Eurographics Conference on Visualization (EuroVis) 2013
B. Preim, P. Rheingans, and H. Theisel
(Guest Editors)

Volume 32 (2013), Number 3

Synthetic Brainbows

Y. Wan[1] and H. Otsuna[2] and C. Hansen[1]

[1]Scientific Computing and Imaging Institute, University of Utah, USA
[2]Department of Neurobiology and Anatomy, University of Utah, USA

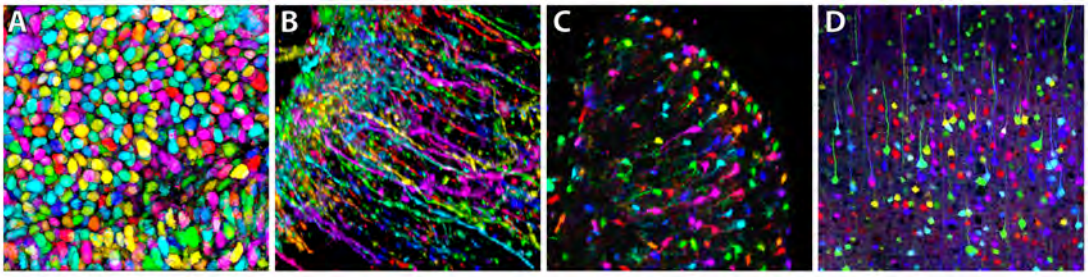

Figure 1: *Results from our synthetic Brainbow technique and a true Brainbow image. A: Cells in the eye of a zebrafish embryo. B: Neurons in a Drosophila brain. C: Eye of a Drosophila. D: The cerebral cortex of a mouse (Confocal image by Tamily Weissman. Mouse by Jean Livet and Ryan Draft. Image source: http://www.conncoll.edu/ccacad/zimmer/GFP-ww/cooluses0.html). A, B, C are single-channel confocal scans processed with our synthetic Brainbow technique, in comparison with the true Brainbow image in D.*

Abstract

Brainbow is a genetic engineering technique that randomly colorizes cells. Biological samples processed with this technique and imaged with confocal microscopy have distinctive colors for individual cells. Complex cellular structures can then be easily visualized. However, the complexity of the Brainbow technique limits its applications. In practice, most confocal microscopy scans use different florescence staining with typically at most three distinct cellular structures. These structures are often packed and obscure each other in rendered images making analysis difficult. In this paper, we leverage a process known as GPU framebuffer feedback loops to synthesize Brainbow-like images. In addition, we incorporate ID shuffling and Monte-Carlo sampling into our technique, so that it can be applied to single-channel confocal microscopy data. The synthesized Brainbow images are presented to domain experts with positive feedback. A user survey demonstrates that our synthetic Brainbow technique improves visualizations of volume data with complex structures for biologists.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—; J.3 [Life and Medical Sciences]: Biology and genetics—;

1. Introduction

Brainbow [LWK*07] is a genetic engineering technique that randomly colorizes cells. Biological samples processed with this technique and imaged with confocal microscopy have distinctive colors for individual cells. It is useful for disambiguating visual clutter in confocal data, identifying complex cellular structures, and for cell tracking. However, the application of the Brainbow technique on a certain species of animals requires complex transgenic manipulations. In practice, most confocal microscopy scans use different antibody staining with typically at most three distinct cellular structures. These structures are often packed and obscure each other in rendered images making analysis

difficult. The problem is commonly addressed through segmentation. Accurate segmentation of confocal microscopy data, which are typically full of fine details, depends greatly on users' prior knowledge of the data. Such knowledge does not only come from experience, but also is more and more importantly from visualizations of the data. Visualizing confocal microscopy data requires techniques that on the one hand are interactive and preserve the fine details on the other. Inspired by the Brainbow technique, we explore techniques that randomly colorize single-channel confocal microscopy data. The outcome of our random colorization technique assumes similar appearance of Brainbows. Adjacent complex structures can then be clearly visualized by color variations. Our synthetic Brainbow technique leverages a process known as GPU framebuffer feedback loops, which is a random process in the massive parallel computing environment of GPUs. In addition, we incorporate ID shuffling and Monte-Carlo sampling into the technique. The random colorization in our synthesized Brainbow images respects structural information and preserves fine details. The results are presented to domain experts with positive feedback. A user survey demonstrates that our synthetic Brainbow technique improves visualization of volume data with complex structures for biologists.

2. Background and Related Work

Randomness is an inherent character accompanying all natural processes. Researchers in biology have taken advantages of randomness. We are particularly interested in one recent technique in life science: Brainbow. In [LWK*07], Livet et al. described a series of strategies to randomly express fluorescent proteins in individual cells of mouse nervous system. They exploited the advantages of the widely used Cre/lox recombination system [BD04], which is able to turn on or off the expression of one or several different fluorescent proteins in a gene sequence. For different cells that are genetically modified to work with this technique, different combinations of the fluorescent proteins can occur. This is because the Cre/lox recombination system randomly chooses the gene expressions for recombination. The end result is that different cells, despite the same type, are fluorescently stained with different colors. This technique is useful to visualize and distinguish detailed structures in the nervous system, where cells are packed and can be touching. However, the application of the Brainbow technique on a certain species of animals is limited because of complex transgenic manipulations. Disambiguation of cellular structures in single-channel datasets are commonly addressed by segmentation techniques (e.g. Cohen et al. [CRT94]), which usually change the appearance of the original data and may be undesired for visualization purposes. Inspired by Brainbow, we would like to use computational techniques to randomly colorize confocal microscopy data processed with common antibody staining. By generating Brainbow-like results where different structures are distinctively colored, our proposed technique is an improvement to the visualization of single-channel confocal microscopy data.

