Visualization and Analysis of Digital Light-Sheet Microscopy

Luke Peeler

7 August 2014

Abstract

The movement of cells during embryonic development is an important part of understanding how tissues and organs function in complex organisms. The nerves within nervous system are examples of structures that depend on cell migrations in early development. Early neurological development of zebrafish, mice, and humans includes the migration of facial branchiomotor neurons (FBMNs), an identifiable group of motor neurons that migrate tangentially in mammals and fishes, from their origin in the rhombomere (r4) of the zebrafish hindbrain to their final destination. There are many unanswered questions about the movement of the first FBMN, the pioneer neuron, that may be answered with computational analysis of light-sheet microscopy imaging. Light-sheet microscopy is the only modality that images the entire organism at cellular resolution in three-dimensions, for the time-span over which migration occurs. Visualization and analysis of the light-sheet data will be facilitated by the use of Diderot, a new parallel domain-specific language.

Contents

List of Figures

Chapter 1

Introduction

The study and understanding of cell behavior in organisms at various stages of development is a central goal of developmental biology [1]. In particular, the dynamic behavior and motion of cells during embryonic development hold important keys to a greater understanding of how more fully developed systems, organs, and tissues function. Microscopy of embryonic development and resulting image analysis and visualization are thus indispensable to arriving at a more complete model of cellular dynamics. Of special interest is the tracking of cells through time to discern their provenance and, if possible, to construct a model of their lineage.

Digital light-sheet microscopy (DLSM) is a new, effective technology that has improved performance and image analysis results over other methods, such as confocal and point-scanning two-photon microscopy [1]. Light-sheet microscopy greatly reduces the exposure of a specimen to laser light by only illuminating a single slice at any given time. This process allows for a complete 3D image of a specimen to be quickly acquired with a high signal-to-noise ratio, while reducing damaging effects, such as photo-bleaching. Imaging of organisms using this technology, especially during embryonic development, affords researchers with high-quality 3D+t data sets with which to perform useful analyses, both qualitative and quantitative in nature.

A key aspect of cellular dynamics is concerned with understanding the origins and development of entire lineages of cells. Developing such models allows researchers to understand how complex systems grow and develop from a single cell or group of cells and lends critical insight into the processes by which certain genes are expressed and how, for example, a

given organism might develop predispositions to contract various diseases. Cell tracking is a useful tool in pursuit of such models. Various methods exist which allow researchers to trace the development of individual cells through time in order to ascertain their origins and migratory paths.

The quantitative information afforded by models derived from the results of cell tracking algorithms are complemented by a qualitative view of organismal development offered by 3D volume renderings. Using such visualizations, researchers are able to more effectively contextualize the development of particular systems against the development of the organism as a whole. Moreover, given a sufficiently high temporal sampling rate, 3D volume renderings present a useful high-level view of organismal development through which important developmental events, such as gastrulation, may be studied.

Computational analysis will be facilitated by the use of Diderot, a parallel domain-specific language (PDSL) designed for image analysis and visualization. Diderot offers support for quickly developing and prototyping code suited to image analysis by providing a high-level model of computation and concise notation for expressing image analysis and visualization algorithms [3].

In this paper, I aim to present useful visualization and analysis results using the Teem software and Diderot programming language as a proof-of-concept that can be refined in future work. I will focus on the tangential migration of "pioneer" facial branchiomotor neurons (FBMNs) in zebrafish, a cell type only recently described [10]. During development of vertebrates, all neurons perform radial migration outwards from their birthplace close to the central lumen of the neural tube, but some, such as the FBMNs, also undergo tangential migration, which is important for the proper development of the brain [7, 2, 10]. Visualizations and analysis of the pioneer FBMN and its path during the early stages of embryonic development will be complemented by 3D volume renderings that will offer a high-level view of organismal motion and individual cellular dynamics at varying time points.

Chapter 2

Related Work

The promise of achieving a more complete understanding of cellular dynamics and developmental processes in early development of organisms through light-sheet microscopy and various computational methods, such as cell tracking, has contributed to a growing body of work in which model organisms, such as the zebrafish, fruit fly, and mouse are imaged and studied.

In order to learn more about the movement and cellular dynamics of epiblast cells in mouse embryos during gastrulation, imaging using light-sheet microscopy and computational analysis, including cell tracking, have been performed, with methodology and software made freely available [4].

Moreover, due to its effectiveness and flexibility, light-sheet microscopy has been used for imaging of specimens at varying spatial and temporal scales. This has facilitated the study of many developmental processes, such as gene expression and cellular dynamics, particularly migration. The adaptability of light-sheet microscopy to varying spatial scales may be seen from observations of individual molecules in the salivary gland tissue of *C. tentans* larvae to the imaging of wild-type and mutant zebrafish for 24 hours, at the embryo level [9].

In the latter case, researchers leveraged the high spatial and temporal resolution provided by light-sheet microscopy to generate 'digital embryos', which include quantitative information related to cell position, migratory paths, and division patterns. Digital embryos were computed using automated image segmentation, and the information gleaned from this analysis is to be used in further models of the mechanical forces contributing to

morphogenesis, which constitutes an important step toward a more complete model of early organismal development [5].

Chapter 3

Methods

Analysis and visualization will be performed via two methods, each serving distinct research needs. The important insight provided by understanding the provenance and migratory pathways of cells may be addressed from both a qualitative and quantitative perspective using cell tracking methods. Visualizations which clearly delineate cellular dynamics through time and space offer researchers a high-level view from which to formulate conceptual frameworks governing early organismal development. Additionally, the quantitative data provided by cell tracking methods facilitates more precise modeling though further quantitative analysis. 3D volume renderings are complementary to the results of cell tracking methods, as such visualizations assist researchers in contextualizing the information provided they provide.

Cell tracking methods' aim, in essence is to recover the motion of cells between frames of a 2D+t or 3D+t dataset. There are a number of useful algorithms employed for this purpose, such as cross-correlation methods, deterministic methods, and Bayesian methods [8]. For our purposes and image data, cross-correlation between individual frames seems well-suited. Cross-correlation is also known as a sliding dot product or sliding inner-product. In essence, the image at time $t + 1$ is analyzed against the image at time t using cross-correlation to produce values corresponding to the degree of overlap, and predicting the most likely location of the cell at the later time step is performed using least-squares fitting against such values–specifically through minimization of the χ^2 function, described below. This method suits our data particularly well due to the high signal-to-noise ratio provided by light-sheet microscopy and the fluorescence of the neurons.

