adjacency matrix (FIGURE 3) represents 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. Using standard alpha composition, over 50% of the colored pixels would be at full saturation. With AR, an image that displays more nuanced patterns is achieved.

3.3 HIGH-DEFINITION ALPHA COMPOSITING

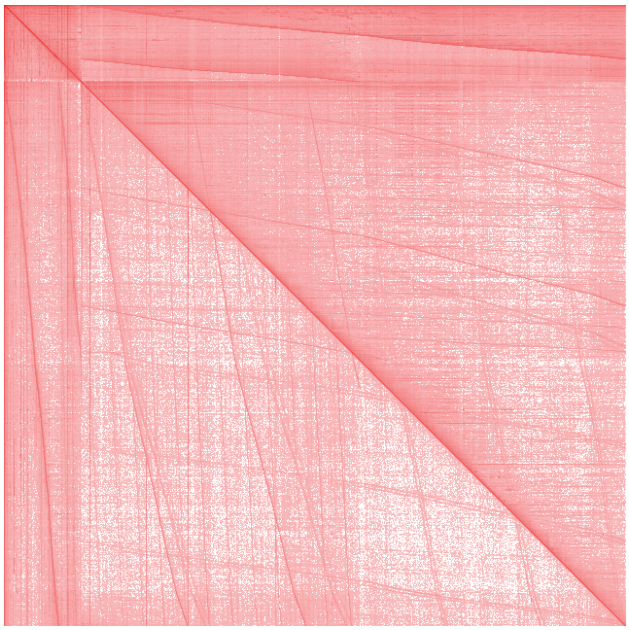


FIGURE 3: *Adjacency matrix of the Kiva dataset with density log transformed and homogeneous alpha compositing applied.*

Full high-definition alpha composition extends the concern of buffer over-saturation seen in homogeneous alpha composition to both the alpha and the color buffers. As with homogeneous alpha composition, the key is to measure the extrema before applying the interpolation