Our synthetic Brainbow technique leverages a process known as GPU framebuffer feedback loops. This process reads and writes the same framebuffer by multiple rendering or computing threads on GPU. If the output value of one pixel is dependent of its neighbors' values, it essentially creates race conditions among different threads. Without locking or synchronizing of the threads, the results become nondeterministic. Experienced graphics program developers avoid the nondeterministic behavior of GPU framebuffer feedback loops by framebuffer Ping-Pong, which is a technique using two framebuffers for reading and writing, thus synchronizing different threads. In fact, setting up a framebuffer feedback loop by binding the same framebuffer to a shader's (or computing kernel's) input and output is not considered an error by graphics hardware specifications. Developers are simply warned against doing so because the results are "undefined" [SA12]. However, considering that a framebuffer feedback loop is computationally more efficient by saving half the memory and using fewer context switches, we do believe it deserves a closer examination. In our research, we find that the nondeterministic behavior of given graphics hardware induced by asynchronism can be statistically tested and determined, which is discussed in Section 3.

Applications of GPU framebuffer feedback loops are rare in previous work, due to the fact that deterministic results are generally desired in computations with GPUs, such as filtering in image processing and equation solving in simulations. However, its theoretical development has long preceded the appearance of GPUs. An iterative computational model of GPUs is equivalent to a cellular automaton. The framebuffer can be thought as a grid of the cellular automaton with its pixels as the cells. The shader or computing kernel provides the rules for updating the states of the cells. The study of cellular automata dates back to the early history of computer science, including work of Ulam and Neumann [vN66]. Different types of cellular automata have been extensively studied through the later development of computer technology. Cellular automata have been proposed as computational models for simulations in physics [Mar84] [RK88], material science [Bia94] [SBYR*91], and biology [HG93] [NM92]. They have also been extensively used in image segmentation algorithms [VK05] [KP08] [KP10] [KSH10] [GYXS11] [LCH*12]. However, most research

focused on synchronous cellular automata, where the state of every cell is updated together. Using framebuffer Ping-Pong in an iterative computational model is an example of a synchronous cellular automaton. In contrast, a framebuffer feedback loop should update individual cells independently, and the new state of a cell affects the calculation of states in neighboring cells, thus an asynchronous cellular automaton. Asynchronous cellular automata are generally less studied due to their nondeterministic behaviors. A lot of effort has been spent in recent research on asynchronous cellular automata to find efficient ways of computing deterministically without global synchronization. The research is mostly based on chaotic relaxation (Chazen and Miranker [CM69]), which described necessary and sufficient conditions for an asynchronous and chaotic process to converge. Baudet [Bau78] presented a class of asynchronous iterative methods for solving a system of equations. Adachi et al. [AL04] presented an asynchronously updating cellular automaton that conducts computation without relying on a simulated global synchronization mechanism. Galilée et al. [GMRC07] proposed a joint algorithm-architecture for computing watershed segmentation. Their algorithm is programmed as a set of concurrent communicating iterative programs that are efficiently mapped onto an asynchronous parallel architecture. Venkatasubramanian and Vuduc [VV09] described GPU implementations of Jacobi's iterative method for the 2D Poisson equation. Their implementations include a "wild" asynchronous example, which removed synchronization between iterations. They have shown that the "wild" asynchronous implementation on GPU has 1.2-2.5x speedups against best synchronous implementation, thanks to highly efficient memory bandwidth utilization. For GPU implementations of connected component labeling, Oliveira and Lotufo [OL10] discussed an ID merging method using asynchronous automata and presented an improved algorithm that included local and global merging stages. They also reported their method achieved 5-10x speedup in relation to Stephano-Bulgarelli's [dSB99] serial algorithm. We regard the previously presented connected component labeling methods as primitive forms of more sophisticated colorization. Different from previous research, we leverage the randomness and use a computational stochastic process to simulate the results from a biological stochastic process: Brainbow.

3. Randomness in a GPU Framebuffer Feedback Loop

We would like to first examine the behavior of GPU framebuffer feedback loops. In order to avoid unnecessary complexity, we restrict the investigation to integer textures and turn off texture filtering. The behavior of GPU framebuffer feedback loops can be studied with cellular automaton models. This is inspired by Hawick et al. [HLP10] and Oliveira and Lotufo [OL10], whose work used cellular automaton models for connected component labeling. Specifically, we are interested in a cellular automaton described by Algorithm 1. We use a 1D example to demonstrate the reason that we chose this particular cellular automaton for examination of the random behavior of GPU framebuffer feedback loops. In Figure 2, a 1D cellular automaton has eight cells. At its initial state, each cell is assigned an integer ID, which is in ascending order. We examine three different methods of updating the cells. First, the cells are updated synchronously, which means we need an extra buffer to save the intermediate results. In this case, the order of how the cells are updated makes no difference to the results. It requires seven iterations to converge to all the same ID. Then, we update the cells asynchronously, i.e. reading and writing IDs without using an extra buffer. Since the order of how the cells are updated can be random and influence the result, we examine two extreme cases among all the combinations of update orders. The second method updates the cells asynchronously in order from left to right. The result looks exactly the same as when the cells are updated synchronously. Lastly, we update the cells asynchronously, but in reverse order. It only requires one iteration for the maximum ID to propagate. We are able to make several observations from this example. First, if we color-map the IDs, we can see a fixed stripe pattern "marching" through the grid when the IDs are ordered and the updates are synchronous. Second, when the updates are not synchronous, the "marching" pattern is the same as synchronous updates only if the update order is the same as the ID order, and is disturbed otherwise. We are able to detect such disorderliness by comparing the result from asynchronous update to that of synchronous update. Suppose that the occurrences of ordered and reverse ordered asynchronous updates are about the same, the "marching" IDs are so disturbed that we should not observe regular patterns. Third, asynchronous update can potentially accelerate ID propagation.

