Stevens Dot Patterns for 2D Flow Visualization

Laura G. Tateosian Brent M. Dennis Christopher G. Healey*
Computer Science Department, North Carolina State University

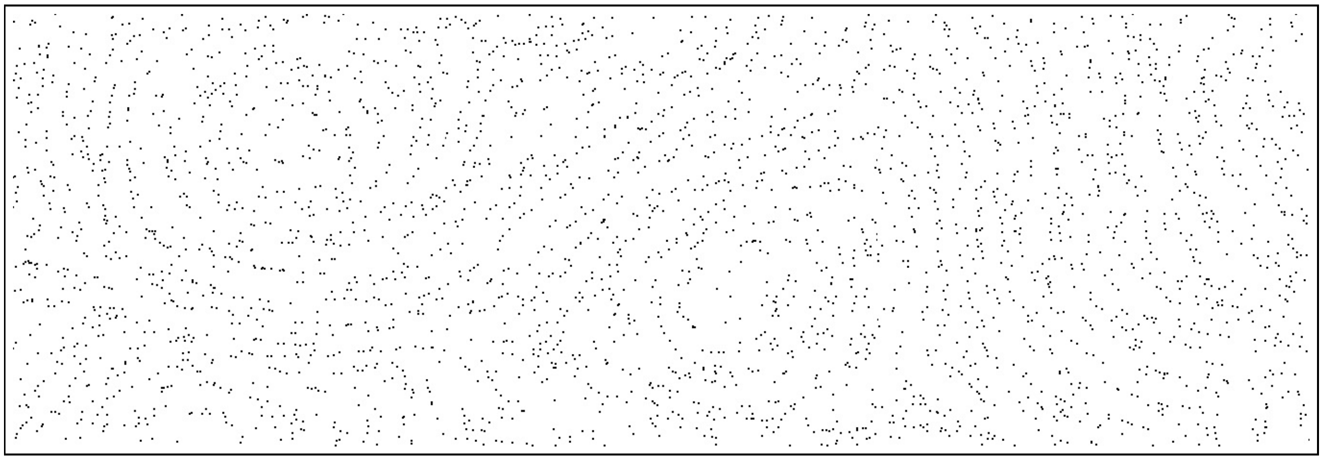


Figure 1: A Stevens-based dot pattern visualizing flow orientations in a 2D slice through a simulated supernova collapse

Abstract

This paper describes a new technique to visualize 2D flow fields with a sparse collection of dots. A cognitive model proposed by Kent Stevens describes how spatially local configurations of dots are processed in parallel by the low-level visual system to perceive orientations throughout the image. We integrate this model into a visualization algorithm that converts a sparse grid of dots into patterns that capture flow orientations in an underlying flow field. We describe how our algorithm supports large flow fields that exceed the capabilities of a display device, and demonstrate how to include properties like direction and velocity in our visualizations. We conclude by applying our technique to 2D slices from a simulated supernova collapse.

CR Categories: H.1.2 [**Models and Principles**]: User/Machine Systems—*Human factors, human information processing;* I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Display algorithms*

Keywords: color, flow, motion, multidimensional, orientation, perception, vision, visualization

1 Introduction

This paper describes a new technique to visualize two-dimensional flow fields using individual points (or dots). Our goal is a technique that minimizes the amount of on-screen information needed to encode flow orientation. Much of the motivation for our work comes from perceptual experiments designed to study how the human visual system "sees" local orientation in a sparse collection of dots. Initial work by Glass suggested that a global autocorrelation is used to identify orientation [Glass 1969; Glass and Perez 1973]. Later work by Stevens hypothesized that the orientation of virtual lines between pairs of dots around a target position defines the local orientation we perceive at that position [Stevens 1978]. This distinction is important, because it implies that different types of flow patterns can be rapidly perceived in different parts of a single image (Fig. 1). Stevens presented both experimental results and a software system to support his vision model.

When we saw the Glass and Stevens images, we were struck by their similarity to a flow visualization. We wondered if it was possible to reverse the process, that is, rather than using Stevens's model to determine the perceived local orientation at each dot, could we position a small collection of dots to produce perceived local orientations that match an already-existing flow pattern? This paper describes how we achieved this goal.

A number of issues motivated our interest in pursuing this technique. First, since a Stevens-based flow visualization is built on a strong perceptual foundation, it may allow us to make certain guarantees about a viewer's ability to see the information (i.e., the flow orientations) we are visualizing. Second, the sparse nature of the dot patterns allows information to be placed in the background, yet still be visible. Colleagues who are studying methods for extending line integral convolution (LIC) techniques to visualize multidimensional flow data are excited about this possibility. Finally, we were inspired by the simplicity and effectiveness of Stevens's algorithm, which describes how a seemingly random collection of dots can combine with one another to form intricate flow patterns.

The remainder of this paper is organized as follows. First, we present a short background review of related flow visualization techniques. Next, we describe Glass and Stevens dot patterns, and discuss Stevens's local orientation model in detail. We present a hierarchical visualization algorithm that converts a sparse grid of dots into a Stevens dot pattern that captures flow orientations in a user-selected flow field. We discuss methods for including flow di-

*e-mail: healey@csc.ncsu.edu

rection and velocity in our images, and demonstrate our technique with real-world flow datasets. Finally, we conclude with a summary of our findings and directions for future research.

2 Background

Numerous methods have been proposed to visualize vector fields in general, and flow data in particular. We briefly discuss previous work that is most directly related to our proposed technique, with specific effort made to highlight areas where we may be able to offer improvements over existing methods.

Streamlines and streaklines. A well known technique for visualizing flow direction is to place dye sources within the flow field, then advect the dye either in the direction of flow (in a steady field) or dynamically over time (in an unsteady field). This produces streamlines or streaklines that visualize the shape of the flow field. One requirement is the need to correctly select the dye source positions without overpopulating the display. Interesting regions in the flow field will be missed if no dye passes through the given phenomena.

Glyphs. Another common way to visualize flow fields is to use geometric elements or glyphs that “point” in the flow orientation at set locations in the flow field (e.g., arrows as shown in [Kirby et al. 1999; Turk and Banks 1996]). Normally, the entire field is seeded with a systematic distribution of glyphs to visualize flow throughout the field. The appearance of each glyph can be augmented to display additional data values, for example, scaling to show magnitude, or coloring to show field potential. One disadvantage of glyphs is the amount of screen space they require. Numerous pixels are normally needed to position and render each glyph. This can reduce the number of flow samples that can be visualized simultaneously on a single screen.