Assuming our image is of the following form:

$$I_c(\vec{x}) = \sum_{n=1}^{N} I_p(\vec{x} - \vec{x}_n(t); \ldots),$$

where N is the number of particles and

$$I_p(\vec{x}; \ldots)$$

is a function describing the shape of an idealized particle, with its center at the origin. This function may also depend on other properties of the particle or imaging system.

The most likely location of a particle can be found using least-squares fitting. We define:

$$\chi^2(\vec{x_0}; \ldots) = \int W(\vec{x} - \vec{x_0})[I(\vec{x}) - I_p(\vec{x} - \vec{x_0}; \ldots)]^2 \, d\vec{x},$$

where $W(\vec{x} - \vec{x_0})$ is some weight function corresponding to how sharply particles are defined with respect to the local image region; broad weight functions, such as when $W = 1$, assume that the ideal particle image is sharply defined. We minimize χ^2 when $\vec{x_0}$ is at the position of some particle, allowing us to locate its position. We can extend this process to all particles and find several minima of χ^2 using convolution.

We expand from above:

$$\chi^2(\vec{x_0}; \ldots) = \int W(\vec{x} - \vec{x_0})[I(\vec{x})^2 - 2I(\vec{x})I_p(\vec{x} - \vec{x_0}; \ldots) + I_p(\vec{x} - \vec{x_0}; \ldots)^2] \, d\vec{x}$$

giving us

$$\chi^2(\vec{x_0}; \ldots) = I^2 * W - 2I * (WI_p) + \langle WI_p^2 \rangle$$

where $*$ and $\langle \cdot \rangle$ are defined as follows:

$$f * g = [f * g](\vec{x_0}) = \int f(\vec{x}) g(\vec{x} - \vec{x_0}) \, d\vec{x}$$

$$\langle f \rangle = 1 * f$$

For simplification, we take $W = 1$ and get the following:

$$\chi^2(\vec{x_0}; \ldots) = \int I^2 \, d\vec{x} - 2I * I_p + \langle I_p^2 \rangle$$

We can see that $I * I_p$ will achieve a maximum at the positions of the particles, reflecting the greatest degree of overlap. This is because the "ideal" particle (though we are not using a synthetic model in our computations) will theoretically align with the particle in the target image at its actual center, yielding the highest possible value of $I * I_p$. This is true because the dot product of two vectors, $\vec{u}$ and $\vec{v}$, is maximized when $\vec{u} = \vec{v}$. When $I * I_p$ is at a maximum, we see that χ^2 achieves a minimum, reflecting the minimal error achievable for least squares fitting. For our purposes, we cannot assume that FBMNs will maintain an ideal shape defined by a model suggested by I_p, so we adjust I_p on a per-frame basis using actual data, where we use a small region surrounding the previously-projected center of the cell in frame t to compute the convolution between frames t and $t + 1$, with the aim of maximizing $I * I_p$ nearer to the actual center of the cell in frame $t + 1$.

The cross-correlation program may be found in Figure 6-1 of Appendix A.

The `crossCorrImg` routine performs some setup and basic computation, such as allocating the output image and storing information about the input images in local variables. From here, the function iterates from `-bound` to `bound` along each spatial axis inside the given window size, `wsz`, using the estimated center of the cell in frame t as an origin. The bounds are related to the expected magnitude of displacement between frames, and the size of the window in which to perform the computation is related to the particle diameter. The `crossCorr` function is then called to perform the dot-product computation between the two sections of the input images specified by the given bound, computing a dot-product on a pixel-by-pixel basis and normalizing to the intensity of each image at each pixel. During these iterations, the maximum value returned by the `crossCorr` routine is maintained, along with the offsets along each axis where the maximum was achieved. These offsets are then used to recover the motion of the cell, relative to its position in the previous frame, whose coordinates are supplied at runtime.

Volume renderings offer researchers an informative, high-level view from which to observe organismal development through time and study important developmental milestones, such as gastrulation. Moreover, image data that is sufficiently well sampled spatially provides researchers the opportunity to view embryonic development on the scale of individual cells, offering potential clues into the mechanisms and forces which influence important processes, like cell differentiation and gene expression.

Maximum intensity projection (MIP) is one effective means of generating volume renderings for biological image data. As opposed to direct volume rendering techniques, which require each sample of the image data to be assigned an opacity and color by the transfer function, MIP projects out the voxels of the highest intensity that are intersected by rays traced from the viewpoint to the projection plane. The assignment of opacity and color to these voxels aims to give the appearance of a 3D image, despite being a 2D projection.

The Diderot code for generating an image using the MIP method may be found in Figure 6-2 of Appendix A.

First, the image is read in and stored in-memory. Other initialization parameters set up the viewing space of the image, such as camera/viewer position, viewer direction, and viewer orientation. Additionally, the view space is specified by the positions of the near and far clipping planes and the field-of-view.

From here, the initial position of each ray with respect to the viewing plane (coordinates U and V) is determined by linearly interpolating with respect to the viewing space and image resolution. Then, each ray is traced using a small step size through the viewing space, approaching the far clipping plane. At each step, the image data is sampled, and the intensity maxima are iteratively updated. Upon reaching the far clipping plane, a final computation is made to assign an RGB color value based on the maximum intensity values encountered by the ray during the tracing process.

Another effective means of volume rendering, initially presented by Marc Levoy, involves a similar process of ray casting, but computes opacity and shading of a voxel in independent steps, combining them in the image plane using front-to-back compositing [6]. The opacity assigned to a voxel by the transfer function seeks to avoid aliasing artifacts by assigning voxels near the chosen isovalue to have an opacity that scales inversely with the magnitude

of the local gradient vector. The opacity is assigned as below:

$$\alpha(\vec{x_i}) = \alpha_v \cdot \begin{cases} 1 & \text{if } |\bigtriangledown f(\vec{x_i})| = 0 \\ & \text{and } f(\vec{x_i}) = f_v \\ 1 - \frac{1}{r}\left|\frac{f_v - f(\vec{x_i})}{|\bigtriangledown f(\vec{x_i})|}\right| & \text{if } |\bigtriangledown f(\vec{x_i})| > 0 \\ & \text{and } f(\vec{x_i}) - r \cdot |\bigtriangledown f(\vec{x_i})| \leq f_v \leq f(\vec{x_i}) + r \cdot |\bigtriangledown f(\vec{x_i})| \\ 0 & \text{otherwise} \end{cases}$$

where α_v is the chosen opacity, f_v is the chosen isovalue, and r is the desired thickness in voxels.