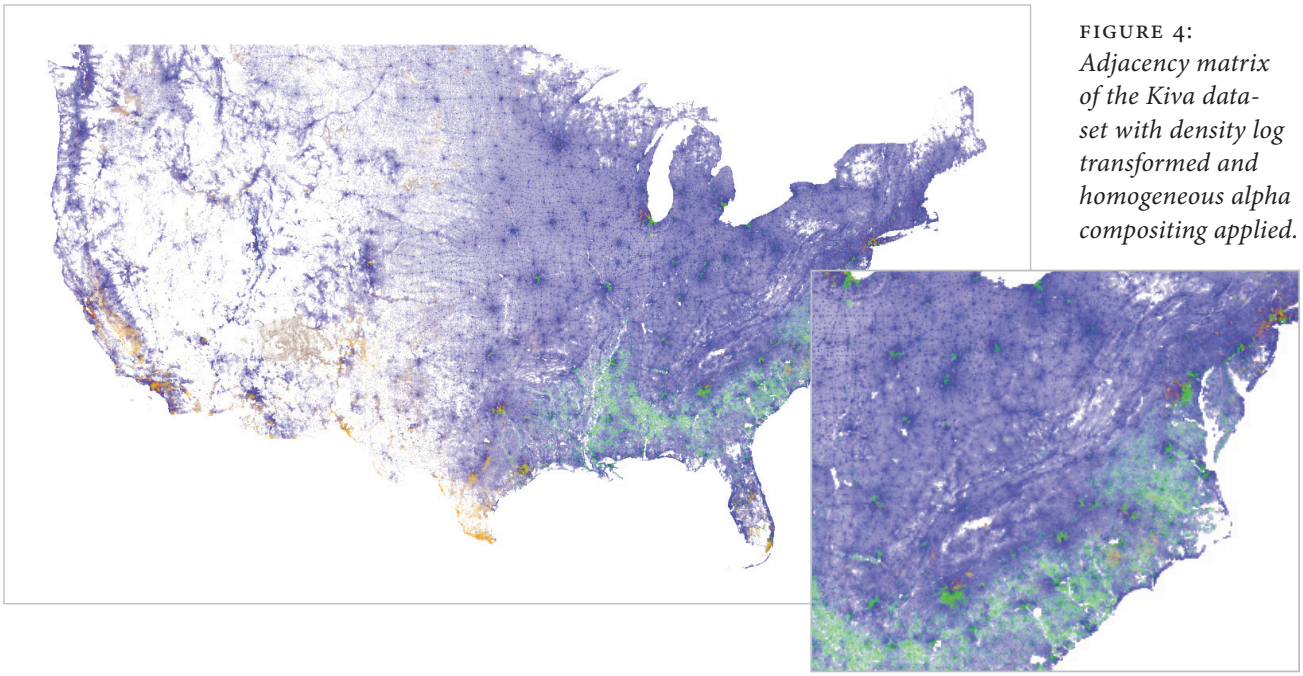


FIGURE 4: *Adjacency matrix of the Kiva dataset with density log transformed and homogeneous alpha compositing applied.*

function. Equation *HighAlpha* from SECTION 2.1 gives the definition for high-definition alpha composition. Details on these range calculations and scalings can be found in the earlier works of Muelder et al and Johansson et al.[18] The necessary range information is derived from the bin-values (B). Equation *HighAlpha* is applied in FIGURE 4.

High-definition alpha composition can be modified by sorting on the output color or a data field to achieve "stratified" alpha composition. This stratification can be used to emphasize particular values in the plot (since alpha composition is order dependent) or provide more efficient rendering (for example, WebGL often performs faster when there are fewer pen-color changes).

3.4 COLOR WEAVING

WEAVE

$$b1_{xy} = countCategories(\{race(g) | g \in intersects(G, p_{xy})\})$$
$$b2_{xy} = reKey(b1_{xy}, \{African: Green, Asian: Red, Caucasian: Blue, ...\})$$
$$c_{xy} = randomCategory(b2_{xy})$$

Color weaving takes an alternative tack to representing more than one item in a space. Rather than blending colors in one pixel as with alpha composition, it represents a mosaic pattern of the original source colors throughout the space. To achieve this, new "selector" and "transfer" functions are is used. Instead of just containing items that land in a pixel, *sameContainer* gets all items that land in the same region as the current pixel. For FIGURE 5, any pixel that lands in a state will get values for all people in the state. The second transfer function, *randomCategory*, selects a single category to from the list and returns just its key (which, by virtue of *reKey*) is a color. Implementing *randomCategory* in accordance with Hagh-Shenas et al, results in proportional color weaving.[19] Modifications to how *randomCategory* selects values yields different weaving variations.

3.5 AUTOMATED VISUALIZATION EVALUATION

Generally, automated visualization evaluation requires modeling perception and cognition for image interpretation, joined with more traditional data analysis techniques on the source data. The bin-based representation in abstract rendering provides a place for some of the relevant models to interact. Two simple perceptual models have already been implemented (*see* FIGURE 5 & 6).

The first model warns of over and under saturation (e.g., dynamic range clipping) in a chosen color ramp.