Dense textures. A number of powerful flow visualization techniques based on dense texture representations have been proposed. Original work in this area includes spot noise [van Wijk 1991], where ellipsoid-shaped kernels are seeded within a flow field and convolved with white noise to produce a luminance pattern that highlights flow orientations, and line integral convolution (LIC) [Cabral and Leedom 1993], a texture-based method that advects target positions through the flow field to build streamlines that are convolved with a noise pattern to produce thread-like textures that follow flow orientations. More recent work improves algorithm efficiency (e.g., Fast LIC [Hege and Stalling 1998]) and extends the techniques to unsteady flow fields (e.g., time-dependent spot noise [de Leeuw and van Liere 1997] or Unsteady Fast LIC [Shen and Kao 1997]). Real-time, interactive algorithms have also been presented, for example, image-based flow visualization (IBFV) [van Wijk 2002], where small quadrilaterals are advected and decayed at each timestep, then blended with a warped image of the previous timestep. Warping is based on flow direction. Blending occurs on the GPU, further improving IBFV’s execution time.

Although our algorithm produces results that are glyph-like in appearance, we differ from existing methods, both in how we construct our patterns, and in the end result that we generate. The positions of the dots in our visualizations are based on a model of how we perceive local orientations within a sparse dot field. Rather than constructing a texture that fills the screen, we try to minimize the amount of information (i.e., the number of dots) we need to display to visualize the underlying flow orientations. Our technique was specifically designed to handle large flow fields that cannot fit on a single screen. Hierarchies of dot patterns form multiple levels of detail that allow us to show both an overview of a flow field, and fine-grained context contained in individual sample points.

Stevens dot patterns also have a number of limitations relative to existing flow visualization algorithms. Currently, we only support steady flow. Preprocessing time is needed to generate the dot patterns, so our system does not allow users to interactively vary the flow data. Like glyphs, our algorithm requires a minimum spacing between samples to ensure appropriate local neighborhood sizes. Although we include all the original data at the lowest level of the visualization hierarchy, higher levels produce filtered representations of the flow field’s features.

Many of these disadvantages are similar to ones that existed for the original spot noise or LIC algorithms. In spite of this, we believe the use of human visual perception offers an interesting perspective on flow visualization, and that with further effort many of these limitations can be overcome. Even in its current form, the Stevens algorithm provides benefits that may be useful for specific visualization problems (e.g., using the economy of screen space to show two independent flow fields: an underlying dense texture representation with a perceptual dot pattern overlaid on top).

3 Stevens’s Local Orientation Model

Numerous methods have been proposed, both to explain how we perceive orientation, and to display oriented elements (e.g., arrows, sticks, or streamlines) to visualize direction. One technique that has not been used in visualization, at least to our knowledge, is the careful positioning of a small set of dots to produce the perception of an underlying flow pattern.

Leon Glass studied how the visual system can rapidly perceive structure in a set of non-directional elements, specifically, in a sparse collection of dots [Glass 1969; Glass and Perez 1973]. Glass started with a random collection of dots X, then applied a global transformation T to each dot to produce a second pattern $Y = T(X)$. Although both X and Y appear random in isolation (Fig. 2a), when they are composited together the transformation used to produce Y from X is clearly visible (Fig. 2b). The composite images are often referred to as *Glass patterns*. Glass hypothesized that the low-level visual system performs a global autocorrelation to identify the transformation contained in the image. Numerous transformations were shown to produce this effect, including translations, spirals, and rotations.

Later work by Kent Stevens continued the investigation of Glass patterns and the visible structure they produce [Stevens 1978]. Stevens suggested that the visual system does not use an autocorrelation to identify the transformation in a Glass pattern. Rather, he believed that a simple local operator was applied in parallel to each dot to identify a perceived orientation at that position in the image. Stevens hypothesized that the visual system assumes relatively stable orientations within a local spatial region. He proposed a model that identifies neighbors for a target dot, then builds virtual lines between all pairs of neighbors. The virtual line orientation that occurs most frequently is the one that the visual system perceives at the target dot (i.e., we perceive an orientation at the target dot that is “similar” to the orientations around that dot). Fig. 2c uses short line segments to show the orientations that Stevens’s model computes for each dot in Fig. 2b.

One important consequence of Stevens’s interpretation is the ability to perceive different transformations applied to different regions in a dot pattern (Fig. 1). A global autocorrelation would only allow a single transformation to be perceived within an image. In fact, Stevens generated exactly this type of multiple-transformation image, and showed that the visual system can easily distinguish the different transformations. Stevens further validated his model by comparing its results to transformations reported by human subjects across a collection of modified Glass patterns. Analysis showed that parameters within Stevens’s model could be set to duplicate human performance.

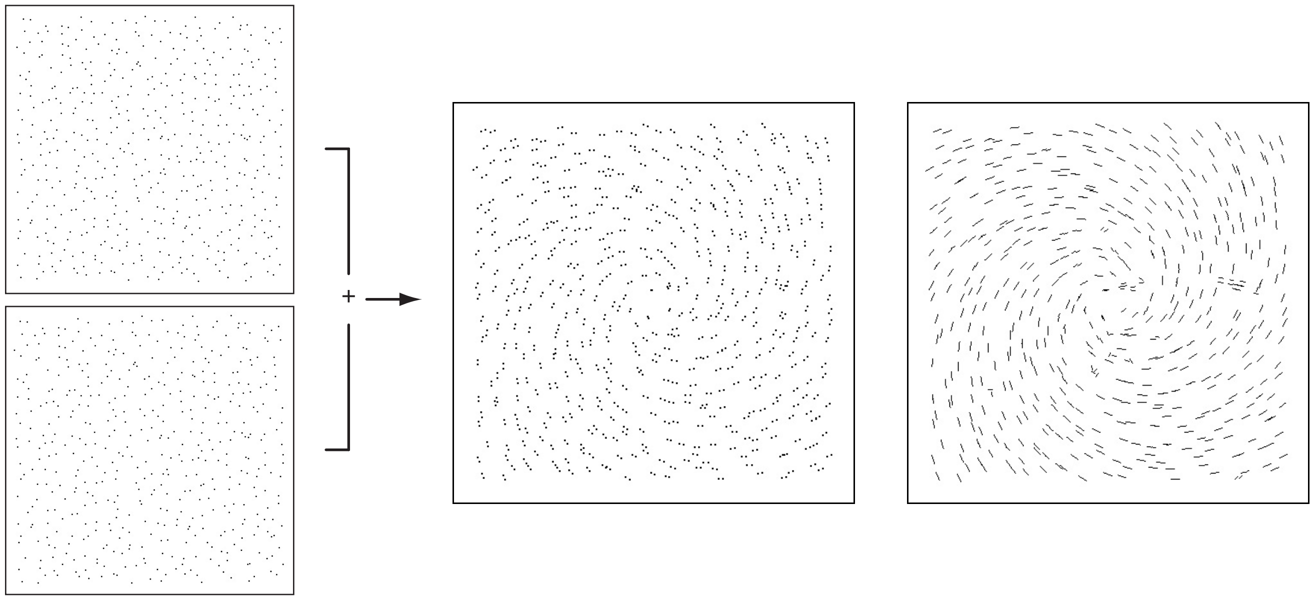


Figure 2: Glass patterns: (a) two random dot sets, the bottom set is a transformed version of the top set; (b) the Glass pattern created by compositing the two sets; (c) Stevens's perceived orientation for each dot in (b), shown as an oriented line segment