Shading is computed using a single directional light:

$$I = k_d L_d(\vec{l} \cdot \vec{n})$$

where I is the resulting intensity coefficient for the final RGB value, k_d is a diffuse reflection coefficient determined by surface properties, L_d is the chosen intensity of the directional light, $\vec{l}$ is the direction of the light, and $\vec{n}$ is the surface normal at a given pixel.

The shading and opacity values are blended with the chosen RGB value for the pixel by front-to-back compositing:

$$C_o = C_A + C_B(1 - \alpha)$$

where C_o is the output color, C_A is the color of the previous sample, C_B is the color of the current sample, and α is the opacity value.

The Diderot code implementing this volume rendering method may be found in Figure 6-3 of Appendix A.

As in the MIP code, the image is loaded into memory and initialization parameters are set to define the viewing space of the image. Ray positions are then interpolated, and each ray is traced according to the step size through the viewing space. The image data is sampled at each step, and the image value and local gradient are used to compute opacity, `aa`, using the method described above, with the isovalue having been chosen by visual inspection using a range of input values for our particular data.

Opacity is then recomputed to be weighted in terms of the step size of the ray. From here the depth of the ray, relative to the near and far clipping planes, is computed and the shading computation is performed. The RGB color assigned to the given voxel is computed and compositing is performed. The `transp` value is decreased by a factor of `1-aa` to facilitate the next iteration's compositing computation, reflecting the increasing opacity of the voxels as the ray continues to penetrate the volume of interest. Tracing of a given ray may terminate upon passing below the given transparency threshold or upon exiting the view space.

A final volume rendering approach was devised, which incorporates the above-described method using both channels of image data, and the combination of a distance transform and colormap. The use of this visualization stems from a potential anomaly identified in a 2-channel volume rendering of the data which revealed that nuclei appeared too near the mid-line of the zebrafish hindbrain. Rather than reside near the centers of the FBMNs, nuclei appeared at or very near the neurons' membranes.

This distance-encoded volume rendering approach is similar to the method describe above, except that both channels of the image are rendered, with a few modifications. Firstly, rather than render the FBMNs' membranes using the red color of the fluorescent protein marking them, the membranes are rendered to be translucent in order to more clearly present the nuclei they contain. Secondly, the region enclosed by the isovalue chosen to represent the neurons' membranes is used to generate a distance map, which is a data structure used during the computation to determine the signed distance of a given point in the view space from the membrane. This distance is then used as an index into the colormap at right, whose values are used to render the nuclei according to their distance to the membrane rather than the green color of their fluorescent protein.

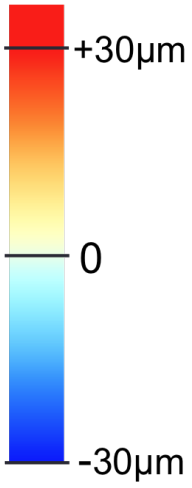


Figure 3-1: Colormap to visualize signed distance of nuclei from cell membrane

Chapter 4

Results

As a means of verifying the correctness of the 3D cross-correlation routine above, a small amount of synthetic data was generated, which approximates an ideal particle for which cross-correlation should be most accurate. The synthetic data represents a particle of radius 10 voxels in a 3D space, with dimensions $217 \times 205 \times 105$. Figure 4-1 (below) shows two 3D images projected along each axis with the maximum intensity.

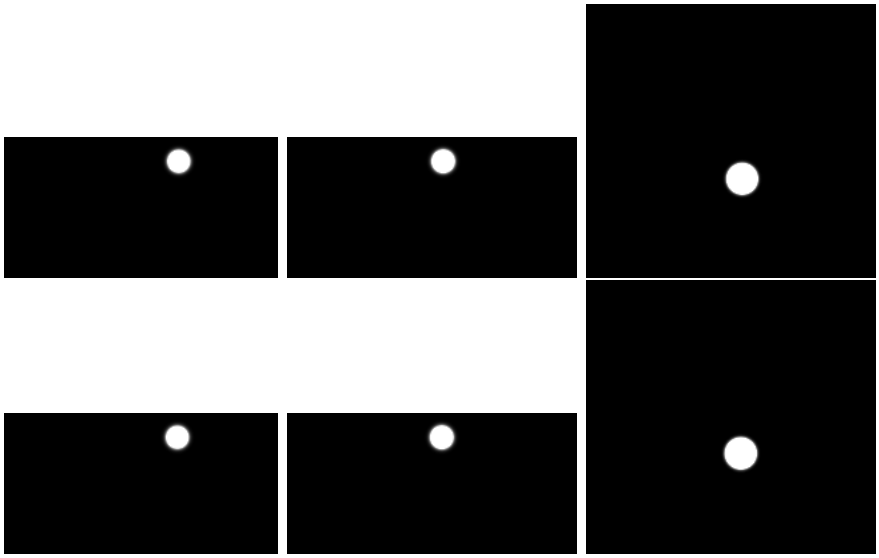


Figure 4-1: Max projections along x, y, and z axes, respectively, of two synthetic image data samples

The results of the 3D cross-correlation routine run on these images indicate that there was a shift along the x-axis of -1 units in index space and a shift along the y-axis of -1 units in index space. These results were consistent with measurements by visual inspection.

The maximum intensity projection along the z-axis of the cross-correlated data within the sliding window can be seen below in Figure 4-2, with a clearly delineated peak of intensity, demonstrating the crux of the algorithm: the maximization of $I * I_p$ and resultant minimization of χ^2.

Figure 4-2: Max projection along z-axis of cross-correlated synthetic image data

Figure 4-3 (below) shows a maximum intensity projection along the z-axis of two adjacent time steps of actual image data, acquired at ? hpf and ? hpf, respectively.