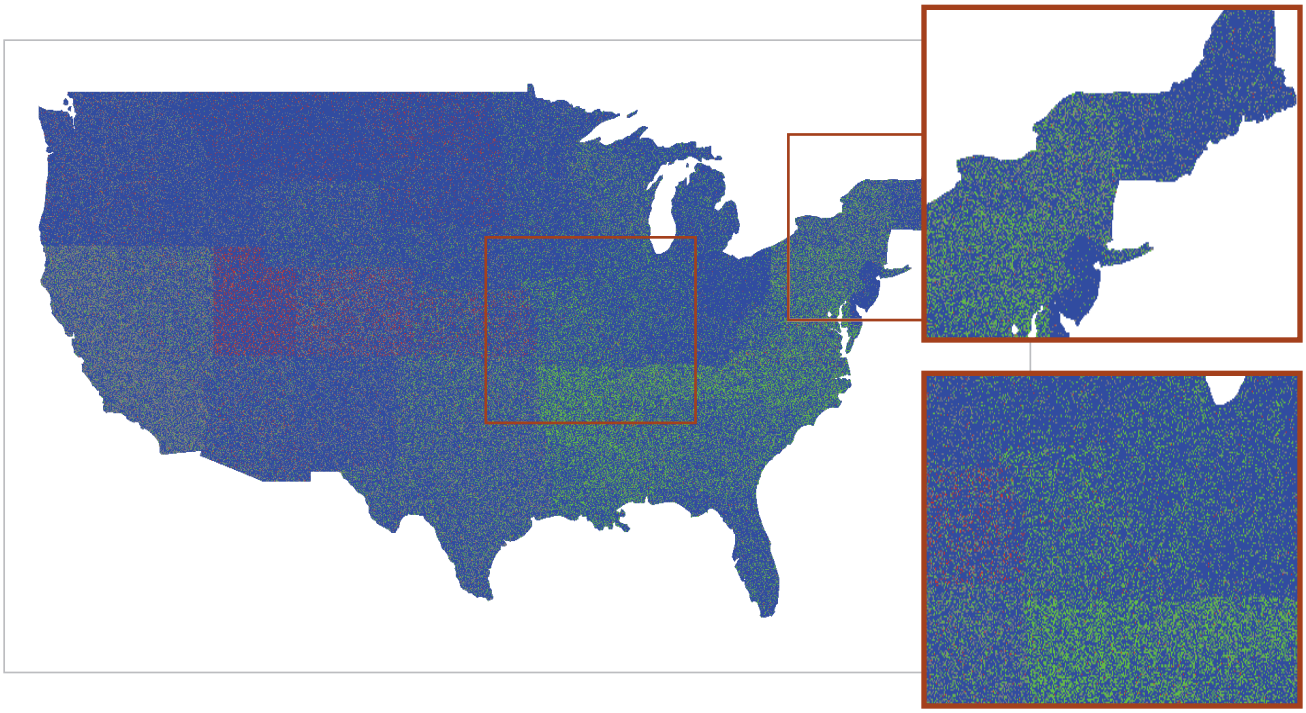


FIGURE 5: *US Census Map with races data aggregated and then woven at the state level.*