3.1 Model Design

The goal of Stevens's algorithm is to assign a *perceived orientation* to each dot A in an image. This is done by computing a histogram of the orientations of virtual lines formed by pairs of dots in the local spatial neighborhood of A. The result is an orientation $\theta_{A,h}$ with the highest frequency in the histogram. Finally, the virtual line from A to a neighbor $n_{A,i}$ with orientation $\theta_{A,i}$ closest to $\theta_{A,h}$ is identified. $\theta_{A,i}$ is the perceived orientation assigned to A.

The histogram divides the continuous set of non-directional orientations $[0,180)$ into a discrete number of ranges. Stevens found that his algorithm most accurately modeled human performance with ranges of width 10°. The histogram is therefore divided into 18 buckets $[0,10)$, $[10,20)$, …, $[170,180)$.

Neighborhoods are formed as circular regions of radius r centered about a target dot. For example, the solid circular boundary centered about A in Fig. 3 shows A's neighborhood, containing neighbors B, C, D, E, F, and G. The dashed circular boundary about C shows C's neighborhood, containing neighbors A, G, H, and J. These neighborhoods are used to construct the virtual lines whose orientations are added to the histogram. For each neighbor $n_{A,i}$ of A, we construct $n_{A,i}$'s neighborhood to identify its neighbors, build virtual lines from $n_{A,i}$ to each of its neighbors, then record the orientations of these virtual lines in the histogram. Duplicate virtual lines (e.g., CG and GC) are only recorded once. The histogram is complete when every neighbor $n_{A,i}$ of A has been processed in this manner.

The contribution of each virtual line's orientation to the histogram is weighted based on the virtual line's length l. Intuitively, the neighborhood radius r is the maximum distance the visual system will travel to associate two dots. The farther the dots are from one another, the less likely the visual system is to consider them as a virtual line, and therefore the less weight that line's orientation will contribute towards deciding on a perceived orientation. Stevens suggested the following weights w, again derived from matching his algorithm's performance to human subject results:

$$w = \begin{cases} 1, & \text{if } \quad l \le \frac{1}{4}r \\ \frac{2}{3}, & \text{if } \frac{1}{4}r < l \le \frac{1}{2}r \\ \frac{1}{3}, & \text{if } \frac{1}{2}r < l \le r \end{cases} \tag{1}$$

Once the histogram is populated, its peak can be identified to define $\theta_{A,h}$. The final step is to examine all virtual lines from A to a neighbor $n_{A,i}$ and select the line whose orientation $\theta_{A,i}$ is closest to $\theta_{A,h}$. This orientation $\theta_{A,i}$ is assigned as the perceived orientation of A.

Fig. 3 shows an example of this process for A, with participating dots $B,\ldots,M$ shown about A, and the resulting histogram displayed below the dot pattern. The peak in the histogram occurs at $\theta_{A,h} = 130°$. Among virtual lines AB, AC, AD, AE, AF, and AG, AB's orientation $\theta_{A,B} = 107°$ is closet to $\theta_{A,h}$. A perceived orientation of 107° is therefore attached to dot A.

4 Flow Visualization Algorithm

When we first saw Glass and Stevens dot patterns, we were struck by their correspondence to flow visualization images. Various glyph-based flow visualization techniques produce results that are visually similar to a Stevens dot pattern. The details of the algorithms used to generate these images are very different, however. More importantly, Stevens's algorithm *reports* on perceived orientations in an already-existing set of dots; it cannot be used to *generate* a desired orientation pattern.

Because of the visual simplicity and strong perceptual characteristics of the Stevens images, we wondered: "Is it possible to integrate Stevens's model of perceived orientation into an algorithm that generates dot patterns to visualize a user-selected vector field?" That is, given a 2D flow field, can we generate a dot pattern whose perceptual orientations capture the flow directions embedded in that field?

In addition to being an interesting academic exercise, the use of Stevens dot patterns may offer some important advantages over

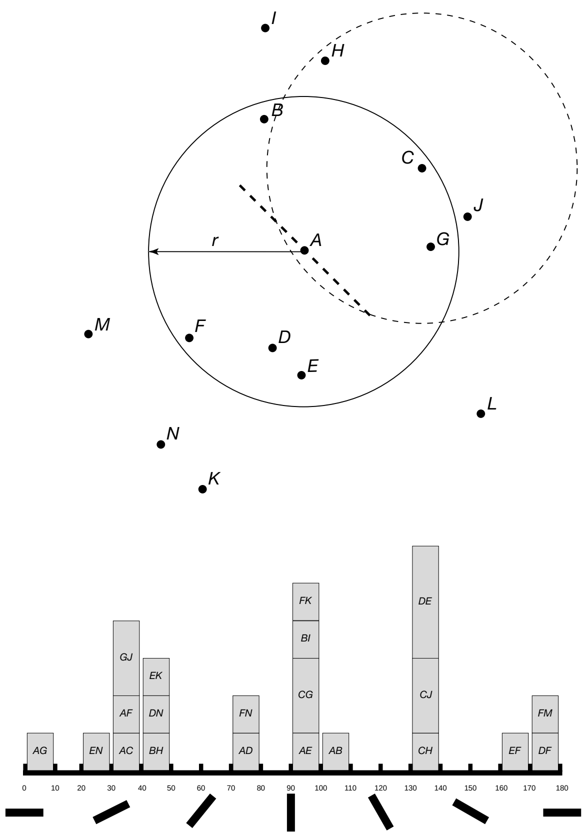


Figure 3: Perceived orientation at *A*, dashed line shows actual flow orientation $\theta_{A,f} = 135°$: *A*'s neighbors lie within distance *r* from *A* (solid boundary). Each neighbor forms virtual lines with its neighbors to contribute to *A*'s histogram (e.g., *C*'s neighbors *A*, *H*, *G*, *J* form entries *AC*, *CH*, *CG*, *CJ*). *AB*'s orientation of 107° most closely matches the histogram peak of 130°, producing perceived orientation at *A* of 107°

existing flow visualization methods. For example:

- Stevens's algorithm is modeled on how the human visual system perceives local orientation, so we can provide certain guarantees on the visual salience of our flow patterns.
- Stevens dot patterns are sparse, so they allow additional data to be visualized in the background layer beneath the dots. For example, we could combine a Stevens-based technique with a multidimensional visualization technique to visualize multi-attribute flow data.
- Our investigation of how to use Stevens's algorithm to visualize flow data may further extend our knowledge of perceived orientation. These discoveries would be useful to both the visualization and the psychophysics communities.