Algorithm 1 Basic ID merging

```
For each cell
    A unique ID is assigned as the initial state;
For each iteration
    For each cell
        The cell's state is replaced by the maximum
        ID within its neighborhood;
```

We extend the above idea and use it to detect the

Figure 2: *Comparison of synchronous and asynchronous updates of a 1D cellular automaton. The illustration shows only the first iteration in detailed steps. In the synchronous case, the original buffer is in red and the extra buffer for intermediate results is in green. Values are updated according to the maxima within the moving window (in blue), indicated by the yellow arrows. In the asynchronous cases, there is no extra buffer, so updates are immediate. The updated values in each step when the window moves are in dark red. We can repeat the iteration for the first two cases until all cells are updated to the maximum ID. Their results look exactly the same. The last case has already converged after the first iteration.*

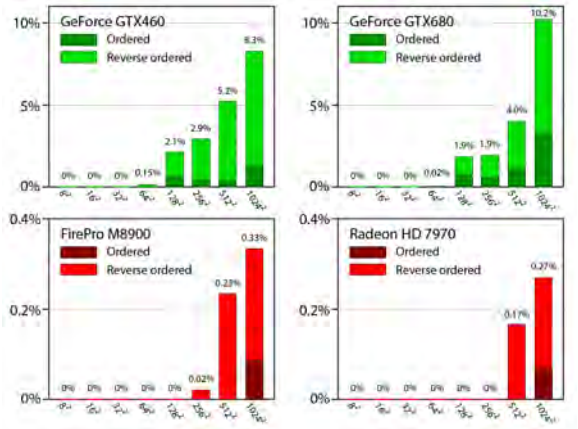

Figure 3: *Test results of occurrence of asynchronous updates for four graphics cards: nVidia GeForce GTX 460, GTX 680, AMD FirePro M8900 and Radeon HD 7970. The horizontal axes of all plots are texture sizes tested. The vertical axes are the occurrence of asynchronous updates in percentage.*

occurrence of asynchronous updates in GPU framebuffer feedback loops, which we assume to be the sole cause of randomness in the process. For a given graphics card and a framebuffer texture of given size, we first assign IDs in ascending order. In 2D, this can be row-first or column-first, which does not influence the result. We run Algorithm 1 once in a framebuffer feedback loop and compare the outcome with that from a synchronous update. In the asynchronous result, we count the number of pixels that have different IDs than in the synchronous result. These pixels have different IDs because the orders of asynchronous updates are *against* the ID order. To count asynchronous updates of the reverse directions, we then do the same but with reverse ordered IDs. Both experiments are repeated for ten times. We then calculate the average percentages of asynchronous updates for both ordered and reverse ordered IDs. The two average values are added to estimate the total occurrence of asynchronous updates. We experimented with four graphics cards and each with eight different texture sizes. The results are illustrated in Figure 3. The tested graphics cards should be representative for current main-stream models from the two major GPU manufacturers: AMD and nVidia. The tested results reveal important characteristics of framebuffer feedback loops. Firstly, contrary to our initial speculation, framebuffer feedback loops are not entirely asynchronous. This is due to texture caching. Framebuffer feedback loop exhibits similar behavior to framebuffer Ping-Pong for small texture sizes (no asynchronous updates), as texture cache and graphics memory are working as two buffers for reading and writing. Secondly, occurrence of asynchronous updates increases as texture size increases. However, even for asynchronous updates, GPU threads tend to access memory with ascending order. This can be seen from the much higher occurrence rate of asynchronous updates when the IDs are in reverse order. Thirdly, GPUs from different manufacturers (AMD vs. nVidia) exhibit different occurrence rates of asynchronous updates. However, GPUs from the same manufacturer have similar results, even if they are in different series (for example, GeForce 400 series vs. 600 series). In conclusion, for specific graphics hardware, the behavior of framebuffer feedback loop is nondeterministic but predictable within a certain range, which is the result of a hybrid of synchronous and asynchronous updates. It is worth mentioning that the authors did the tests within their available resources. The behavior of framebuffer feedback loop may vary greatly for older or future hardware, or integrated GPUs. We continue our discussion in the following sections with regard to the tested graphics hardware. Framebuffer feedback loops are always used in the subsequent discussions.

4. Synthetic Brainbows

4.1. ID Shuffling

In fact, Algorithm 1 was discussed in the work of Hawick et al. [HLP10] and that of Oliveira and Lotufo [OL10] as a local merging step in their GPU implementations of