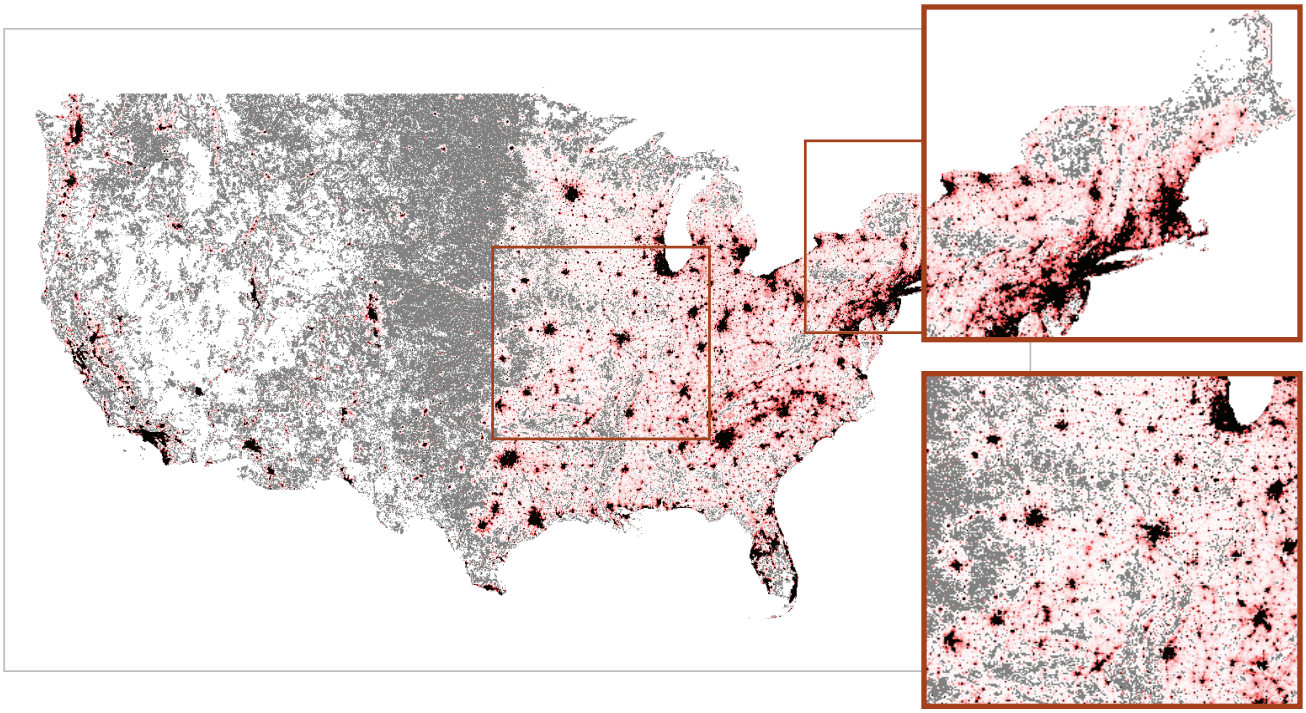


FIGURE 6: *US Census population with standard alpha composition (*FIGURE 2A *& clip-warning).*

FIGURE 6 shows pixels where non-zero values are mapped, colors are indistinguishable from the background and where non-maximal values are mapped to the same value as the maximum. Color comparison is done with a simple model of perception. The AR phrasing extends equation *HomoAlpha* (SECTION 3.2) with a set of functions that measures the outputs and identifies offending pixels.

The second model represents sub-pixel distribution information, indicating locations with non-uniform distributions hidden below the bin aggregation level. FIGURE 7A illustrates the basic premise. Each column in the figure has the same number of items, but a different distribution. FIGURE 7B shows a sub-pixel analysis of the Kiva adjacency matrix. In this analysis the ratio of values in sub-divided pixels is drawn. High ratios are plotted in black, while low ratios are plotted in white. This highlights some interesting asymmetric and temporal features (on the left edge) of the dataset. The basic idea is to preform AR at a higher resolution than the eventual display will have. One transfer step combines all of the higher resolution values that will correspond to a single output pixel and computes a distribution score. FIGURE 7B represents the resulting interpolating distribution scores for FIGURE 3. Darker regions have less uniform distributions.

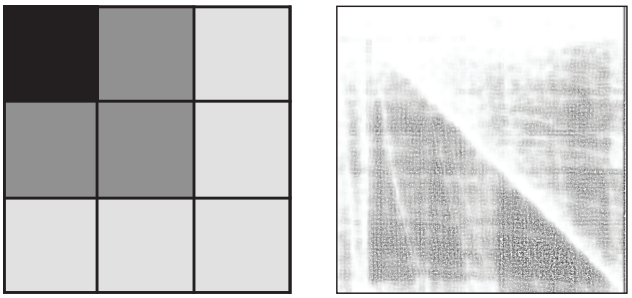


FIGURE 7: *Sub-pixel bin-based analysis provides a view into regions of interesting substructure.*

3.6 SPREADING

SPREADING

$$b_{xy} = select(\{color(g) | g \in intersects(G, p_{xy})\})$$
$$c_{xy} = spread(b_{xy}, x, y, residues(B, x, y))$$