4.1 Design

Our technique begins by seeding a flow field with a jittered grid of dots. An error e_A is assigned to each dot *A* to describe the difference between its perceived orientation and the actual orientation of the flow field at *A*'s position. The dots are placed in a priority queue ordered by error. The algorithm then iterates over the dots as follows:

1. Remove the dot *A* with the highest error at the front of the priority queue.
2. Evaluate how *A*'s neighbors contribute to its error.
3. Select a neighbor $n_{A,i}$ and move it to reduce *A*'s error.
4. Re-evaluate the error of all dots affected by $n_{A,i}$'s move, then decide whether to accept or reject the move.
5. If the move is accepted, update the error of each affected dot, then re-insert them into the priority queue.

This iteration continues until the sum of the error over all dots falls below a threshold value.

4.2 Initial Dot Patterns

Stevens initially used a random distribution of dots for the images in his experiments. However, he found that clusters, sparse regions, and chains of points appeared when the patterns were transformed and composited. For a controlled experiment, these visual irregularities are problematic, because they provide clues about the transformation being applied, and can bias subject performance. Stevens hand-corrected his images to remove any artifacts he identified.

Though the intent of our displays differs from Stevens, we found that a random initial distribution resulted in visualizations with the same types of clustering and sparse regions. Areas containing these artifacts do a poor job of visualizing the underlying flow directions. These artifacts seem to persist because our algorithm moves dots locally, and does not reposition them over long distances. To correct for this, we construct an initial distribution of dots on a regular grid, then apply a small jitter to each dot. Now, the use of spatially local operations is advantageous, since it helps to maintain the relatively uniform distribution of our dots.

4.3 Neighborhood Radius

Computing perceived orientation and the corresponding error for a dot *A* depends critically on the radius *r* of the local spatial neighborhood. Stevens did not define the ideal value for *r*, but it can be estimated from his experiment results.

Given a display density $\rho = \frac{D}{n}$, the ratio of the number of dots *D* to the total number of pixels *n* in a display, the average number of dots $\overline{D}$ in a circle of radius *r* is:

$$\overline{D} = (\pi r^2)\rho \qquad (2)$$

Stevens's experiments showed ρ of 0.0085 and neighborhoods containing six to seven points matched human performance. Using $\rho = 0.0085$ and $\overline{D} = 7$, we can therefore define *r* to be:

$$r = \sqrt{\frac{\overline{D}}{\rho\pi}} = \sqrt{\frac{7}{0.0085\pi}} = 16.19 \text{ pixels} \qquad (3)$$

4.4 Orientation Error

Our algorithm arranges the dots so that the perceived orientation of each dot matches closely the flow orientation at the dot's location. Given a perceived orientation $\theta_{A,i}$ for dot *A* and a flow orientation $\theta_{A,f}$ at the location of *A*, their absolute difference $|\theta_{A,i} - \theta_{A,f}|$ represents the *perceptual error* at *A*.

Initial testing showed that managing perceptual error alone is not sufficient to move dots in a way that converges to an acceptable solution. Initially, the total perceptual error over all dots decreases as individual dots are moved, but it often stabilized well above our desired error threshold. Further investigation revealed that common dot patterns with moves that produce improved configurations do

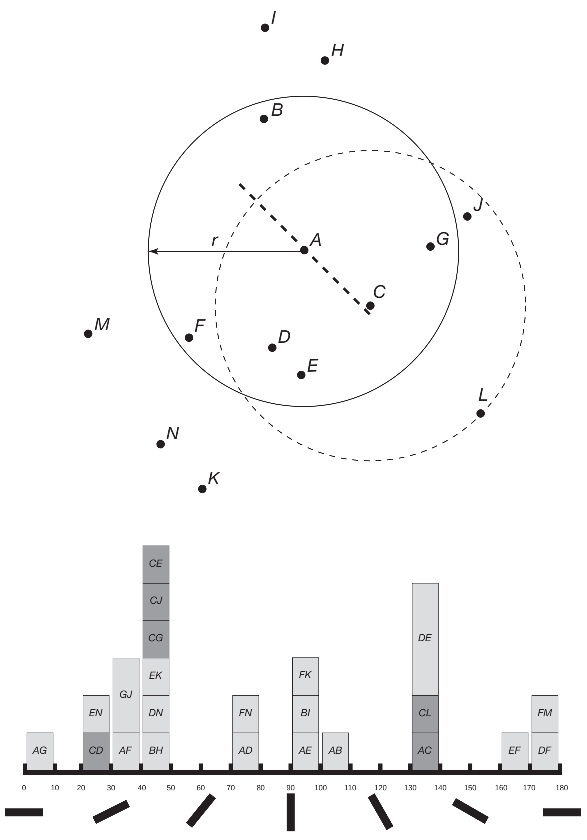


Figure 4: Virtual line error: C is moved to form a virtual line AC closer to histogram peak of 130° (Fig. 3), reducing $e_{A,v}$. Histogram entries involving C change from Fig. 3, however, producing a new histogram peak at 40° and histogram error $e_{A,h} = |40 - 135| = 95°$. AF's orientation of 37° most closely matches the new histogram peak with virtual line error $e_{A,v} = |37 - 40| = 3°$

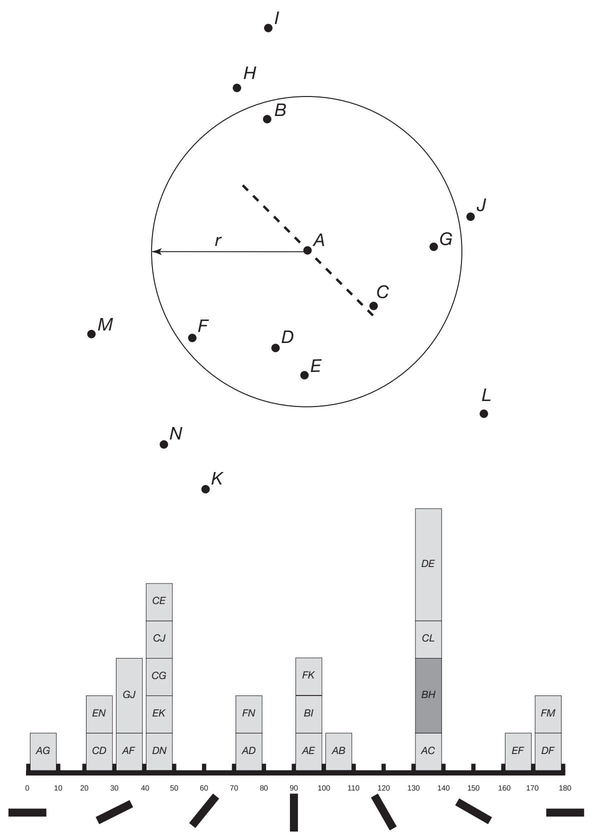


Figure 5: Histogram error: Given the histogram error in Fig. 4 $e_{A,h} = |40 - 135| = 95°$, BH is randomly selected from the 40° bucket and shifted to orientation 135°, producing a new histogram peak. Since histogram correction is designed to preserve previous virtual line corrections, AC from Fig. 4 is maintained to produce perceived orientation at A of 130°, a good match to $\theta_{A,f}$.

not always decrease the total perceptual error. To correct this, we decomposed perceptual error into two separate metrics: *histogram error* and *virtual line error*.