connected component labeling. If we introduce a binary mask into an iterative process, assigning and propagating IDs only within the masked regions, it converges to the labeled connected components of the mask. With framebuffer feedback loops, the result still converges and is deterministic, since cell values are monotonically increasing and upper bounded within each connected component. This also can be considered as a simple case of chaotic relaxation [CM69]. However, if we focus on the process itself rather than its convergence, we should be able to observe one problem when IDs are ordered as initial states. For example, a spiral is used as the binary mask in Figure 4. If we consider the spiral as a complex structure, the visualization task here is to decompose the structure into simpler shapes so that each component as well as the spatial relationships among components can be more easily studied. Connected component labeling, which considers the complex structure as one component, does not fulfill the requirement. A simple solution is to stop the iterative process of local ID merging at fixed iterations. This generates stochastic patterns due to asynchronous updates. But also, because the tested graphics cards largely exhibit synchronous behavior, as discussed in Section 3, local ID merging generates many small sub-regions (Figure 4 A and B). The high frequency patterns in these un-merged regions can be visually distractive. To compensate for this and generate fewer sub-regions after certain iterations, we propose ID shuffling. ID shuffling does not randomly change the order of IDs. Instead, IDs are meant to be *evenly* distributed. This is formally defined as maximizing the grid distance between any ordered pair of adjacent IDs. Intuitively, it can be understood as the process of placing IDs, from large to small, onto an empty grid. Wherever an ID is placed on the grid, the next smaller ID is placed so that the pair can be as far apart as possible. The result is that local merging is centered at local maxima and small sub-regions are quickly merged into larger regions (Figure 4 C). For easy implementation, we first consider an ID shuffling algorithm for grids of size 2^n. The shuffled IDs are easily generated from their grid coordinates, as the IDs are spatially placed according to a binary (or quad- for 2D, or oct- for 3D, etc.) encoding tree. We first introduce the shuffling algorithm for a 1D grid (Algorithm 2), which is illustrated in Figure 5. In the algorithm, the function reverse_bit() reverses the order of the binary code of the input integer. The subtraction step at the end is only for indices starting from 0. We want to exclude 0 from valid IDs, since it is used as a mask value. It is equivalent to increasing the reversed index value by 1.

Shuffling algorithms for higher dimensional grids are extensions of Algorithm 2, as shown below. In Algorithm 3, the function interleave_bit() combines the bits of its multiple inputs in an interleaving fashion. For example, in a 3D grid, we have calculated I'_x, I'_y, and I'_z for one cell. The first bit of the function's output I' is the first bit of I'_x, and then the second bit of I' is the first bit of I'_y, and then the first bit of I'_z, and so on. For a grid of an arbitrary size, it is considered as a sub-region of a larger grid of size 2^n. Then IDs of any

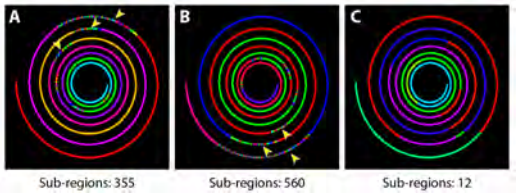

Figure 4: *A 512×512 binary image of a spiral is used as a mask for Algorithm 1. After 512 iterations, different ID orderings exhibit different patterns when IDs are color-mapped. A: IDs are in ascending order. B: IDs are in descending order. C: IDs are shuffled with Algorithm 3. The numbers of remaining sub-regions are shown below the images. Yellow arrowheads point to regions where high amount of un-merged sub-regions are present, due to the largely synchronous behavior of the graphics card. With shuffled IDs, the algorithm is able to generate fewer sub-regions with the same number of iterations. The tests are done on an AMD Radeon HD 7970 graphics card, which exhibits the least asynchronous behavior in Figure 3. There are fewer remaining sub-regions when a more asynchronous graphics card is used, e.g. nVidia cards. However, the difference between ordered and shuffled ID orderings is still quite large.*

Algorithm 2 ID shuffling for a 1D grid

Define N = the number of cells of the grid and
$N = 2^n, (n \in Z)$;
For each cell
    I = the cell's grid index;
    I' = reverse_bit(I);
    ID = N - I';

Algorithm 3 ID shuffling for a high dimensional grid

Define $\{N_i\}$ = the number of cells in each dimension and $N_i = 2^{n_i}, (n_i \in Z)$;
Define $N = \prod(N_i)$;
For each cell
    $\{I_i\}$ = the cell's grid index in each dimension;
    $\{I'_i\}$ = {reverse_bit(I_i)};
    I' = interleave_bit($\{I'_i\}$);
    ID = N - I';

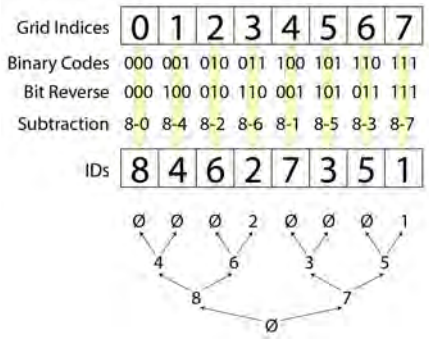

Figure 5: *ID shuffling for an 8-cell 1D grid. The binary code of each cell index is reversed and then subtracted from the total cell number. The IDs are placed in the order of visiting the binary tree shown below, with depth-first traversal. In the tree, larger IDs are on nodes of higher levels.*

grid can be calculated using Algorithm 3. Our algorithm is not the unique way to shuffle IDs. Equivalent algorithms can be derived for example by swapping nodes on the same level of the binary tree in Figure 5. However, our algorithm is suitable for parallel evaluation, as each cell's ID is uniquely defined by its indices. For repetitive evaluations of different grids, IDs can be pre-calculated and reused. Furthermore, in addition to asynchronous computing models, ID shuffling is potentially an improvement to existing connected component labeling algorithms, since it accelerates the local merging step. However, for visualization purposes, our ID merging algorithm with shuffling can only be applied on binary data. The decomposition of complex structures still seems to be arbitrary. In order to apply it on grayscale data and for the colorization to follow structural information, finer control over the local merging process is necessary, which is discussed next.