Spreading is a pixel-oriented technique where items that would be over-plotted are instead placed on nearby pixels instead. Spreading algorithms typically work in two phases. In the first phase, items that will not be spread are selected. In the second, a new location for each spread item is selected. Details of the spreading algorithm determine the properties present in the results.[20]

In Equation Spreading, *select* produces a pair; one item to keep in place, and a list of items to spread. (It returns two null items if no item is on the given pixel.) *residues* takes an aggregate set and collects all items that need a new location. *spread* either takes the fixed item from b_{xy} if there is one present or picks an item from the set produced by residues if there was no fixed item in a_{xy}. The details of *select* and *spread* determine the exact spreading algorithm being used. An even more abstract version of spreading can be obtained by replacing color with a data-returning function and including a projection in spread to convert that abstract representation into a color.

3.7 SUBSET ENHANCE

ENHANCE

$$b_{xy} = count(\{id(g) | g \in intersects(G, P_{xy})\})$$
$$c_{xy} = interpolate(b_{xy}, red.10, red, min(subset(B, ...)), max(subset(B, ...)))$$

All prior technique discussions have assumed that the entire dataset was equally relevant. However, in many cases only a subset of the data is of direct concern, the remaining data providing context. Abstract rendering provides a direct means to implement focus-plus-context techniques that operate at a pixel level. For example, homogeneous alpha composition (SECTION 3.2) as applied to the US Census data in FIGURE 2B, uses the full set of bin-values to determine the maximum and minimum values in the range. FIGURE 8 shows an alternative encoding where just a portion of the Midwest is used to setup the dynamic range. Values inside of the box carry the original high definition alpha dynamic range guarantees. Values outside of the box are encoded on the same ramp as those inside the box, but may experience value clipping. The boxed area gets the full dynamic range (i.e. it is 'enhanced'), while other areas are made more obscure. Equation enhance shows homogeneous alpha modified