Recall that perceived orientation at a dot A is computed in two steps. First, a histogram of the neighbor's virtual line orientations is constructed, with the peak of the histogram $\theta_{A,h}$ selected as the desired orientation for A. Second, the virtual line between A and neighbor $n_{A,i}$ with orientation $\theta_{A,i}$ closest to $\theta_{A,h}$ is identified, with $\theta_{A,i}$ assigned as A's perceived orientation. These steps represent two sources of potential error when we try to match perceived orientation to an existing flow orientation. The difference $e_{A,h} = |\theta_{A,h} - \theta_{A,f}|$ between the histogram's peak and the desired orientation could be large. This is histogram error. Even if $e_{A,h}$ is small, the orientation $\theta_{A,i}$ of the closest virtual line through A could be far from the histogram's peak, producing a difference $e_{A,v} = |\theta_{A,i} - \theta_{A,h}|$ that is still large. This is virtual line error.

We use a linear combination of histogram and virtual line error to form *bivariate error* $e_A = c_1 e_{A,h} + c_2 e_{A,v}$, where c_1 and c_2 are constants used to weight the different error metrics. Our testing showed that $c_1 < c_2$ produces a faster, more stable reduction in error. A neighbor (or a neighbor's neighbor) $n_{A,i}$ of A is moved to correct for both types of error. This move affects not only A's error, however, but also the error of any other point B when $n_{A,i}$ is a neighbor or a neighbor's neighbor of B. Moving a single dot $n_{A,i}$ to reduce A's virtual line error will normally produce only small disruptions in the peaks of the histograms that share $n_{A,i}$. Moving $n_{A,i}$ to reduce histogram error may significantly increase virtual line error if $n_{A,i}$ forms the endpoint of a virtual line to B, however. Choosing $c_1 < c_2$ appropriately penalizes these types of histogram corrections. It also favors processing dots with large virtual line error first, followed by dots with large histogram error.

Virtual Line Error. When $e_{A,v}$ is large, no neighbor exists to form a virtual line from A with an orientation close to $\theta_{A,f}$. We correct this by repositioning a randomly chosen neighbor $n_{A,i}$ (e.g., C is chosen in Fig. 4). $n_{A,i}$ is moved with two constraints: (1) $\theta_{A,i} \in [\theta_{A,f} \pm 5]$ (i.e., the virtual line formed by A and $n_{A,i}$ is within 5° of the flow orientation), and (2) the length l of the virtual line is $l = \alpha r,\ \alpha \in (0,1]$. Recall that each virtual line contributes to its histogram bucket with weight $w = 1$, $\frac{2}{3}$, or $\frac{1}{3}$ depending on l. We use a probability distribution for α to favor l that produce $w = \frac{2}{3}$.

Histogram Error. Histogram error occurs when the peak of a histogram $\theta_{A,h}$ occurs away from the flow orientation $\theta_{A,f}$. To correct for this, we reorient virtual lines to move them from the histogram's current (incorrect) peak into a bucket near $\theta_{A,f}$. Specifically, we:

1. Randomly choose a virtual line CD from the histogram's current peak such that C is a neighbor of A, D is a neighbor of

C, and D is not a direct neighbor of A (i.e., D is a neighbor's neighbor of A).

2. Move D such that CD's orientation $\theta_{CD} \in [\theta_{A,f} \pm 5]$ (i.e., CD is either in the desired histogram bucket, or one away from that bucket).

This reduces the height of the histogram's current peak, and increases the height of the buckets around where we want the peak to occur. For example, in Fig. 5 we selected BH from the histogram's current peak, and moved H to change BH's orientation such that $\theta_{BH} \in (130^\circ, 140^\circ]$, the bucket that contains the actual flow orientation $\theta_{A,f}$.

Updating Error. A proposed move is accepted or rejected depending on the overall change in bivariate error it produces. Because Stevens's algorithm operates on local regions, there is no need to recalculate the error at every dot. If $n_{A,i}$ is moved, only its spatially local neighbors will be affected. To find these neighbors, we:

1. Identify $n_{A,i}$'s neighbors, and the neighbors of its neighbors at both its original and its new positions.
2. Union the two sets to obtain the combined set $D_{A,i}$ of all dots that would be affected by moving $n_{A,i}$.

We can then compute the total bivariate error for the dots in $D_{A,i}$ both before and after the proposed move. If bivariate error falls, we accept the move and re-insert the dots in $D_{A,i}$ back into the queue with their possibly new error values.