4.2. Monte-Carlo Sampling

Our goal in this paper is to apply ID merging on a single channel of confocal microscopy data, and generate Brainbow-like images. We want the colorization process to be applied directly as we do not want to rely on pre-segmented results. Since we want to have unique IDs for individual regions, one difficulty of applying the colorization described in Section 4.1 to grayscale data is that IDs cannot be simply scaled according to scalar intensities. However, we can control the merging speed of IDs based on scalar intensities. This is achieved through temporal Monte-Carlo sampling. The algorithm for colorizing a grayscale volume is listed below. In Algorithm 4, M is a scalar value calculated from the original scalar volume. It measures the features that we use to control the merging speed. For example, the stopping function in a standard anisotropic diffusion [PM90] can be used as the measure of homogeneity. We can use this measure if we want the merging to be faster in homogeneous regions and slower at edges (less homogeneous). The value N can be seen as a pseudo-random number generated from a 4D (3D plus time or iteration) noise function [Gus05]. If the value of M is higher, there is also a higher chance that the ID is merged, and vice versa.

Algorithm 4 Colorization of a grayscale volume

```
Define S the scalar volume of original data;
Define ID the ID volume generated by Algorithm 3;
For each cell in ID
    M = the measure of features in S;
    N = a sample from a 4D noise function;
    If M > N
        The cell's ID is replaced by the maximum ID
        within its neighborhood;
```

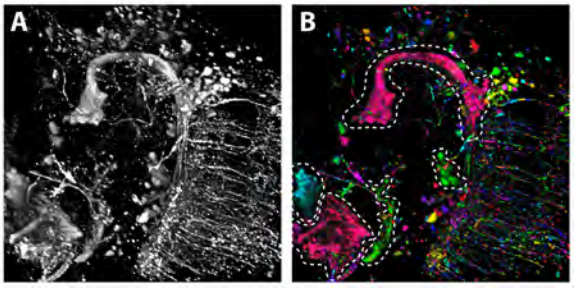

Figure 6: *Colorization of a confocal scan of a Drosophila brain (512×512×85×8bit). A: The volume rendering of the original dataset. B: The volume rendering of the colorized dataset. Dotted outlines indicate large and homogeneous structures. It took 200 iterations to generate the result. Generating the result took around 1 second on an AMD Radeon HD 7970.*

We first experimented with a common measure of edges, which is defined by the gradient magnitude of the intensity value S:

$$M = e^{-\frac{|\nabla(S)|^2}{\sigma^2}} \quad (1)$$

Figure 6 shows the result of applying Algorithm 4 on a confocal scan (512×512×85×8bit) of a *Drosophila* brain. The nervous system of the *Drosophila* brain has complex structures, where important structures can easily be obscured. We ran 200 iterations of ID merging and examined the patterns generated from Algorithm 4. Notice that the ordered ID sequence was repeatedly mapped to a palette of bright colors that resemble fluorescent markers. Because of ID shuffling (Algorithm 3), the

colors were spatially shuffled too. The colored volume was then modulated by the original scalar intensities. The number of iterations was chosen to generate the desired color variations. Comparing the colorized volume (Figure 6 B) with the original volume (Figure 6 A), we can observe that large and homogeneous structures are emphasized, which are indicated by the dotted outlines in Figure 6 B. Small structures are also distinctively colored. This is helpful when one structure is obstructing another. Their spatial relationship becomes clear, as they are colored differently. An apparent drawback of using only gradient magnitude as the measure for edges is that faintly connected structures, such as the fibers in the lower right region of Figure 6, are colored differently even for the same branch. To generate the result in Figure 6, σ in Equation 1 was set to 0.5. If we further increase its value, more structures are merged and colored the same, which reduces the resolving capability. This is illustrated in Figure 7 by an example of two idealized touching biological cells, where we have to increase the σ value in order to merge the surrounding IDs. For just two cells, we can easily stop the merging process when the two groups of IDs join at the boundary, similar to Figure 7 D. This becomes impractical for a large amount of cells or complex structures, since the iteration number is a global parameter and cannot be tuned for all structures of different sizes.

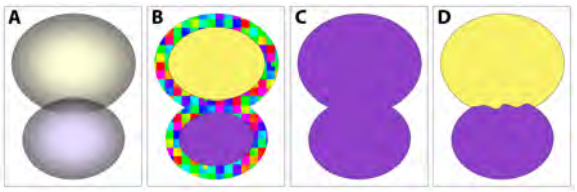

Figure 7: *An illustrated example of two idealized touching biological cells. A: The original data. Cells have high intensity and low gradient magnitude at their centers and low intensity but high gradient magnitude at borders. B: Algorithm 4 is used with the measure in Equation 1. When the σ value in Equation 1 is low, the centers of the cells are colored differently. However, they are surrounded by a cloud of various colors (IDs), which obscures the content inside. C: When we increase the σ value, the two cells are fused together. D: When we introduce a size constraint, the two cells can be colored as desired.*

A great advantage of using the ID merging process is that the size of each individual structure having the same ID can be retrieved by counting the number of cells with the same ID. The size of each structure is then used to control the merging process. The problem of two touching cells can be solved using a two-pass method. In the first pass, a low σ value is used and the result looks like Figure 7 B. Then we count the size of each structure having the same ID, similar to calculating component sizes in connected component analysis. In the second pass, we use a high σ value and set a constraint on component size, which is described in Algorithm 5.

Algorithm 5 Synthetic Brainbow (colorization of a grayscale volume with size constraint)

```
Define S the scalar volume of original data;
Define ID the ID volume generated by Algorithm 3;
Define B a Boolean volume whose voxels represent
if they are in a component with size over a threshold;
For each cell in ID
    M = the measure of features in S;
    N = a sample from a 4D noise function;
    If M > N and !B
        The cell's ID is replaced by the maximum ID
        within its neighborhood;
```

If the size constraint is set to lower than the size of the smaller cell, the surrounding IDs in Figure 7 B are merged into the two cells. The IDs from the two cells, however, cannot be merged into one another, because of the size constraint. The result should look like Figure 7 D. Notice that although the size constraint is a global parameter, it is used to safely merge IDs in noisy regions. So it is usually set at a relatively small value. Larger structures are merged by adjusting the measure of features and iteration numbers.