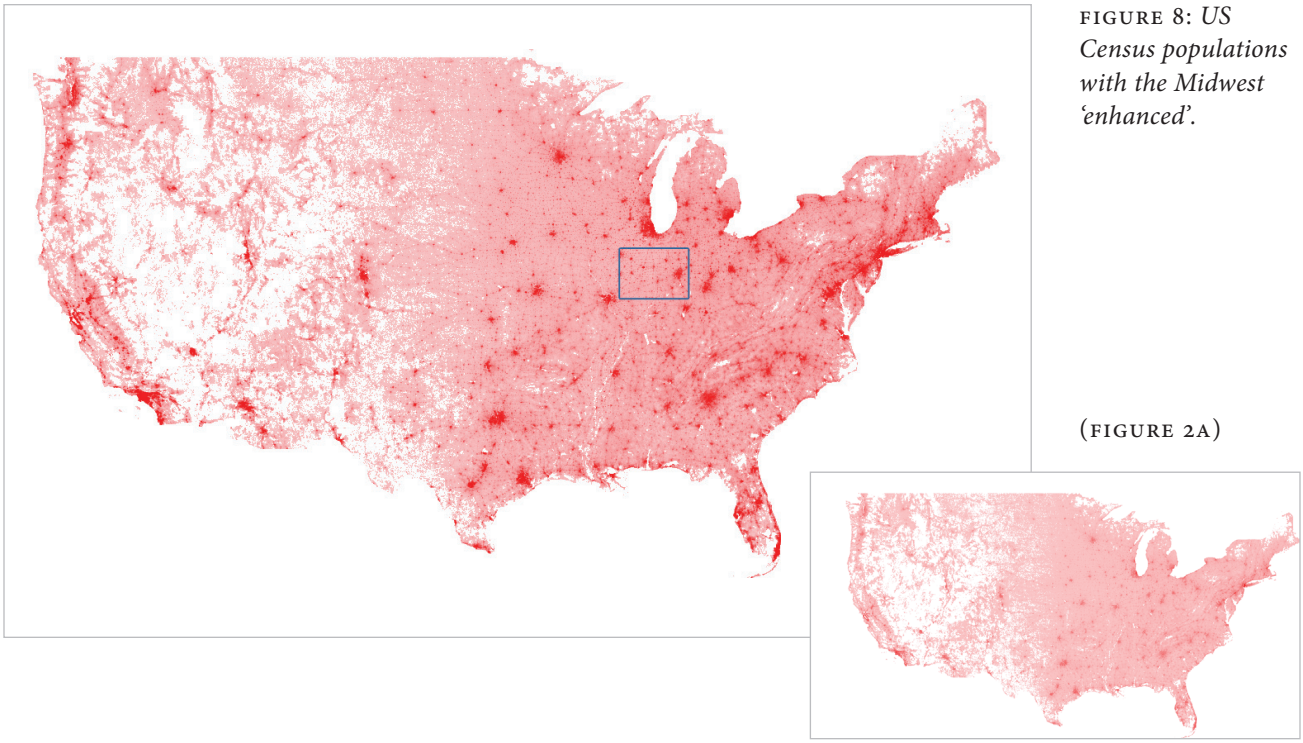


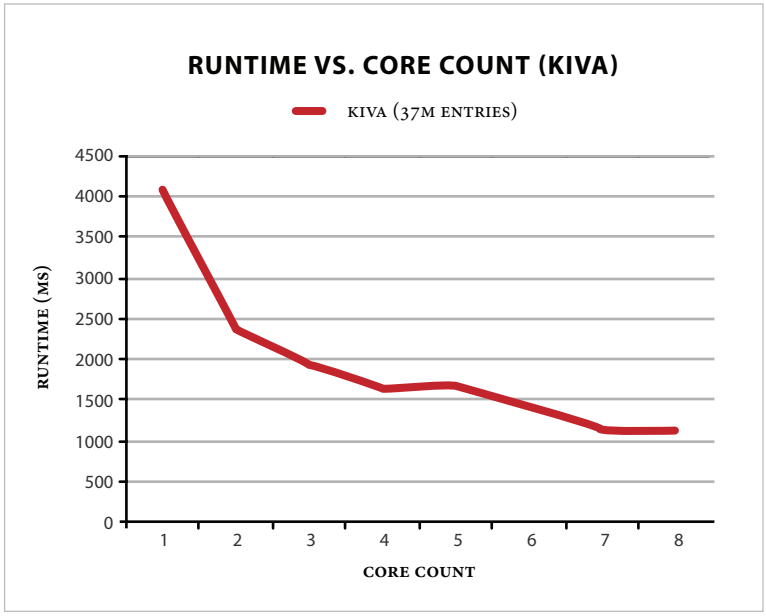
FIGURE 8: *US Census populations with the Midwest 'enhanced'.* (FIGURE 2A)

for region enhancement. The change is to transfer equation by using subset (*B*, ...) instead of *B* directly to compute the max and min values. Corresponding changes can be made to other AR encodings, providing a general means for selective enhancement.

4 PERFORMANCE

Runtime performance is an important part of all visualization frameworks. To be applicable to interactive data analysis, reasonable execution times are required in all parts of the framework. Simply put, the two equations AR phases (see SECTION 2.1) also divide the framework into two performance regimes. Performance in aggregation equation is largely determined by the size of the input data and efficient handling thereof. In contrast, performance in the transfer equation dependent on the number of bin-values, and thus related to resulting image size instead of the input size. Practically speaking, transfer functions are also used for interactivity and thus have significant pressure to reach runtimes less than 100MS.