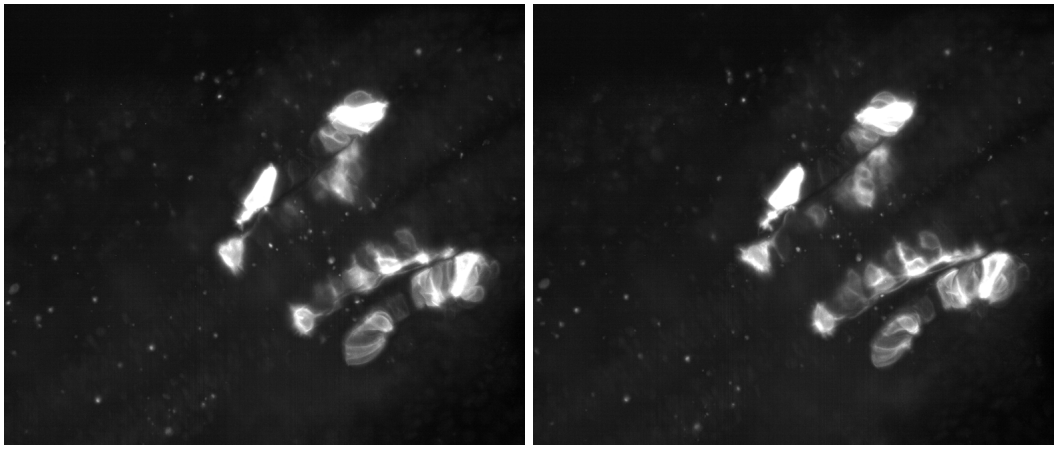


Figure 4-3: Max projection along z-axis of two adjacent time steps of zebrafish embryo image data

Results from the 3D cross-correlation routine run on these images indicate that there was a shift in pioneer FBMN position along the x-axis of -4 units, a shift of 6 units along the y-axis, and a shift along the z-axis of 2 units in index space. As indicated previously, a small region of frame t is defined as a particle template, and computation against the image in frame $t + 1$ is restricted to this window, which is centered at the estimated center of the cell in frame t, plus some bound along each spatial direction indicating maximum possible displacement between frames. Figure 4-4 (below) shows maximum intensity projections along the x, y, and z axes, respectively, of the cross-correlation output, given the image data shown in Figure 4-3 (above). The peaks in intensity can be clearly identified.

Visual inspection reveals a shift of -7, 4, and 4 units, along the x, y, and z axes, respectively, though such estimates are relatively imprecise.

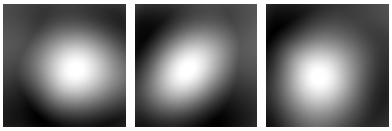


Figure 4-4: Max projection along x, y, and z axes of output of cross-correlation algorithm, given zebrafish embryo image data

Results from the 3D cross-correlation routine run on all images from the data set are visualized below in Figures 4-5, 4-6, and 4-7 using red circles to represent projected cell locations, with a green circle circumscribing the projected position of the cell in the present frame. Each frame represents a single time step, projected onto either the x-y, y-z, or x-z plane, with the full, projected path in the plane appearing on each frame.

Figure 4-5: Visualization of projected trajectory of pioneer neuron in the x-y plane

(i) (ii) (iii)

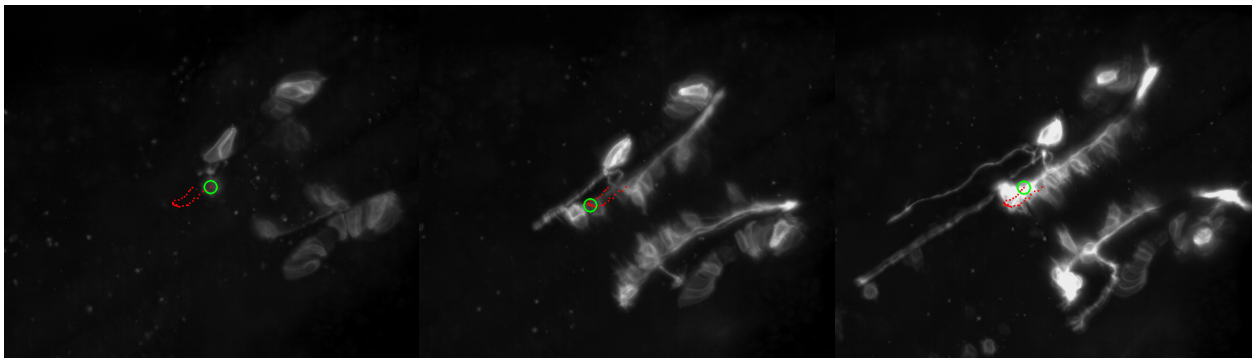


Figure 4-6: Visualization of projected trajectory of pioneer neuron in the y-z plane

(i) (ii) (iii)

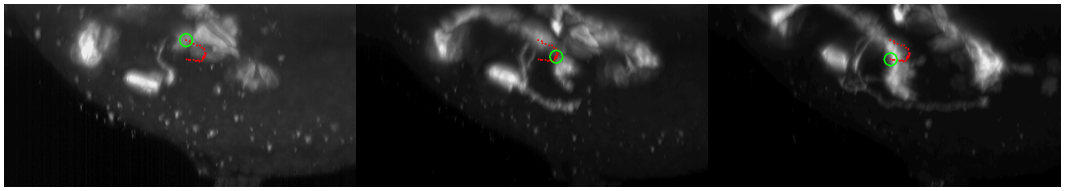
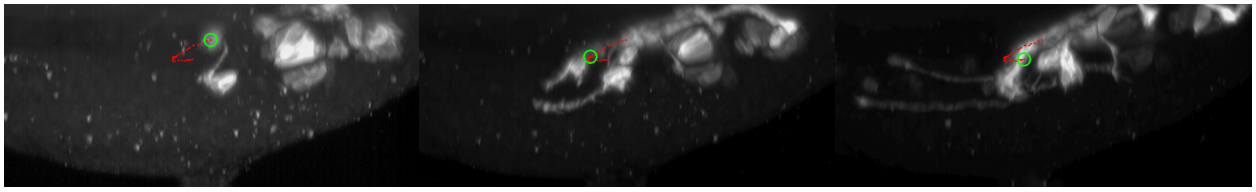


Figure 4-7: Visualization of projected trajectory of pioneer neuron in the x-z plane

(i) (ii) (iii)



Upon initial observation, the projected locations of the pioneer neuron seem incorrect, given the "U-turn" like trajectory suggested by cross-correlation algorithm. However, given the gradual shifts in the field of view as a whole, some error in tracking may be accounted for. The second–and arguably more significant–source of error is due to the approach toward the location of the pioneer FBMN by the neurons which are thought to follow its path. It is certainly possible that these neurons fluoresce brighter in aggregate than the pioneer neuron due to their proximity to it and the increasing presence of the fluorescent marker protein as development progresses through time. It is also critical to note that we are not tracking particles whose shape is static, but rather the membranes of complex cells that are changing in shape and direction as time progresses. An additional drawback of our chosen approach relates to the fact that computation begins with a single, hand-estimated center in the first time step; this means that small errors may propagate and compound in successive computations.