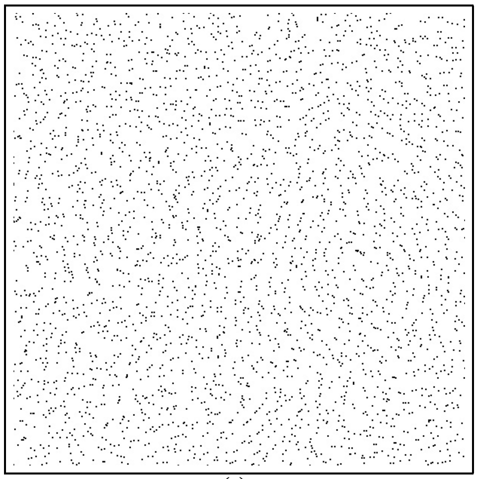
4.5 Dot Pattern Hierarchies

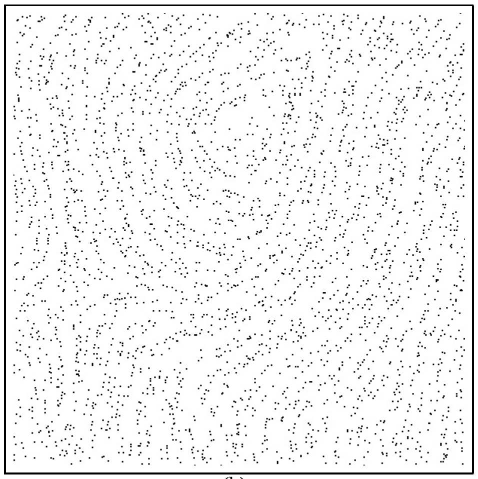
A separate issue we must address is how to manage flow fields that are too large to visualize in their entirety. Given a required dot-per-pixel density $\rho = 0.0085$ and a display with n pixels, we can seed the screen with at most $0.0085n$ dots. If we need one neighborhood of seven dots to properly visualize each flow sample, we can visualize at most $s = \frac{1}{7}(0.0085n) = 0.001214n$ samples (e.g., a typical 1280×1024 screen can visualize about $s = 1592$ samples). This problem is not unique to our algorithm. For any visualization technique, the desired level of detail (e.g., the number of samples or number of values per sample) is constrained by the physical properties of the display device. Various methods have been proposed to address this problem, for example, data summarization, multidimensional visualization algorithms, and focus+context techniques.

We chose to implement a hierarchical solution to allow flow fields of any size, while still maintaining the dot density needed to build our orientation patterns. Intuitively, our method works as follows. The flow field is interpolated to reduce the number of samples to a size that will fit on-screen. This represents a high-level overview of the dataset. The viewer can zoom in on different parts of the flow field to obtain more detail. Additional dots are added during the zoom to maintain the proper dot density, and to increase the level of detail being displayed. Zooming halts when the user reaches a level of detail that represents the original, unfiltered flow values.

We begin by halving the size of the dataset repeatedly until we reach a size that fits on-screen. Each reduction represents a *layer* which we will visualize with a dot pattern. Given a flow field with m samples, we generate layers of size m, $\frac{m}{2}$, $\ldots$, $\frac{m}{2^i}$ where layer i is the first layer to contain s or fewer samples, $i = \lceil log_2 \frac{m}{s} \rceil$.

Next, we build a dot pattern for each layer to visualize the flow orientations contained in its samples. We start with the topmost layer i, which contains the fewest samples, and work down to layer 0, which contains the original flow field values. In order to apply our algorithm to each layer, we need to know: (1) how many samples the layer contains, (2) how many "pixels" the layer contains,


(a)


(b)

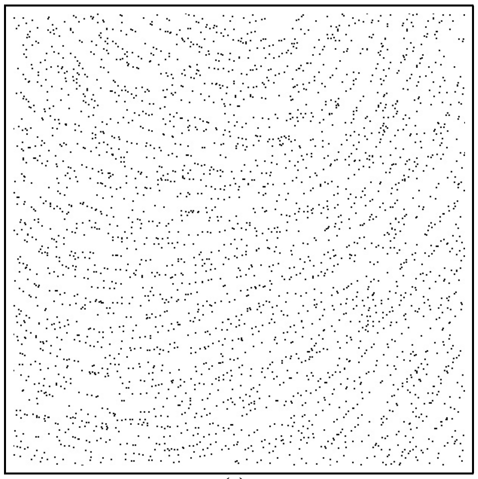

(c)

Figure 6: Supernova flow visualizations: (a) an overview showing the highest layer in the visualization, layer 7; (b) a zoom in on the center of the field in (a) down to layer 4; (c) a further zoom in on the center of the flow field down to the original unfiltered data at layer 0

and (3) how many dots to seed the layer with. As described above, layer i contains $\frac{m}{2^i}$ samples and $(\frac{7}{\rho})(\frac{m}{2^i})$ pixels[1], and is seeded with $\frac{7m}{2^i}$ dots. Based on these values, we populate each layer with dots as follows:

1. For the topmost layer i we apply our algorithm to position $\frac{7m}{2^i}$ dots.
2. For each lower layer j, $j = i-1, \ldots, 0$, we insert the $\frac{7m}{2^{(j+1)}}$ dots from layer $j+1$ and lock them so they cannot be moved, then insert an additional $\frac{7m}{2^{(j+1)}}$ dots and apply our algorithm to position them to visualize layer j's samples.

Once the dot patterns for each layer are built, we can visualize the flow data as follows. We begin by showing layer i; this represents an overview of the flow values at the highest zoom level. Viewers can pan and zoom in on different parts of the flow field. As they zoom in, the dots on layer i spread apart, reducing the dot density. Stevens stated that each neighborhood must contain at least three dots to ensure distinguishable orientations. As we approach the point where the average number of dots per neighborhood is 3.5, we fade in the next layer in our dot pattern hierarchy. This doubles the number of dots in the display (i.e., it returns the dot density to seven dots per neighborhood). It also doubles the number of samples being visualized. Because layer j contains all the dots in layer $j+1$, there is no visual discontinuity during this operation. New dots smoothly appear to fill in the empty spaces between the existing dots. This zoom and add continues until layer 0 is introduced. At this point zooming is halted, since layer 0 represents a visualization of the original data at the lowest zoom level. If viewers choose to zoom out, the same process is applied in reverse. Dots converge, and just before they exceed our desired density, the next highest layer is visualized and half the dots are faded out.

Our hierarchical approach allows us to visualize flow fields of any size. Large fields simply mean more layers, and more zooming required by a viewer to reach the bottom layer.

5 Practical Applications

One final issue we studied was the application of our theoretical model to real-world data. We are collaborating with a number of practitioners who want to visualized both real and simulated two-dimensional flow fields. We decided to visualize flow data provided by astrophysics collaborators. Ongoing research in their laboratory involves investigating various aspects of supernovas. Part of this effort includes simulating how a supernova collapses. We were provided with a number of 2D slices through a 4D volume (x,y,z and t) produced by their simulation. The astrophysicists were particularly interested in our algorithm's ability to visualize both high-level overviews and low-level detail in the slices. The animations they are currently generating do not include multiple levels of detail, so their visualizations normally represent a high-level overview of an entire slice, possibly at the expense of displaying important local detail.

Data points within each slice form a 500×500 regular grid of sample points composed of the attributes Δx, Δy, and *magnitude*. Δx and Δy were used to determine the orientation of the flow at each sample point. Given our requirement of seven dots per sample, we were tasked with arranging 1.75 million dots. With $\rho = 0.0085$, this would require a screen size of $(\frac{1750000}{\rho})^{\frac{1}{2}} = 14349$ pixels square.

[1]By definition, only the topmost layer contains a number of pixels that can fit on-screen; layers below that will always have more pixels than the display, however, these lower layers will only be shown when the viewer zooms in on the flow field. The lower layers can be seen as "virtual" displays, that is, what we would have shown on a higher resolution display device.