5. Results and User Survey

Using the above method, we generated synthetic Brainbows for three single-channel confocal scans. We were able to adjust the number of iterations, the σ value in Equation 1, and the size constraint during the ID merging process. For each scan, we performed several experiments with different parameter combinations until reaching a satisfactory result. The results are listed in Figures 8 to 10. All listed results are from an AMD Radeon HD 7970 graphics card. We also generated synthetic Brainbow images with other graphics cards tested in Section 3. Because occurrence of asynchronous updates is about 5% at the highest for 512x512 textures, only limited variation can be observed from these results, which are included in the supplementary material.

For most cases, e.g. Figure 8 and 9, our synthetic Brainbow technique does not alter information from original datasets. It enhances visualization by randomly applying colors to different structures. We chose bright colors that resemble fluorescent markers used in biology research, so that structures are not accidentally emphasized/de-emphasized because of color variance.

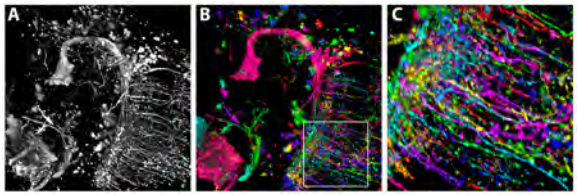

Figure 8: *Synthesized Brainbow of the same confocal scan of a Drosophila brain (512×512×85×8bit) in Figure 6. A: The volume rendering of the original dataset. B: Rendering of the colorized result generated with size constraint. C: A close-up of the fibers. Individual fibers are connected and different fibers are colored differently. In the first pass of ID merging, iteration number is set to 200, and σ is set to 0.5. In the second pass of ID merging, iteration number is set to 200, σ to 1.0, and size constraint is set to 100 voxels. The colorization process took 2.81 seconds on an AMD Radeon HD 7970.*

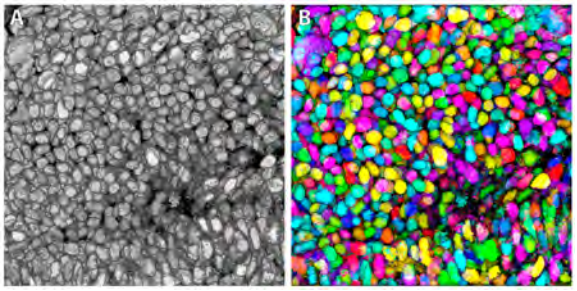

Figure 9: *Synthesized Brainbow of a confocal scan of an eye of a zebrafish embryo (512×512×33×8bit). A: The volume rendering of the original dataset. B: The volume rendering of the colorized result. In the original scan, many structures are fused together, which are better discriminated in the colorized result. The colorization took two passes. In the first pass, iteration number is set to 50, and σ is set to 0.35. In the second pass, iteration number is set to 300, σ to 1.0, and size constraint is set to 250. The colorization process took 1.34 seconds on an AMD Radeon HD 7970.*

For noisy datasets, e.g. Figure 10, we suppressed noise with MIP. We believe such colorization can help visualizing complex structures in biology research. Since the colorization process takes relatively short time, it can be integrated into a biologist's visualization workflow. Another prospective use is when a group of biologists are looking at scans, instead of pointing on the datasets and referring structures as "this" or "that", specific color names can be used. However, this can only be validated when the synthetic Brainbows are used in practice.

Since biologists are the potential users of our technique, we created an online survey and sent its link

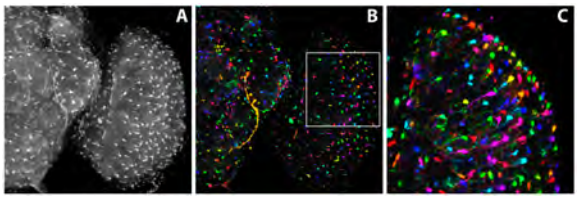

Figure 10: *Synthesized Brainbow of a confocal scan of a Drosophila brain (512×512×115×8bit). A: The volume rendering of the original dataset. This is a noisy dataset. B: The volume rendering of the colorized result. The colored volume is rendered with maximum intensity projection (MIP) [WMLK89] plus a shading overlay, as described by Wan et al. [WOCH12], in order to see the colored structures clearly. C: A close-up of the cells. The colorization took two passes. In the first pass, iteration number is set to 200, and σ is set to 0.35. In the second pass, iteration number is set to 10, σ to 1.0, and size constraint is set to 50. This 10-iteration process is then repeated five times in the second pass. This is because the dataset is noisy and we need to look at the result and decide if more iterations are necessary. The colorization process took 1.98 seconds on an AMD Radeon HD 7970, excluding the time for manual parameter adjustment.*

to biologists that are experts in confocal microscopy. The participants should be familiar with the Brainbow technique, but they may not necessarily be working with the technique. There were two parts in the survey. In each question of the first part, an image was shown and participants were asked how likely the image was generated by the Brainbow technique. There were eight images: four generated with our technique, two true Brainbow images, and two images generated by first anisotropic diffusion [PM90] and then thresholding plus connected component labeling. The images were shown in random order. In the second part, we revealed the images that were generated by our technique and showed their original renderings as in Figure 8, 9 and 10. We asked the participants how likely they would use this technique for visualization enhancement in their research. We received answers from 16 participants, some of which also left comments. The answers to the first part are plotted in Figure 11. Most participants agreed that our technique was able to generate results similar to Brainbow. Furthermore, 70% of the participants showed great interest in using our technique. Details of the survey can be found in the supplementary material. From the survey results and participants' feedback, we learned that visualizations altering the appearance of the original data would be commonly rejected. Researchers often want fine details and sometimes even noise to be preserved. Interestingly, some of the biologists were able to tell that data had been altered because of their over-smoothed and

thus unnatural look. This is the main reason why the thresholded and labeled results had low scores. The data in Figure 9 had been pre-processed with a median filter. We believe this led to the lower score. Thanks to the ID merging process, our method is able to preserve fine details of the original data, which is important to biologists for a visualization technique.