Results from the maximum-intensity projection (MIP) method are seen below in the same frames as the results presented in Figures 4-5 through 4-7.

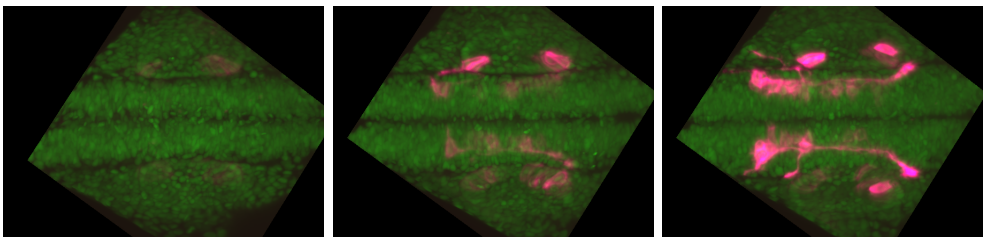


Figure 4-8: 2-channel maximum intensity projections along the z-axis

These results give researchers a familiar visualization in 2 channels from which to get a basic sense of the developmental context in which early migration of the pioneer FBMNs occurs.

Below are results of the volume rendering method involving opacity and shading. The isovalue defining the isosurface visualized below was chosen by visual inspection to best approximate the shape of neurons' cell membranes.

These visualizations give a much better sense of the shape and size of FBMNs through time. More accurate visualizations may be achieved with a choice of isovalue that more closely approximates the cell membrane; however, results will generally only approximate the FBMNs' shape due to the relationship of the choice of isovalue to the nature of the

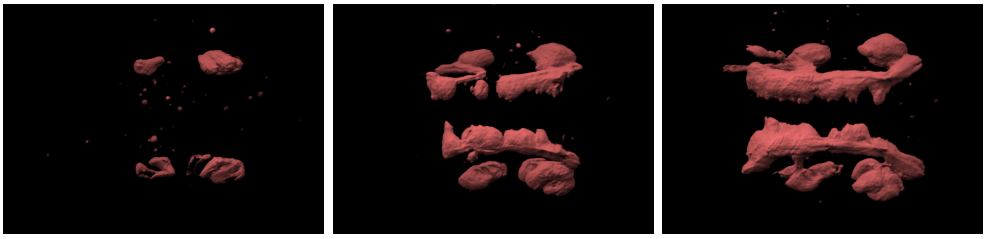


Figure 4-9: 3D volume rendering of FBMNs at an isovalue of 1500

computation.

Finally, we have the results of the distance-encoded volume rendering that visualize the positions of nuclei relative to the FBMNs' membranes. Given the quantitative information

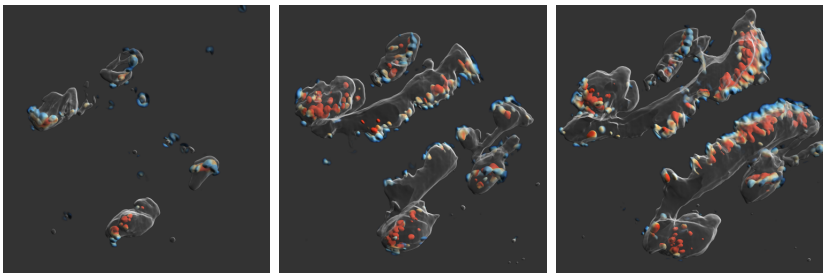


Figure 4-10: Distance-encoded volume rendering showing relative positions of nuclei to neuron membranes

provided by the colormap in Figure 3-1, these visualizations will be of use to researchers in altering the parameters of image acquisition to ensure the accuracy of image data and in determining if there is some legitimate biological cause for the nuclei positions to be what they are in these renderings.

Chapter 5

Future Work

A logical next step for further work would involve acquiring more image data, perhaps with acquisition parameters informed by the results presented in this paper. However, a number of improvements may be made to our methods using the same image data. Chief among these is the preprocessing of image data to improve its signal-to-noise ratio and systematic identification of cell centers at various time steps to provide a quantitative, rather than only qualitative, sense of error in the cross-correlation method. Another possibility involves some method of normalizing image intensity across time steps, as the increasing presence of fluorescing proteins at later time steps might affect our computation. In terms of performance and efficiency, the template cell region chosen at frame t when examining frame $t + 1$ might be cropped prior to computation, as a means of improving cache locality and performance.

In addition to studying the cellular dynamics of the pioneer FBMN, understanding the provenance of all FBMNs during early embryonic development is of particular value to developmental biologists. One important method to support this work is the construction of an abstract model of cell lineage. Such a tree-based model could include, at each node, information about position, time, stage of mitosis, etc. The abstract nature of this analysis would enable biologists to study the development of multiple organisms from multiple image acquisitions without needing to observe and decipher the raw image data, enabling them to identify common developmental patterns and possible anomalies in early development. The methods and results of this work will hopefully serve as a springboard that informs and guides further research.

