2017

Visualizing Morphogenesis with the Processing Programming Language

Avik Patel  
*Touro University California*

Amar Bains

Richard Millet

Tamira Elul  
*Touro University California, tamira.elul@tu.edu*

Follow this and additional works at: https://touroscholar.touro.edu/tuccom_pubs

Part of the *Biological Phenomena, Cell Phenomena, and Immunity Commons*, and the *Programming Languages and Compilers Commons*

Recommended Citation


This Article is brought to you for free and open access by the College of Osteopathic Medicine at Touro Scholar. It has been accepted for inclusion in College of Osteopathic Medicine (TUC) Publications and Research by an authorized administrator of Touro Scholar. For more information, please contact carrie.levinson2@touro.edu.
Visualizing Morphogenesis with the Processing Programming Language

Avik Patel, Amar Bains, Richard Millet, and Tamira Elul

We used Processing, a visual artists’ programming language developed at MIT Media Lab, to simulate cellular mechanisms of morphogenesis – the generation of form and shape in embryonic tissues. Based on observations of in vivo time-lapse image sequences, we created animations of neural cell motility responsible for elongating the spinal cord, and of optic axon branching dynamics that establish primary visual connectivity. These visual models underscore the significance of the computational decomposition of cellular dynamics underlying morphogenesis.

Introduction

Processing is a Java based software language and environment created at MIT Media Laboratory in 2001 for visual modeling (www.processing.org). Processing has grown into a popular tool and online community for programming within the visual arts and data visualization fields. Several artists have used Processing to model or simulate various biological phenomena. One artist animated a matrix of biological cells switching between two states (Polyptic by George Legrady), while another simulated fungal hyphae growth using images as food (Mycelium by Ryan Alexander). A textbook further describes how Processing can animate dynamic processes in nature using diverse underlying theoretical models (Shiffman 2012). However, Processing has yet to make its full mark in modeling of biological phenomena. One potential area of application for Processing is computational visualization of morphogenesis in developmental biology.

Morphogenesis - the generation of shape and form in embryonic tissues - is a critical process in developmental biology (Gilbert 2013). Morphogenesis is responsible for elongating the body axis, forming the organs in the body, and establishing neuronal circuitry in the brain, among other processes. A central issue in the study of morphogenesis is determining the cell behaviors (e.g. motility, rearrangement, shape change, and division) that are responsible for changing tissue shapes. A powerful technique for elucidating the dynamic cellular mechanisms of morphogenesis is time-lapse video-microscopic imaging of cells in embryonic tissues undergoing morphogenesis (Elul and Keller 2000; Elul et al., 1997; Harris et al., 1987). The mechanisms underlying morphogenetic processes can be clarified by observation of cell dynamics from these time-lapse video sequences. Morphometric measurement further defines the quantitative and statistical relationships between the cell dynamics that underlie morphogenesis (Kim and Davidson 2011; Marshak et al., 2007; Elul et al., 1997; Witte et al., 1996). Mathematical decomposition of cell dynamics additionally facilitates the computational modeling of cellular mechanisms that drive morphogenesis (Satulovsky et al., 2008).

Methods

In this paper, we use the Processing programming language to visualize dynamic cell behaviors driving morphogenesis in the developing nervous system. Based on in vivo time-lapse image sequences, we created models of cell dynamics underlying two morphogenetic processes in the developing nervous system. We first computationally modeled the cell motility that drives neural tube formation and spinal cord elongation. The terminal branching of optic axonal arbors that establish synaptic connectivity in the primary visual projection was then simulated. These visualizations highlight the process of computational decomposition of the dynamic cell behaviors that shape the developing nervous system, which may be useful for artists seeking to illustrate the development of form in biology.

Results

Modeling of Cell Motility Driving Neural Ectoderm Morphogenesis

Our first Processing model simulated the cell motility underlying neural tube formation and spinal cord elongation. During development, the spinal cord is formed by a series of morphogenetic processes that transform the flat neural plate into the closed neural tube. Disruptions of these morphogenetic processes result in neural tube defects that can severely affect nervous system function. One morphogenetic
process critical to neural tube formation is ‘convergent extension’, in which the neural ectoderm simultaneously extends its antero-posterior axis and narrows its mediolateral axis. Researchers have used time-lapse imaging and quantitative analysis to study the cellular mechanisms of neural convergent extension in embryos of diverse species, including newt (Burnside and Jacobson 1968), frog (Elul and Keller 2000; Elul et al., 1997), zebrafish (Hong and Brewster 2006), chick (Ezin et al., 2009), and mouse (Williams et al., 2014). The conclusion from these studies is that the embryonic spinal cord tissue elongates by neural cells intercalating (rearranging) along the mediolateral axis (Figure 1a). Some studies further demonstrate that during convergent extension, neural cells express mediolaterally-oriented or randomly-directed dynamic protrusions, which they apply to one another to generate force for their rearrangement (Figures 1a and 1b; Elul and Keller 1999).