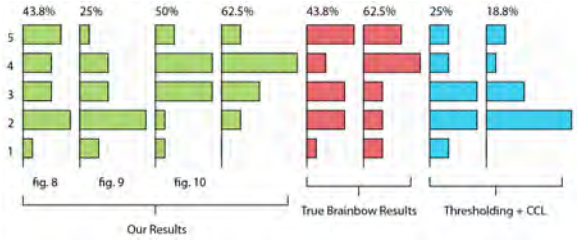

Figure 11: *Results from the first part of our survey. The collective answers to the likelihood of each image being generated with the Brainbow technique are plotted in one bar plot. The length of a bar represents the frequency of each choice being selected (5 - most likely, 1 - most unlikely). Here the images are grouped according to techniques used. In the survey, they were shown to the participants in random order. Above the plots are the combined percentages of the participants who answered 5 or 4. The answers to most of our images are similar to those of the true Brainbows.*

6. Discussion and Conclusions

ID merging with asynchronous cellular automata had been used in connected component analysis, which we regard as the primitive form of more sophisticated colorization. Our quest for computationally generating Brainbow-like images from single-channel confocal microscopy data started with examinations of GPU framebuffer feedback loops. We initially thought it would be purely nondeterministic because of asynchronous memory access in parallel. However, when we tested its behavior with a cellular automaton, we found the behavior of GPU framebuffer feedback loop, despite being nondeterministic, is less random than we thought. In order to use the patterns generated in the iterative process of GPU framebuffer feedback loops to synthesize Brainbow-like results, we introduced ID shuffling and Monte-Carlo sampling into the ID merging process. Our technique is able to enhance visualizations of data with complex structures, such as the biological datasets demonstrated in this paper. Our technique has advantages over traditional segmentation plus labeling methods because of its speed, and also because it preserves fine details of original data. Both make it a visualization technique appealing to domain experts, as we learned from a user survey. Currently, a drawback of our technique is the lack of an intuitive user interface, which limits its users to the authors and their close collaborators. We would like to address this issue in future work.

There are apparent similarities between our technique and many segmentation methods for cellular or fibrous structures in biological data. Algorithm 5 can be considered as a general framework for segmentation, instead of a specific segmentation method. Firstly, it combines component labeling and feature detection into one process. It is also related to fuzzy connected component analysis [US96]. An important benefit of such combination is that size information is readily available and used as a constraint. The size calculated from component labeling is a more accurate descriptor than the previously proposed size-based transfer function [CM08]. Our size descriptor works better with fibrous and branching structures, since they may have big size but be considered small by a local size descriptor. Secondly, the quality of segmentation can be refined by using more specific feature measures. For example, a measure of tubeness [SNS*98] can be added for fibers, and a measure using similarity between smoothed gradient vectors [LLT*07] can be added for cells. Thirdly, Algorithm 5 can be used with or without GPU framebuffer feedback loops. The purpose of this paper is to find a random colorization technique that can be used with GPU framebuffer feedback loops, which saves memory and utilizes memory bandwidth more efficiently. We paid more attention to the stochastic patterns generated in the process and did not discuss convergence in great detail. However, when appropriate constraints are applied, the process can still converge to a segmentation of the input.

In addition to user interface improvement, in future work, we would like to make differently colored structures selectable from three-dimensional views. The selections can then be combined, disconnected, or removed. A further extension would be applications for time sequence data, as we would like to use this method to identify similar features through time.

Acknowledgments

This research was sponsored by NIH-1R01GM098151-01, the DOE NNSA Award DE-NA0000740, KUS-C1-016-04 made by King Abdullah University of Science and Technology (KAUST), DOE SciDAC Institute of Scalable Data Management Analysis and Visualization DOE DE-SC0007446, NSF OCI-0906379, NSF IIS-1162013. We would like to thank A. Kelsey Lewis and her colleagues of the Department of Human Genetics at the University of Utah for a good explanation of the Brainbow technique. We also want to thank all the biologists participated in the survey. The reviewers' comments are very encouraging and helpful to us.