Chapter 6

Appendix A - Visualization and analysis code

Figure 6-1: 3D cross-correlation program

```
static double
crossCorr(const unsigned short *aa, const unsigned short *bb,
          const unsigned int sza[3], const unsigned int szb[3],
          const int off[3], int wsz, int prev_pos[3]) {


  int xi, yi, zi, lo[3], hi[3];
  unsigned int ii;
  double dot; // numerator
  double dot_chan0, dot_chan1;
  double lena, lenb; // factors in denominator

  double lena_chan0, lena_chan1, lenb_chan0, lenb_chan1;
  for (ii=0; ii<3; ii++) {

    // determine coordinates of window of computation along each axis
    lo[ii] = AIR_MAX(0, prev_pos[ii] - wsz/2);
    hi[ii] = lo[ii] + wsz;
```

```
}



dot_chan0 = dot_chan1 = lena_chan0 = lena_chan1 = lenb_chan0 = lenb_chan1 = 0;
dot = lena = lenb = 0;

for (zi=lo[2]; zi<=hi[2]; zi++) {
  for (yi=lo[1]; yi<=hi[1]; yi++) {
    for (xi=lo[0]; xi<=hi[0]; xi++) {

      float a_chan0, a_chan1, b_chan0, b_chan1;

      // compute offsets into first image
      int idx_aa_x = xi;
      int idx_aa_y = sza[0]*yi;
      int idx_aa_z = sza[0]*sza[1]*2*zi;
      int idx_aa = idx_aa_x + idx_aa_y + idx_aa_z;
      a_chan0 = aa[idx_aa];
      a_chan1 = aa[idx_aa + sza[0] * sza[1]]; // get value for next channel by
          offseting by the size of a whole xy-slice


      // compute offsets into second image
      int idx_bb_x = xi + off[0];
      int idx_bb_y = szb[0] * (yi+off[1]);
      int idx_bb_z = szb[0] * szb[1] * 2 * (zi + off[2]);
      int idx_bb = idx_bb_x + idx_bb_y + idx_bb_z;


idx_bb = (idx_bb < 0) ? 0 : idx_bb;
      b_chan0 = bb[idx_bb];
      b_chan1 = bb[idx_bb + szb[0] * szb[1]]; // get value for next channel by
          offsetting by the size of a whole xy slice
```

```
        // compute dot product and denominator values
        dot += a_chan0*b_chan0 + a_chan1*b_chan1;
        lena += a_chan0*a_chan0 + a_chan1*a_chan1;
        lenb += b_chan0*b_chan0 + b_chan1*b_chan1;


        dot_chan0 += a_chan0 * b_chan0;
        dot_chan1 += a_chan1 * b_chan1;

        lena_chan0 += a_chan0 * a_chan0;
        lena_chan1 += a_chan1 * a_chan1;
        lenb_chan0 += b_chan0 * b_chan0;
        lenb_chan1 += b_chan1 * b_chan1;
      }
    }
  }

  return dot_chan0 / (sqrt(lena_chan0)*sqrt(lenb_chan0)) + dot_chan1 /
      (sqrt(lena_chan1)*sqrt(lenb_chan1));
}

int
crossCorrImg(Nrrd *nout, int maxIdx[3], Nrrd *nin[2], int bound, int windowSz,
    int prev_pos[3], airArray *mop) {


  static const char me[]="crossCorrImg";
  char *err, done[13];
  unsigned int sza[3], szb[3], ii, szc;
  unsigned short *aa, *bb;
  double *cci, maxcc, cc;
  int ox, oy, oz;


  // check that each supplied NRRD is 3D, 2 channel (technically 4D)
```

```
for (ii=0; ii<2; ii++) {
  if (!( 4 == nin[ii]->dim && nrrdTypeUShort == nin[ii]->type )) {
    fprintf(stderr, "%s: input %s isn't 3D, 2 channel ushort array "
            "(instead got %u-D %s array)\n", me, !ii ? "A" : "B",
            nin[ii]->dim, airEnumStr(nrrdType, nin[ii]->type));
    return 1;
  }
}
sza[0] = nin[0]->axis[0].size;
sza[1] = nin[0]->axis[1].size;
sza[2] = nin[0]->axis[3].size; // in this case, z-axis is stored as 4th axis
szb[0] = nin[1]->axis[0].size;
szb[1] = nin[1]->axis[1].size;
szb[2] = nin[1]->axis[3].size; // in this case, z-axis is stored as 4th axis


// store data in memory as flat array
aa = AIR_CAST(unsigned short *, nin[0]->data);
bb = AIR_CAST(unsigned short *, nin[1]->data);



szc = 2*bound + 1;
if (nrrdAlloc_va(nout, nrrdTypeDouble, 3,
                 AIR_CAST(size_t, szc),
                 AIR_CAST(size_t, szc),
                 AIR_CAST(size_t, szc))) {
    airMopAdd(mop, err = biffGetDone(NRRD), airFree, airMopAlways);
    fprintf(stderr, "%s: trouble allocating output:\n%s\n", me, err);
    return 1;
  }

  cci = AIR_CAST(double *, nout->data);

  maxcc = AIR_NAN;
  fprintf(stderr, "%s: computing ...   ", me); fflush(stdout);
```

```
for (oz=-bound; oz<=bound; oz++) {
  int off[3];
  fprintf(stderr, "%s", airDoneStr(-bound, oz, bound, done));
  fflush(stdout);
  off[2] = oz;

  for (oy=-bound; oy<=bound; oy++) {
    off[1] = oy;

    for (ox=-bound; ox<=bound; ox++) {
      off[0] = ox;

      cc = cci[ox+bound + szc*(oy+bound) + szc*szc*(oz+bound)] =
          crossCorr(aa, bb, sza, szb, off, windowSz, prev_pos);


      if (!AIR_EXISTS(maxcc)) {
        maxcc = cc;
      } else {
        // remember where the max is
        if (cc > maxcc) {
          maxIdx[0] = ox;
          maxIdx[1] = oy;
          maxIdx[2] = oz;
          maxcc = cc;
        }
      }
    }
  }
}

fprintf(stderr, "%s\n", airDoneStr(-bound, oz, bound, done));
fflush(stdout);
```

```
      return 0;
}



char *corrInfo=("Simple program for measuring 3D, 2-channel image correlation"
                " over a 3D array of offsets. "
                "Prints out offset coordinates that maximized the "
                "cross correlation");

int
main(int argc, const char **argv) {
  const char *me;
  hestOpt *hopt;
  hestParm *hparm;
  airArray *mop;

  char *err, *outName;
  Nrrd *nin[2], *nout;
  int bound, maxIdx[3], wsz, prev_pos[3], tIdx;

  me = argv[0];
  mop = airMopNew();
  hparm = hestParmNew();
  hopt = NULL;
  airMopAdd(mop, hparm, (airMopper)hestParmFree, airMopAlways);
  hestOptAdd(&hopt, "i", "a b", airTypeOther, 2, 2, nin, NULL,
             "two input 3D, 2 channel images A and B to correlate", NULL, NULL,
             nrrdHestNrrd);
  hestOptAdd(&hopt, "b", "max", airTypeInt, 1, 1, &bound, "10",
             "maximum offset");
  hestOptAdd(&hopt, "o", "file", airTypeString, 1, 1, &outName, NULL,
             "output filename");
  hestOptAdd(&hopt, "w", "window size", airTypeInt, 1, 1, &wsz, NULL,
             "window size (related to diameter of feature in image for which
                 cross-correlation is of interest)");
```

```
hestOptAdd(&hopt, "pp", "x y z", airTypeInt, 3, 3, prev_pos, NULL,
          "cell position in index space of image a");

hestOptAdd(&hopt, "t", "timestep index", airTypeInt, 1, 1, &tIdx, NULL,
          "timestep index of image a (0 if image a is the first in the series,
              1 if the second, etc.)");
hestParseOrDie(hopt, argc-1, argv+1, hparm,
              me, corrInfo, AIR_TRUE, AIR_TRUE, AIR_TRUE);
airMopAdd(mop, hopt, (airMopper)hestOptFree, airMopAlways);
airMopAdd(mop, hopt, (airMopper)hestParseFree, airMopAlways);

nout = nrrdNew();
airMopAdd(mop, nout, (airMopper)nrrdNuke, airMopAlways);



if (crossCorrImg(nout, maxIdx, nin, bound, wsz, prev_pos,mop)) {
  fprintf(stderr, "%s: trouble\n", me);
  airMopError(mop);
  return(1);
}


printf("cc = %d %d %d\n", maxIdx[0], maxIdx[1], maxIdx[2]);


// write out projected coordinates of image B to file, for use as seed value in
// next iteration, assuming code is run in scripted style
char fname[64];
// tIdx + 1 means we're writing the projected coordinates of the next frame
    (e.g., frame t+1)
sprintf(fname, "coords-%d.txt", tIdx + 1);
FILE * next_pos_fp = fopen(fname, "w");
if (!next_pos_fp) {
  fprintf(stderr, "%s: trouble writing projected coordinates for next frame\n",
      me);
```

```
  }

  char buf[64];
  sprintf(buf, "%d %d %d", prev_pos[0] + maxIdx[0], prev_pos[1] + maxIdx[1],
      prev_pos[2] + maxIdx[2]);

  fputs(buf, next_pos_fp);
  fclose(next_pos_fp);


  if (nrrdSave(outName, nout, NULL)) {
    airMopAdd(mop, err = biffGetDone(NRRD), airFree, airMopAlways);
    fprintf(stderr, "%s: trouble saving output:\n%s\n", me, err);
    airMopError(mop);
    return(1);
  }

  airMopOkay(mop);
  exit(0);
}
```