The Java implementation was used to characterize performance. This implementation closely follows the description given in SECTION 2.1, with a few modifications for efficiency and parsimonious coding. The two most significant changes are that aggregation is based around incremental construction, and transfer is based on bulk-operations. Incremental aggregation construction means that a current aggregate value is combined with a single new info value. This change enables aggregation to only visit each input node once (as opposed to once for each pixel it touches). As a result of incremental aggregation, the aggregator must be commutative and associative and provide an identity value. Transfer with bulk operations means transfer functions are expected to take an entire set of aggregates and produce an entire new set instead of working one value at a time as presented. This change makes it easier to create transfer functions that can be composed and execute efficiently. Both changes contribute significantly to the ability to automatically parallelize the imple-



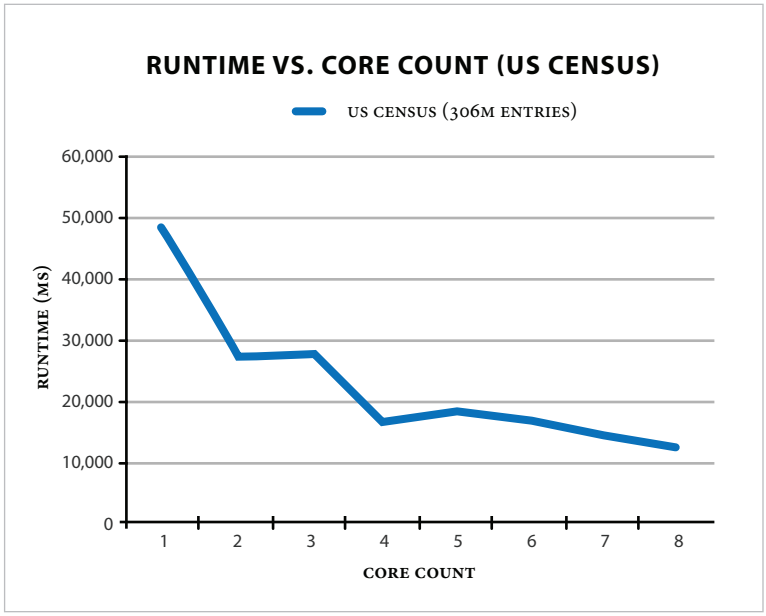


FIGURE 9: *Scaling behavior of* AR *as processor core count and task count in-crease. Both datasets had the same general behavior, though the absolute values differed for the different analysis steps and data size.*

mentation using Java's fork/join framework.

A more complete performance analysis is given in other work.[21] However, the general result is that the abstract rendering implementation scales as more processors are provided to aggregation (FIGURE 9). Furthermore, the general performance of aggregation enables large datasets to be approached (hundreds of millions of items), provided the data can be read efficiently and transformed into a geometric representation. Transfer execution also scales with more processors, maintaining execution speeds below 100MS across the image sizes produced and regardless of the input data size. Overall, the performance numbers are supportive of interactive visualization applications.

CONCLUSIONS

Visualization relies on encoding data in pixels to communicate useful information. Directly preserving that data in the rendering process (instead of indirectly through color or position) enables a wide variety of visualization techniques. These techniques include methods for addressing overplotting, focus-plus-context methods, and automated evaluation. Abstract rendering presents a compact and accessible way to conceptualize these techniques. Using AR, the techniques discussed in this paper can be compactly described and efficiently executed. Overall, retaining data long into the rendering process is a practical way to deal with datasets of all scales to produce meaningful visualizations.

ACKNOWLEDGEMENTS

This work was funded in part under the DARPA XDATA program and through the Lilly Endowment.

BIOGRAPHY

Joseph A. Cottam is a postdoctoral researcher at Indiana University. His work focuses on systems and programming languages to support data analysis and visualization.

Peter Wang is the president and co-founder of Continuum Analytics. He has worked on data analysis and visualization products for the past ten years.