Figure 1. (A) Time series of still images depicts a small group of fluorescently-labeled neural cells intercalating in a neural ectoderm explant. In the images depicted, the green cell moves between the yellow and blue cells, and the purple cell then moves between the green and blue cells. This results in elongation of the group of cells along the antero-posterior axis of the tissue, which is oriented vertically. Scale Bar – 10 μm. (B) Contours of a biological neural ectoderm cell shows how its shape (protrusions) and position change over time, during 2 minute intervals (* represents cell at 0 min). (C) Contours of the model motile cell during iterations of the program, shows how its shape and position changes as it moves towards the cursor (* represents cell at start of program).

The computational model was based on a time-lapse image sequence of a small group of fluorescently-labeled motile neural cells intercalating in an explant undergoing convergent extension (Figure 1a; Supplementary Video 1; Elul and Keller 1999). Our Processing program focused on animating the motility of a single two-dimensional neural cell within this group (Figures 1b and 1c; Supplementary Video 1). Future versions of the model will attempt to simulate the more complex rearrangements and interactions of a group of motile neural cells. To model the dynamics of an individual neural cell, we incorporated several shape and motility parameters (Figures 1b and 1c; Supplementary Code 1). We determined that a slightly off-circle polygon, with eight nodes or vertices, was a suitable representation of neural cell shape. Using the ‘mouse’ command in Processing, we set the cursor to be an external force, attracting the cell. We then modeled the change in position of the cell center and nodes in response to the cursor position. The ‘acceleration’ of the cell was linearly proportional to the distance between the cell and the cursor, becoming stronger the farther away the cell was from the cursor. ‘Springing’ and ‘damping’ parameters were added to positively and negatively modulate the acceleration of the cell, respectively. In each loop of the program, the new cell center position was determined as a function of the acceleration, while new node positions were established based both on the acceleration, and on a specific random number or ‘frequency,’ assigned initially to each node. This resulted in each of the nodes moving somewhat independently from the other nodes of the cell. In addition, during movement of the cell, the nodes rotated through an angle, mimicking the rotational motion of the neural ectoderm cells in the time-lapse sequence. Finally, we established an organic constant parameter which was also dependent on the acceleration (distance between cursor and cell center). The organic constant modulated the curvature or convexity of the polygon sides as the cell moved towards the cursor. We created analogous shape and motility parameters for the nucleus of our model cell, which in the future, could allow us to explore nuclear-cellular motility relationships.

Visualization of Optic Axon Branching in the Developing Visual System

We also modeled the terminal branching of optic axonal arbors that form the primary visual connection and establish visual function. During development of the nervous system, axons extend to their target tissues in the brain, where they branch into arbors that contain synaptic junctions. Axonal arbors are dynamic structures, with remodeling of branches and synapses occurring throughout life, in correlation with cognitive processes such as learning and memory (Meyer and Smith 2006). Optic axonal arbors that establish visual connectivity are peripherally located in the brain, and can be imaged in intact, living animals, including lower vertebrates such as tadpoles of frogs or fish (Figure 2a; Marshak et al., 2009; Witte et al., 1996; Harris et al., 1987). From in vivo time-lapse videos of optic axonal arbors, previous researchers quantified the rates of branch additions and retractions, and the mean lifetimes of branches that underlie the remodeling of these arbors (Figure 2a; Marshak et al., 2007; Witte et al., 1996). Another study measured and modeled the mean branching angle of axon arbors in vitro (Shefi et al., 2005). Studies showed that molecular perturbations can misshape...
optic axonal arbors by altering the rates of addition, retraction and lifetime of arbor branches (Wiley et al., 2008; Elul et al., 2003). Some of these molecular perturbations of the optic axonal arbor dynamics have also been associated with disruptions of visual synaptic function (Mannitt et al., 2009; Ruthazer et al., 2006; Alsina et al., 2001).