Figure 6-2: Diderot code for generating an image using the MIP method

```
// volume dataset
field#0(3)[] FF0 = tent  image("vol-0.nrrd");
field#0(3)[] FF1 = tent  image("vol-1.nrrd");

// set camera and image parameters

vec3 camEye = [-5586.59, 7053.3, -22285.7];
vec3 camAt = [2354.57, 1866.76, 511.564];
vec3 camUp = [-0.566504, -0.824, 0.00986904];
real camNear = -700.0;
real camFar = 700.0;
```

```
real camFOV = 5.02;
int imgResU = 600;
int imgResV = 430;

real rayStep = 2;

// boilerplate computation of camera and light info
real camDist = |camAt - camEye|;
real camVspNear = camNear + camDist;
real camVspFar = camFar + camDist;
vec3 camN = normalize(camAt - camEye);
vec3 camU = normalize(camN  camUp);
vec3 camV = camN  camU;
real camVmax = tan(camFOV*/360.0)*camDist;
real camUmax = camVmax*real(imgResU)/real(imgResV);

// RayCast(ui,vi) computes rendered color for pixel (ui,vi)
strand RayCast (int ui, int vi) {
    real rayU = lerp(-camUmax, camUmax, -0.5, real(ui), real(imgResU)-0.5);
    real rayV = lerp(-camVmax, camVmax, -0.5, real(vi), real(imgResV)-0.5);
    vec3 rayVec = (camDist*camN + rayU*camU + rayV*camV)/camDist;

    real rayN = camVspNear;
    real mip0 = 0.0;
    real mip1 = 0.0;
    output vec3 rgb=[0,0,0];

    update {
       vec3 pos = camEye + rayN*rayVec;
       if (inside (pos,FF0)) {
          mip0 = max(mip0, FF0(pos));
          mip1 = max(mip1, FF1(pos));
       }
       if (rayN > camVspFar) {
          stabilize;
       }
```

```
      rayN = rayN + rayStep;
   }

   stabilize {
      rgb=[mip1, 0.7*mip0, 0.5*mip1];
   }
}

initially [ RayCast(ui, vi) | vi in 0..(imgResV-1), ui in 0..(imgResU-1) ];
```