NOTES

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

2 D. Cable. The Racial Dot Map – Weldon Cooper Center for Public Service. http://www.coopercenter.org/demographics/Racial-Dot-Map 2013, July

3 D.A. Keim. *Designing Pixel-oriented Visualization Techniques: Theory and Applications. IEEE Transactions on Visualization and Computer Graphics* 6 (1), 59–78. 2000, January

4 J. Johansson, P. Ljung, M. Jern, and M. Cooper. "Revealing Structure Within Clustered Parallel Coordinates Displays." In Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization, INFOVIS '05, Washington, DC, USA, pp. 17–. IEEE Computer Society; C. Muelder, F. Gygi, and K.-L. Ma (2009, November). Visual analysis of inter- process communication for large-scale parallel computing. IEEE Transactions on Visualization and Computer Graphics 15 (6), 1129–1136. 2009, November

5 S.T. Biddlecome,Fang and M. Tuceryan. "Image-based Transfer Function Design for Data Exploration in Volume Visualization." In Proceedings of the conference on Visualization 1998, VIS '98, Los Alamitos, CA,

6 T. Urness, V. Interrante, I. Marusic, E. Longmire, and B. Ganapathisubramani. "Effectively Visualizing Multi-Valued Flow Data Using Color and Texture." In Proceedings of the 14th IEEE Visualization 2003 (VIS'03), VIS '03, Washington, DC, USA, pp. 16–. IEEE Computer Society; H. Hagh-Shenas, V. Interrante, C. Healey, and S. Kim. "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 2006, New York, NY, USA, pp. 164–164. ACM.

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

8 J. Heer and M. Agrawala. *Software Design Patterns*

for Information Visualization. IEEE transactions on visualization and computer graphics 12 (5), 853–60. 2006.

9 C North. "Multiple Views and Tight Coupling in Visualization: A Language, Taxonomy and System." In Workshop of Fundamental Issues in Visualization, Las Vegas, NV, USA, pp. 626–632. 2001, June.

10 B. Corrie and P. Mackerras. Data shaders. In Proceedings of the 4th conference on Visualization 1993, VIS '93, Washington, DC, USA, pp. 275–282. IEEE Computer Society; B. McDonnel and N. Elmqvist. "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.

11 Z. Liu, B. Jiang, and J. Heer. "imMens: Real-time Visual Querying of Big Data." Computer Graphics Forum (Proc. EuroVis) 32. 2013. ; L. Lins, J. T. Klosowski, and C. Scheidegger. "Nanocubes for Real-time Exploration of Spatiotemporal Datasets." IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis 2013) 19 (12), 456–2465; H. Wickham. "Bin-Summarize-Smooth: A Framework for Visualizing Large Data." Technical report, had.co.nz. 2013.; M. Stonebraker, L. Battle, and R. Chang. "Dynamic Reduction of Query Results for Interactive Visualization." In IEEE The First Workshop on Big Data Visualization. 2013, October

12 H. Piringer, C. Tominski, P. Muigg, and W. Berger. "A Multi-threading Architecture to Support Interactive Visual Exploration." IEEE Transactions on Visualization and Computer Graphics 15 (6), 1113–1120. 2009; J.A. Cottam. and A. Lumsdaine. "Automatic Application of the Data-state Model in Data-flow Contexts." In IV '10: Proceedings of the 2010 14th International Conference Information Visualization, Washington, DC, USA. IEEE Computer Society.

13 O. A. R. Board, D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison- Wesley Professional. 2007; Khronos OpenCL Working Group (2010, September). The OpenCL Specification, version 1.1.

14 See NOTE 2

15 M. C. Stone. *A field guide to digital color.* A K Peters. 2003.

16 C. Muelder, F. Gygi, and K.-L. Ma. "Visual Analysis of Inter-process Communication for Large-scale Parallel Computing." IEEE Transactions on Visualization and Computer Graphics 15 (6), 1129–1136. 2009, November

17 Ibid

18 See NOTE 4

19 H. Hagh-Shenas, V. Interrante, C. Healey, and S. Kim. "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 2006, New York, NY, USA, pp. 164–164. ACM.

20 See NOTE 3

21 J.A. Cottam, A. Lumsdaine, and P. Wang. "Overplotting: Unified Solutions Under Abstract Rendering." In IEEE The First Workshop on Big Data Visualization. 2013, October.