References

[AL04] Adachi S., Lee J.: Computation by asynchronously updating cellular automata. *Journal of Statistical Physics 114*, 1-2 (2004), 261–289. 3

[Bau78] Baudet G. M.: Asynchronous iterative methods for multiprocessors. *Journal of the ACM 25*, 2 (1978), 226–244. 3

[BD04] Branda C. S., Dymecki S. M.: Talking about a revolution: The impact of site-specific recombinases on genetic analyses in mice. *Developmental Cell 6*, 1 (2004), 7–28. 2

[Bia94] Biafore M.: Cellular automata for nanometer-scale computation. *Physica D: Nonlinear Phenomena 70*, 4 (Feb. 1994), 415–433. 2

[CM69] Chazan D., Miranker W.: Chaotic relaxation. *Linear Algebra and its Applications 2*, 2 (1969), 199–222. 3, 5

[CM08] Correa C., Ma K.-L.: Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1380–1387. 9

[CRT94] Cohen A., Roysam B., Turner J.: Automated tracing and volume measurements of neurons from 3-d confocal fluorescence microscopy data. *Journal of Microscopy 173*, 2 (1994), 103–114. 2

[dSB99] di Stefano L., Bulgarelli A.: A simple and efficient connected components labeling algorithm. In *the 10th International Conference on Image Analysis and Processing* (1999), p. 322. 3

[GMRC07] Galilée B., Mamalet F., Renaudin M., Coulon P.-Y.: Parallel asynchronous watershed algorithm-architecture. *IEEE Transactions on Parallel and Distributed Systems 18*, 1 (2007), 44–56. 3

[Gus05] Gustavson S.: *Simplex noise demystified*, Mar. 2005. http://webstaff.itn.liu.se/ stegu/simplexnoise/ simplexnoise.pdf. 6

[GYXS11] Gao Y., Yang J., Xu X., Shi F.: Efficient cellular automaton segmentation supervised by pyramid on medical volumetric data and real time implementation with graphics processing unit. *Expert Systems with Applications 38*, 6 (2011), 6866–6871. 2

[HG93] Huberman B. A., Glance N. S.: Evolutionary games and computer simulations. *PNAS 90*, 16 (1993), 7716–7718. 2

[HLP10] Hawick K., Leist A., Playne D.: Parallel graph component labelling with gpus and cuda. *Parallel Computing 36*, 12 (2010), 655–678. 3, 4

[KP08] Kauffmann C., Piché N.: Cellular automaton for ultra-fast watershed transform on gpu. In *19th International Conference on Pattern Recognition* (2008), pp. 1–4. 2

[KP10] Kauffmann C., Piché N.: Seeded nd medical image segmentation by cellular automaton on gpu. *International Journal of Computer Assisted Radiology and Surgery 5*, 3 (2010), 251–262. 2

[KSH10] Kim E., Shen T., Huang X.: A parallel cellular automata with label priors for interactive brain tumor segmentation. In *IEEE 23rd International Symposium on Computer-Based Medical Systems (CBMS)* (2010), pp. 232–237. 2

[LCH*12] Liu Y., Cheng H. D., Huang J., Zhang Y., Tang X.: An effective approach of lesion segmentation within the breast ultrasound image based on the cellular automata principle. *Journal of Digital Imaging 25*, 5 (2012), 580–590. 2

[LLT*07] Li G., Liu T., Tarokh A., Nie J., Guo L., Mara A., Holley S., Wong S. T.: 3d cell nuclei segmentation based on gradient flow tracking. *BMC Cell Biology 8*, 40 (2007). 9

[LWK*07] Livet J., Weissman T. A., Kang H., Draft R. W., Lu J., Bennis R. A., Sanes J. R., , Lichtman J. W.: Transgenic strategies for combinatorial expression of fluorescent proteins in the nervous system. *Nature 450* (Nov. 2007), 56–62. 1, 2

[Mar84] Margolus N.: Physics-like models of computation. *Physica D: Nonlinear Phenomena 10*, 1-2 (Jan. 1984), 81–95. 2

[NM92] Nowak M., May R.: Evolutionary games and spatial chaos. *Nature 359* (1992), 826–829. 2

[OL10] Oliveira V. M. A., Lotufo R. A.: A study on connected components labeling algorithms using gpus. In *SIBGRAPI 2010* (2010). 3, 4

[PM90] Perona P., Malik J.: Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 7 (1990), 629–639. 6, 8

[RK88] Rothman D. H., Keller J. M.: Immiscible cellular-automaton fluids. *Journal of Statistical Physics 52*, 3-4 (1988), 1119–1127. 2

[SA12] Segal M., Akeley K.: *The OpenGL Graphics System: A Specification*, Nov. 2012. http://www.opengl.org/registry/. 2

[SBYR*91] Smith M. A., Bar-Yam Y., Rabin Y., Margolus N., Toffoli T., Bennett C. H.: Cellular automaton simulation of polymers. In *MRS Fall Meeting* (1991), vol. 248, pp. 483–488. 2

[SNS*98] Sato Y., Nakajima S., Shiraga N., Atsumi H., Yoshida S., Koller T., Gerig G., Kikinis R.: Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images. *Medical Image Analysis 2*, 2 (1998), 143–168. 9

[US96] Udupa J. K., Samarasekera S.: Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation. *Graphical Models and Image Processing 58*, 3 (1996), 246–261. 9

[VK05] Vezhnevets V., Konouchine V.: "growcut" - interactive multi-label n-d image segmentation by cellular automata. In *Graphicon* (2005), pp. 150–156. 2

[vN66] von Neumann J.: *Theory of self-reproducing automata*. University of Illinois Press, 1966. 2

[VV09] Venkatasubramanian S., Vuduc R. W.: Tuned and wildly asynchronous stencil kernels for hybrid cpu/gpu systems. In *the 23rd International Conference on Supercomputing* (2009), pp. 244–255. 3

[WMLK89] Wallis J., Miller T., Lerner C., Kleerup E.: Three-dimensional display in nuclear medicine. *IEEE Transactions on Medical Imaging 8*, 4 (1989), 297–230. 8

[WOCH12] Wan Y., Otsuna H., Chien C.-B., Hansen C.: Fluorender: An application of 2d image space methods for 3d and 4d confocal microscopy data visualization in neurobiology research. In *Pacific Visualization Symposium (PacificVis)* (2012), pp. 201–208. 8