Figure 6-3: Diderot code for generating volume rendering using Levoy transfer function

```
// volume dataset
field#1(3)[] FF0 = bspln3  image("vol-0.nrrd");
field#1(3)[] FF1 = bspln3  image("vol-1.nrrd");

// set camera and image parameters

input bool camOrtho = false;
vec3 camEye = [-5586.59, 7053.3, -22285.7];
vec3 camAt = [2354.57, 1866.76, 511.564];
vec3 camUp = [-0.566504, -0.824, 0.00986904];
real camNear = -600.0;
real camFar = 700.0;
real camFOV = 5.02;
int imgResU = 600;
int imgResV = 430;


real rayStep = 1.5;
vec3 lightVsp = normalize([-2.0, -5.5, -2.0]);

// boilerplate computation of camera and light info
real camDist = |camAt - camEye|;
```

```
real camVspNear = camNear + camDist;
real camVspFar = camFar + camDist;
vec3 camN = normalize(camAt - camEye);
vec3 camU = normalize(camN  camUp);
vec3 camV = camN  camU;
real camVmax = tan(camFOV*/360.0)*camDist;
real camUmax = camVmax*real(imgResU)/real(imgResV);
vec3 light = lightVsp[0]*camU + lightVsp[1]*camV + lightVsp[2]*camN;

input real isoval = 1500;
input real thick = 20;
function real alpha(real v, real g) = 0.1*(1.0 if v > isoval else clamp(0, 1, 1 -
    |v-isoval|/(g*thick)));

// RayCast(ui,vi) computes rendered color for pixel (ui,vi)
strand RayCast (int ui, int vi) {
    real rayU = lerp(-camUmax, camUmax, -0.5, real(ui), real(imgResU)-0.5);
    real rayV = lerp(-camVmax, camVmax, -0.5, real(vi), real(imgResV)-0.5);
    vec3 rayVec = camN if camOrtho
                    else (camDist*camN + rayU*camU + rayV*camV)/camDist;
    vec3 rayEye = (rayU*camU + rayV*camV if camOrtho else [0,0,0]) + camEye;

    real rayN = camVspNear;
    vec3 rgb = [0, 0, 0];
    output vec4 rgba=[0,0,0,0];
    real transp = 1;

    update {

       vec3 pos = rayEye + rayN*rayVec;

       if (inside (pos,FF0)) {

          real val1 = FF1(pos);

          vec3 grad1 = - FF1 (pos);
```

```
        real aa = alpha(val1, |grad1|);

        if (aa > 0) {
            aa = 1 - pow(1-aa, rayStep);
            real depth = lerp(1, 0.2, camVspNear, rayN, camVspFar);
            real shade = lerp(0, 1, -1, normalize(grad1)light, 1);
            vec3 color = [0.4,1.0,0.4];
            rgb += transp*aa*depth*shade*color;
            transp *= 1 - aa;
        }

      }


      if (transp < 0.01) {
        transp = 0;
        stabilize;
      }

      if (rayN > camVspFar) {
        stabilize;
      }
      rayN = rayN + rayStep;
    }

    stabilize {
      rgba = [rgb[0], rgb[1], rgb[2], 1-transp];
    }

}

initially [ RayCast(ui, vi) | vi in 0..(imgResV-1), ui in 0..(imgResU-1) ];
```

Chapter 7

Appendix B - `czireader` software

In addition to the image analysis and visualization work presented above, significant effort in pursuit of these results was devoted to the design of an efficient file reader for the `.czi` file format specified by Carl Zeiss Microscopy, GmbH, the manufacturer of the digital light-sheet microscope from which the image data originated.

Though not of particular interest in terms of the results sought after by developmental biologists, work on this file reader presented an opportunity to apply various skills gained during my coursework as an undergraduate in the Department of Computer Science at The University of Chicago. Various aspects of file format reading, such as file I/O; binary representation of data and bitwise operators; and cache performance, learned from a myriad of systems courses, informed my approach when helping to develop this software. Furthermore, I gained better familiarity with object-oriented design concepts and facility with the C++ programming language through my coursework as a student in the department's Bx/MS program, both of which proved invaluable.

This software was designed with the goal of giving researchers a fast, efficient tool by which to extract image data from `.czi` files, without having to rely on much larger software packages, such as ImageJ, which are designed for a variety of uses. Moreover, we hope to make the `czireader` software available as an open-source project, as there is a great deal of extensibility possible. One such example is outputting image data in formats supported by a variety of tools for image analysis and visualization. The `.nhdr` file extension used by the Teem software package is currently supported, and there are countless similar opportunities for expansion.

Acknowledgments

I would first and foremost like to sincerely thank my advisor, Dr. Gordon Kindlmann, for his extensive assistance throughout all stages of this project; his patience and guidance were invaluable. I'm very grateful also to Scott McDonald of Carl Zeiss Microscopy, who provided information and source code supporting the `czireader` software. I'd also like to thank Dr. Yali Amit and Dr. Victoria Prince for their many helpful suggestions during the process of revising this document. Image data was acquired by members of The University of Chicago Microscopy Core Facility: Dr. Christine Labno, Dr. Vytas Bindokas, and Dr. Sarah Wanner. Development of the Diderot language is thanks to Charisee Chiw, Dr. Gordon Kindlmann, Lamont Samuels, Nicholas Seltzer, and Dr. John Reppy. Lastly, I would like to thank the Department of Computer Science at The University of Chicago for affording me the opportunity to pursue this work as an undergraduate through its Bx/MS program.

Bibliography

[1] Fernando Amat and Philipp J Keller. Towards comprehensive cell lineage reconstructions in complex organisms using light-sheet microscopy. *Development, Growth & Differentiation*, 55(4):563–578, 2013.

[2] Anand Chandrasekhar. Turning heads: Development of vertebrate branchiomotor neurons. *Developmental Dynamics*, 229(1):143–161, Jan 2004.

[3] Charisee Chiw, Gordon Kindlmann, John Reppy, Lamont Samuels, and Nick Seltzer. Diderot: A parallel dsl for image analysis and visualization. In *Proc. Programming Language Design and Implementation (PLDI)*, pages 111–120. ACM, 2012.

[4] Takehiko Ichikawa, Kenichi Nakazato, Philipp J Keller, Hiroko Kajiura-Kobayashi, Ernst H K Stelzer, Atsushi Mochizuki, and Shigenori Nonaka. Live imaging and quantitative analysis of gastrulation in mouse embryos using light-sheet microscopy and 3d tracking tools. *Nat. Protocols*, 9(3):575–585, 03 2014.

[5] Philipp J. Keller, Annette D. Schmidt, Joachim Wittbrodt, and Ernst H.K. Stelzer. Reconstruction of zebrafish early embryonic development by scanned light sheet microscopy. *Science*, 322(5904):1065–1069, 2008.

[6] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, 1988.

[7] Oscar Marín and John L. R. Rubenstein. A long, remarkable journey: Tangential migration in the telencephalon. *Nature Reviews Neuroscience*, 2(11):780–790, 2001.

[8] Jens Rittscher, Raghu Machiraju, and Stephen T. C. Wong. *Microscopic Image Analysis for Life Science Applications*. Artech House, Inc., Norwood, MA, USA, 1 edition, 2008.

[9] Raju Tomer, Khaled Khairy, and Philipp J Keller. Shedding light on the system: studying embryonic development with light sheet microscopy. *Current Opinion in Genetics & Development*, 21(5):558–565, 2011. Developmental mechanisms, patterning and evolution.

[10] Sarah J. Wanner and Victoria E. Prince. Axon tracts guide zebrafish facial branchiomotor neuron migration through the hindbrain. *Development*, 140(4):906–915, 2013.