The model of axonal arborization was based on a time series of images of an optic axon branching in the tectum of an intact tadpole (Figure 2; Supplementary Code 2, Marhsak et al., 2007). The Processing sketch began with a single vertically oriented ‘optic axon’ (Figure 2b). In each loop of the program, the terminal branches split into additional daughter branches (Figure 2b; Supplementary Video 2). The parameters in this model included the number of daughter branches a branch bifurcated into, the maximal branching angles change (*). In addition, parent branches are thicker than daughter branches. Scale Bar – 10 μm (printed with permission from Marhsak et al., 2007). Sequential images show remodeling of model arbor in three subsequent iterations of the program (number of daughter branches set at two, maximal branching angle set at 60°, based on images of biological arbor shown in A). As the arbor grows, new daughter branches are added, angles between the branches are changed, and parent branches grow longer and thicker.

**Figure 2.** (A) Time series of images of an optic axonal arbor in the brain of an intact, living tadpole. In the images shown, new branches are added (arrow), existing branches elongate (open arrow), and branching angles change (*). In addition, parent branches are thicker than daughter branches. Scale Bar – 10 μm (printed with permission from Marhsak et al., 2007). (B) Sequential images show remodeling of model arbor in three subsequent iterations of the program (number of daughter branches set at two, maximal branching angle set at 60°, based on images of biological arbor shown in A). As the arbor grows, new daughter branches are added, angles between the branches are changed, and parent branches grow longer and thicker.

Other researchers have developed rules-based models of cell motility and branching dynamics that may also be useful to artists seeking to illustrate morphogenesis. One study modeled cell movement by defining local protrusion and global retraction factors that modify shape and motility parameters of a model cell (Satulovsky et al., 2008). Another group incorporated adhesive, drag, repulsive, and directed migration forces to change the acceleration of individual cells within an epithelial sheet (Vitorino et al., 2011). Computational models and quantitative descriptions of branching morphogenesis in biology are also prevalent. The textbook, “The Algorithmic Beauty of Plants”, which is popular within the Processing community, describes ‘L-systems’- a fractal-based, recursive model of plant branching and development (Prusinkiewicz and Lindenmayer 1990). In addition, the NeuroMorph database is a repository for thousands of digitized arbor images together with associated morphometric parameters (www.neuromorph.org). Finally, one chapter of the book “Computational Neuroanatomy,” focuses on determining the minimum number of parameters necessary to accurately model geometry of neuronal arbors (Burke and Marks 2002).

**Discussion**

The spatial and temporal resolution we depicted in our Processing animations of cell and arbor dynamics were determined, in part, by the resolution of the underlying time-lapse image sequences. We sketched the motile neural cell in real time (at a frame rate of 25), based on time-lapse image sequences captured at relatively high temporal resolution (30 second intervals; Figure 1a; Supplementary Video 1; Elul and Keller 2000; Elul et al., 1997). In contrast, we modeled axonal branching at a more discrete time scale (frame rate of 1), based on a series of images of optic axons captured at two hour intervals (Figure 2a; Supplementary Video 2, Marhsak et al., 2007). Over the last decade, many new microscopic imaging technologies with greatly increased temporal and spatial resolution have been developed (Krzic et al., 2012). Researchers have used one of these new technologies - light sheet microscopy - to capture 4D images of embryonic development in Drosophila and other small organisms in toto (Chen et al., 2014). Having such higher-resolution, multi-dimensional time-lapse images as the basis for our Processing simulations would likely increase the temporal and spatial scale of the animations.

Researchers have used one of these new technologies - light sheet microscopy - to capture 4D images of embryonic development in Drosophila and other small organisms in toto (Chen et al., 2014). Having such higher-resolution, multi-dimensional time-lapse images as the basis for our Processing simulations would likely increase the temporal and spatial scale of the animations.
Additional morphogenetic processes could be modeled using Processing or other visual programming languages. Morphogenetic processes involving cell movement include gastrulation (Skoglund et al., 2008), eye development (Chauhan et al., 2015), and metastasis of cancer cells (Spanjaard et al., 2015). Branching morphogenetic processes that could be modeled encompass lung morphogenesis (Menshykau et al., 2014), establishment of blood vascular networks (Boas and Merks 2015), and branching of tumors (Miura 2015; Takaki 2005). Blender (www.blender.org), an open source 3D animation and graphics software, could also be used to model morphogenesis. Blender would be especially useful for modelling three-dimensional projections of cells and arbors generated with confocal and light-sheet microscopy techniques (Chen et al., 2014; Elul et al., 2003). In more general terms, tools from the arts and technology fields could help biologists and artists create effective visualizations that communicate the dynamics of morphogenesis to a broader audience.

Acknowledgements

The authors thank the artist who suggested they use the Processing language for animating morphogenesis.