We instead chose a window size of 900×900 pixels capable of displaying about 983 sample points. This produced a total of eight layers in our dot pattern hierarchy to properly visualize the entire slice. A cutoff of 18° (10%) average perceptual error was used during construction of each layer's dot positions.

Fig. 6a shows a 2D supernova slice visualized at its lowest level of detail (layer 7). Even though the topmost layer contains only a high-level overview, interesting features like vortices, concurrent flow regions, and flow boundaries can be identified. Fig. 6b and 6c show the same slice visualized at layer 4 and layer 0 as the viewer zooms in to study an area of interest. Additional dots faded in from the lower layers reveal more detail about how the flow field behaves. In Fig. 6b a vortex is clearly visible. Fig. 6c shows a close-up of the details around the vortex.

Anecdotal feedback from our collaborators suggests they are excited about our technique, in particular, in the ability to interactively zoom in and out of the slice to see an overview of the flow patterns, and to probe areas of potential interest all the way down to the individual sample points.

5.1 Direction and Magnitude

Two vector properties are not included in our basic dot patterns: direction and magnitude. Although local dot configurations show the orientation of the flow field, they do not define which of the two possible directions the flow actually travels. The magnitude of the flow (i.e., vector length or velocity) is also missing in our initial visualizations. These omissions are not unique to our algorithm. For example, the initial forms of spot noise and LIC produced textures that did not include either direction or magnitude.

Numerous techniques have been proposed to add this missing information to an orientation visualization. We investigated two common methods: color integration and simulated motion. In both cases we used perceptual rules to control the distinguishability of the colors, motion directions, and motion velocities we displayed. This fits well with our overall goal of generating perceptually salient visualizations.

One very simple solution for adding velocity values to our visualizations is to integrate a color overlay with our dot patterns. For each dot we query the velocity of the flow at the dot's position, then color the dot to encode the scalar value. Specific colors are selected by mapping the range of possible velocities is to a semi-continuous color scale. Our color scale was selected to ensure that: (1) the luminance of every color was sufficiently high to make it visible on a black background (i.e., coloring the dots would not make some of them difficult to see), and (2) colors along the scale are perceptually balanced, that is, equal differences in velocity will produce color differences that are perceived to be roughly the same. Our selection technique combines a number of experimental results on the perception of color [Healey and Enns 1999]. A single loop spiraling up around the luminance axis is plotted near the boundary of our monitor's gamut of displayable colors in CIE LUV space. The path is subdivided into c named color regions (*i.e.*, a blue region, a green region, and so on). n colors can then be selected by choosing $\frac{n}{c}$ colors uniformly spaced along each of the c color regions. The result is a set of colors selected from a perceptually balanced color model, each with a roughly constant simultaneous contrast error, and chosen such that color distance and linear separation are constant within each named color region.

In Fig. 7a we use a perceptual luminance scale to represent flow velocity while zooming in on the supernova data. Darker dots represent low velocities, while brighter dots represent high velocities (see Fig. 8 for a color velocity example). The data being shown is identical to the data in Fig. 6b. The addition of luminance does not obscure the dot patterns and the perceived orientations they produce. At the same time, sharp differences in flow velocity are visi-

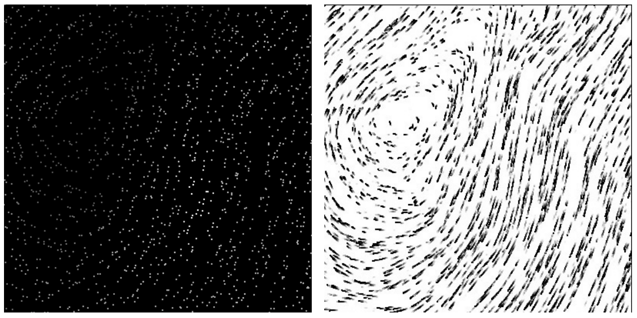

(a) (b)

Figure 7: Velocity and direction, data identical to Fig. 6b: (a) velocity encoded with luminance, darker for low velocity to brighter for high velocity, note identical dot pattern to Fig. 6b; (b) velocity and direction encoded with motion, shown as a composite of 20 consecutive animation frames, note long streamlines in areas of high velocity, shorter streamlines in low velocity regions

ble as variations in luminance.

A second method represents both direction and velocity by animating the dots (Fig. 7b). Each dot follows a linear path based on its perceived orientation, and moves with a speed proportional to the velocity at its location in the flow field. The dot is allowed to "walk" in a cycle within its local neighborhood, fading out as it reaches the neighborhood boundary, then fading back in at the opposite side. The continuous 360° around a dot are subdivided into 36 discrete "buckets", and the continuous velocity range is subdivided into five discrete ranges. These choices are based on results from experiments in our laboratory that studied how the low-level human visual system perceives properties of motion like flicker, direction, and velocity. Different motion directions require 10 to 20° of rotational difference to make them distinguishable from one another [Weigle et al. 2000]. Similarly, differences in velocity require at least 0.5° of subtended visual angle to be "seen" as distinct by our visual system. Given the local neighborhood size and maximum speed we allowed a dot to walk, this produced five usable velocities.

We were initially concerned that moving the dots would disrupt perceived orientations, since any change in a dot's position can potentially change histogram and virtual line error. In practice, however, we have not observed this problem. We believe this partly because consistent changes in a local neighborhood produce only small disruptions in the perceived orientation pattern, and partly because a dot's motion reinforces its perceived orientation, compensating for any additional error that may be introduced.

6 Conclusions

This paper describes a new technique for visualizing 2D flow data. Our method applies a model of how the low-level visual system perceives orientation to position a collection dots to visualize flow orientations in an underlying flow field. Our method offers potential advantages over existing flow visualization algorithms: (1) the use of perception to build flow patterns with orientations that are rapidly and accurately distinguishable; (2) an economy of dots so additional data can be encoded in the display; and (3) the ability to support large flow fields, and to represent related properties like direction and velocity. Our images, together with anecdotal feedback from our collaborators, suggests that Stevens's dot patterns can successfully visualize flow data at multiple levels of detail.

We are investigating a number future research issues. First, we plan to compare Stevens dot patterns to other 2D flow visualization algorithms (e.g., LIC), possibly using a method similar to Laidlaw's [Laidlaw et al. 2005]. This will identify the strengths and limitations of our technique, allow using to plan future research to improve the method. A second study focuses on computational complexity. Currently, our algorithm requires preprocessing time to generate each layer. We want to improve performance, hopefully to a point where the algorithm can run interactively. This is necessary to visualize real-time data. We also hope to extend the dot patterns to 3D flow data. This is complicated by the need to differentiate dot depth in a 2D projected image, and by the lack of detailed knowledge about how we perceive orientation in a 3D dot pattern. Findings from our laboratory on identifying orientation in 3D glyphs may form a starting point for this work. Finally, we are working with colleagues to try to combine Stevens dot patterns with existing multidimensional visualization techniques. The goal is a method to visualize multi-attribute flow fields.