References


of retinotectal synaptic connectivity by modulating axon arborization and synapse formation in the developing brain. Journal of Neuroscience, 29(36):11065-11077. [PubMed] [Cross Ref]


Authors

Avik Patel, BSc, is a third-year medical student at Touro University California. He received his Bachelor of Science in Neurobiology from UC Davis. His interests include visualization of anatomy and biochemical processes. Avik was excited to learn Processing and apply it to the modeling of optic axonal arbors in vivo.

Amar Bains, BA, recently graduated from the University of California, Berkeley with a biochemistry degree. He is pursuing a Master’s degree post-graduation, and plans to eventually attend medical school. In his free time, he practices the tuba and dances Bhangra for UC Berkeley's official team.

Richard Millet is a programmer based in the San Francisco Bay Area. He has worked for Apple, and in the Research Information Technology group at UC Berkeley. Currently, he is the technical lead for CollectionSpace, an open source software for Museum Databases.

Tamira Elul, PhD, is a scientist and educator. She is an Associate Professor at Touro University, California where she teaches medical students histology and directs an experimental...
and theoretical research program focusing on neuronal morphogenesis. In recent years, she has used art and artistic tools to disseminate morphogenesis to a broader community. Tamira Elul can be contacted at tamira.elul@tu.edu.

**Supplementary Video**

**Video 1 (video link also available in the HTML)**

Animation shows computational cell protruding and deforming as it moves towards the cursor as an ‘attractive force’ (left panel). Note change in curvature of sides of the model cell as it moves towards the cursor. A video (which loops 5 times) depicting biological neural ectodermal cells moving and intercalating is also shown (right panel, time interval between frames is 30 seconds).

**Video 2 (video link available in the HTML)**

Animation shows model axonal arbor branching and growing, with two new daughter branches being added to each parent branch in each iteration of the program, and a maximal branching angle of 60°. These parameters are based on analysis of biological arbors, as shown in Figure 2A (Marshak et al., 2007).

**Supplementary Code 1**

**Model Neural Cell Motility**

```java
// slow down springing
accelX = -damping;
accelY = -damping;

// change curve tightness as cell moves
organicConstant = 1 - (1-abs(cX-c0))*(1-abs(cY-c0))*0.5;

//move nodes of the cell polygon incorporating frequency
for (let i=0; i<nodes;i++)
{
  node[i][0] = nodeStart[i][0] + rndf*(angle[i]*Math.sin(angle[i])-organicConstant);
ode[i][1] = nodeStart[i][1] + rndf*(angle[i]*Math.cos(angle[i])-organicConstant);
  angle[i] = angle[i]*frequency[i];
}

// set position, shape and mobility parameters for nucleus
float centerX = 0, centerY = 1;
float radius = 5, rotAngles = 90;
float accelX=0, accelY=0;
float springing = -0.05, damping = -0.5;

// inner nodes for nucleus
int nodes = 10;
float nodeStart[10][0] = new float[nodes];
float nodeStart[10][1] = new float[nodes];
float nodeLin = new float[nodes];
float nodeF = new float[nodes];
float angle[10][0] = new float[nodes];
float angle[10][1] = new float[nodes];
float rate = new float[nodes];

// soft-body dynamics
float organicConstant = 1;

void drawShapes()
{
  // calculate nucleus node starting locations
  for (let i=0; i<nodes;i++)
  {
    nodeStart[i][0] = centerX+rotAngles*radius;
ode[i][0] = centerX+rotAngles*radius;
ode[i][1] = centerY+rotAngles*radius;
    rotAngles = 360/10/nodes;
  }

  // draw polygon for nucleus
curveTightness = organicConstant; // 0.05, b. 202;
  beginShape();
  for (let i=0; i<nodes;i++)
  {
    curveVertex(node[i][0],node[i][1]);
    curveVertex(node[i][0],node[i+1][0]);
    curveVertex(node[i+1][0],node[i+1][1]);
    curveVertex(node[i][0],node[i+1][0]);
  }
  endShape(CLOSE);
}

void moveShapes()
{
  // move center point
  float deltax = mouseX-centerX;
  float deltay = mouseY-centerY;

  // create springing effect for nucleus
  deltax = springing;
deltay = springing;
  accelX += deltax;
  accelY += deltay;

  // move nucleus's center
  centerX += accelX;
  centerY += accelY;

  // slow down springing
  accelX = -damping;
  accelY = -damping;

  // change curve tightness as nucleus moves
  organicConstant = 1 - (1-abs(cX-c0)*(1-abs(cY-c0)))*0.5;

  // move nodes of the nucleus
  for (let i=0; i<nodes;i++)
  {
    node[i][0] = nodeStart[i][0] + Math.sin(angle[i]*Math.sin(angle[i])-organicConstant);
ode[i][1] = nodeStart[i][1] + Math.sin(angle[i]*Math.cos(angle[i])-organicConstant);
    angle[i] = angle[i]*frequency[i];
  }
}
```
void setup() {
    size(300, 640); // set size of window

    // center shape in window
    centerX = width/2;
    centerY = height/2;
    centerW = width/2;
    centerH = height/2;

    // initialize frequencies for corner nodes
    for (let i=0; i<nodes; i++) {
        frequency[i] = random(0.1, 12);
        nold[i] = i;
        framerate = 25; // set rate of movement of cell
    }

    void draw() {
        // set background; this part of the program will loop
        background(0, 0, 0);
        rect(0, 0, width, height);
        drawShape(); // draw cell polygon
        moveShape(); // move cell polygon
        drawShape(); // draw cell nucleus
        moveShape(); // move cell nucleus

        // center point
        float centerX = 0, centerY = 0;
        float radius = 45, n0angle = -80; // set radius and rotational angle for cell polygon
        float accall, accv;
        for (let i=0; i<nodes; i++) {
            nold[i] = i;
            for (let n=0; n<nodes; n++) {
                float noldX = nold[n];
                float noldY = nold[n];
                float nnewX = nold[n];
                float nnewY = nold[n];
                float nangle = nold[n];
                float newX = nold[n];
                float newY = nold[n];
                float newV = nold[n];
                float newA = nold[n];
                float newfreq = nold[n];

                // soft body dynamics
                float organicConstant = 1;

                void drawShape() {
                    // calculate node starting locations
                    for (let i=0; i<nodes; i++) {
                        nodeStartX[i] = centerX + radius * cos(radians(n0angle)) * radius;
                        nodeStartY[i] = centerY + radius * sin(radians(n0angle));
                        n0angle += 360.0/nodes;
                    }
                    // draw cell polygon
                    curveBegin(); // sets quality of the curves connecting nodes of cell
                    for (let i=0; i<nodes; i++) {
                        curveVertex(nodeStartX[i], nodeStartY[i]); // sets vertices of the cell polygon
                    }
                    curveEnd();
                    // move cell polygon
                    moveShape(); // move cell polygon
                    drawShape(); // draw cell nucleus
                    moveShape(); // move cell nucleus
                }

                void moveShape() {
                    if (mouseUp) {
                        moveX = mouseX - centerX;
                        moveY = mouseY - centerY;
                    }
                }
            }
        }
    }
}
### Supplementary Code 2

#### Model Axonal Branching

```java
float x=0;
float q355;
float w1=1;
float e=355;

import java.awt.swing.JOptionPane; //Dialog boxes;
int at;
int bt;

void setup()
{
  size(600,600); //Set canvas size;
  String input1 = JOptionPane.showInputDialog("Input initial branch number (1,2,3,4)");
  a = Integer.parseInt(input1); //Allow users to choose the number of initial branches;
  String input2 = JOptionPane.showInputDialog("Input maximum branching angle (b, \(1\,\text{deg}\))");
  b = Integer.parseInt(input2); //Allows user to define the absolute maximum branching angle;
}

void draw(){
  smooth();
  frameRate(1); //Rate of arbor remodeling is set;
  loop(); //Program will loop continuously;

  newProjectile();
  }

void mousePressed(){
  //Allow user to 'pause' with a cursor click;

  void newProjection(){
    background(10);
    g2d.fillOval(10, 10, 50, 50);
    stroke(40, 40, 40); //Color formatting. Color is dynamic;
    text("Click to 'pause', 10, height-10);
    translate(width/2, height/2);
    branch(); //Branch() is a function that packages multiple 2D transformations;
    
    if(x>height/4){
      x=height/4; //Set vertical growth boundary. Keeps visualization in the frame;
      
    }
}

void branch(float h){
  float w = wq*h; //wq = 1.25;
  stroke(40, 40, 40); //Color formatting. Color is dynamic;
  translate(width/2, height/2);
  branch(); //Branch() is a function that packages multiple 2D transformations;
  
  if(x>height/4){
    x=height/4; //Set vertical growth boundary. Keeps visualization in the frame;
  }
}
```

**Licensing**

The authors have chosen to license this content under a Creative Commons Attribution, NonCommercial, NoDerivatives License 4.0 International License.