References

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging vector fields using line integral convolution. In *SIGGRAPH 93 Conference Proceedings*, J. T. Kajiya, Ed., 263–270.

DE LEEUW, W. C., AND VAN LIERE, R. 1997. Spotting structure in complex time dependent flow. In *Scientific Visualization*, H. Hagen, G. M. Nielson, and F. H. Post, Eds. Springer-Verlag, New York, New York, 286–295.

GLASS, L., AND PEREZ, R. 1973. Perception of random dot interference patterns. *Nature 246*, 360–362.

GLASS, L. 1969. Moire effect from random dots. *Nature 243*, 578–580.

HEALEY, C. G., AND ENNS, J. T. 1999. Large datasets at a glance: Combining textures and colors in scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 2, 145–167.

HEGE, H., AND STALLING, D. 1998. Fast LIC with piecewise polynomial filter kernals. In *Mathematical Visualization–Algorithms and Applications*. Springer-Verlag, New York, New York, 295–314.

KIRBY, R. M., MARMANIS, H., AND LAIDLAW, D. H. 1999. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *Proceedings Visualization '99*, 333–340.

LAIDLAW, D. H., KIRBY, R. M., JACKSON, C. D., DAVIDSON, J. S., MILLER, T. S., DA SILVA, M., WARREN, W. H., AND TARR, M. J. 2005. Comparing 2D vector field visualization methods: A user study. *IEEE Transactions on Visualization and Computer Graphics 11*, 1, 59–70.

SHEN, H. W., AND KAO, D. L. 1997. UFLIC: A line integral convolution algorithm for visualizing unsteady flows. In *Proceedings Visualization '97*, 317–323.

STEVENS, K. A. 1978. Computation of locally parallel structure. *Biological Cybernetics 29*, 19–28.

TURK, G., AND BANKS, D. 1996. Image-guided streamline placement. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., 453–460.

VAN WIJK, J. J. 1991. Spot noise – texture synthesis for data visualization. In *SIGGRAPH 91 Conference Proceedings*, T. Sederberg, Ed., 263–272.

VAN WIJK, J. J. 2002. Image based flow visualization. In *SIGGRAPH 2002 Conference Proceedings*, J. Hughes, Ed., 745–754.

WEIGLE, C., EMIGH, W. G., LIU, G., TAYLOR, R. M., ENNS, J. T., AND HEALEY, C. G. 2000. Oriented texture slivers: A technique for local value estimation of multiple scalar fields. In *Proceedings Graphics Interface 2000*, 163–170.

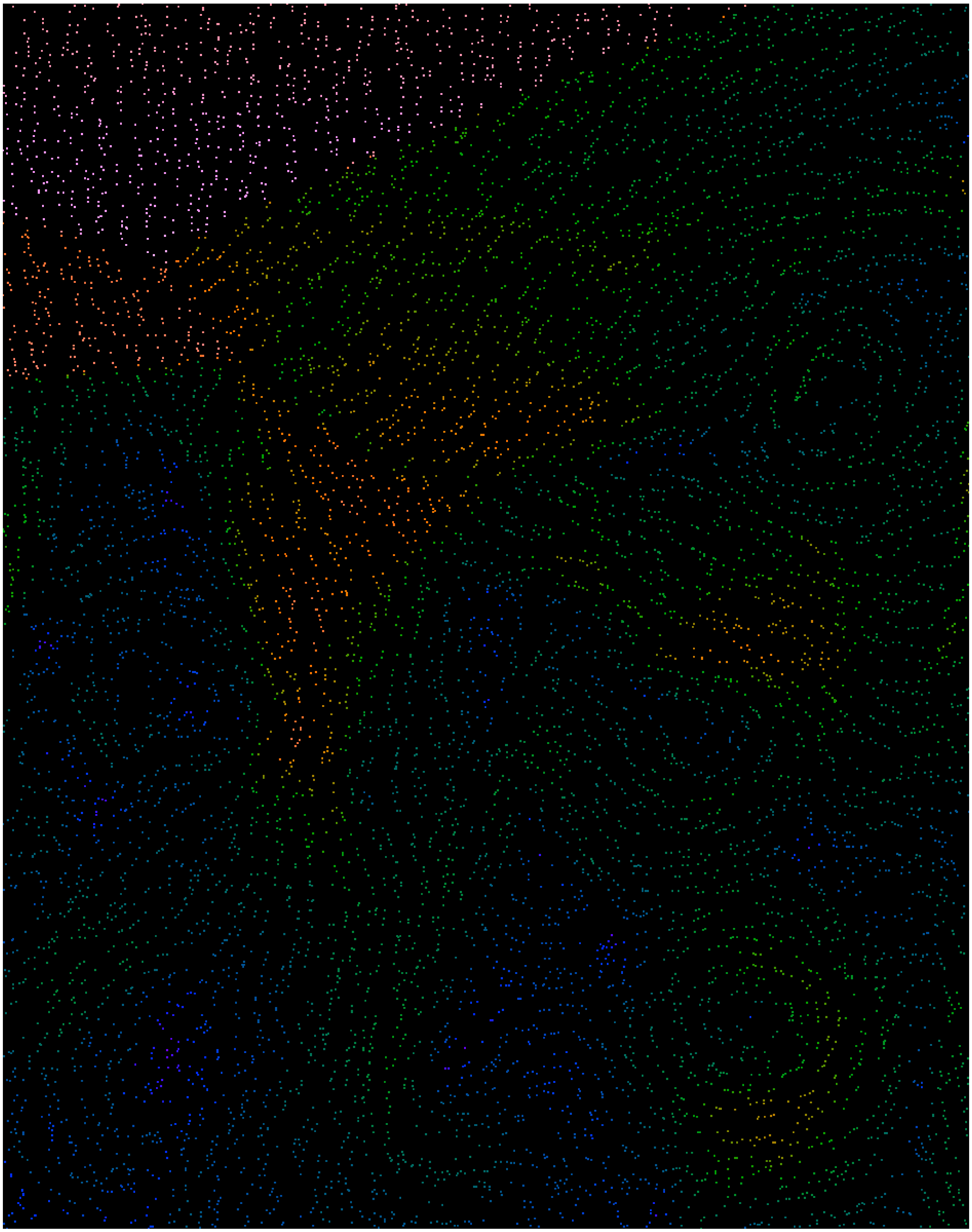


Figure 8: Velocity and direction within a simulated supernova collapse, velocity encoded with color, dark blues and greens for low velocity to bright oranges and pinks for high velocity