High-Level Sprout Geometry Extraction for In Vitro Angiogenesis Assays

Gio Borje
University of California, Irvine
gborje@uci.edu

Craig Steinke
University of California, Irvine
steinkec@uci.edu

ABSTRACT

We have developed an automated system for the quantitative analysis of *in vitro* angiogenesis assays. Specifically, the system is designed to analyze fibrin gel bead sprouting assays developed by the laboratory of Dr. Chris Hughes in the Department of Molecular Biology & Biochemistry at UC Irvine. Our system enumerates sprouts for each bead and calculates the average branching factor and the average length for an imaged assay. Our approach reports accurate sprout enumeration when compared with manual counts by an expert observer while significantly reducing the analysis time.

1. INTRODUCTION

Angiogenesis is a mechanism for the formation of new blood vessels from pre-existing vessels. Tumor angiogenesis in solid tumors is thought to facilitate the transition from a benign tumor to a malignant phenotype. Additionally, the avascular growth phase allows for an an approximate maximum size of 1–2mm in diameter; on the other hand, the vascular growth phase enables unyielding tumor expansion and metastasis [3]. Subsequently, the analysis of *in vitro* angiogenesis assays is necessary to assess the impact of pro-angiogenic and anti-angiogenic agents.

Niemisto et. al developed a general image analysis method for the quantification of similar *in vitro* angiogenesis assays. Their method provides the length and size of each tubule complex as well as the number of junctions within them [5]. The TCS Cellworks Angiokit (Buckingham, UK) images used by Niemisto et. al have two distinguishable and ideal properties in contrast to our images that require us to use a variant approach. First, their images have tubule complexes that are solidly filled whereas our images have visible lumens. Second, their images show no distinguishable origin for sprouts whereas our images have a bead from which sprouts originate. Hence, we propose a modified methodology for our image set.

The image set under analysis is provided by the Hughes Lab. Their fibrin bead assay using human umbilical vein endothelial cells (HUVEC) contrasts with fibrin bead assays containing Bovine aortic endothelial cells (BAEC). HUVEC have been the canonical EC model system and more accurately represents angiogenesis in humans. The assay begins by culturing HUVEC as a monolayer on dextran-coated Cytodex beads. This method induces HUVEC to recapitulate multicellular capillaries in fibrin gels. Furthermore, the method for HUVEC promotes sprouting, lumen formation and long-term stability of neovessels. The high-resolution images of beads are then captured on an IX70 Olympus microscope [4]. A sample image can be seen on Figure 1.

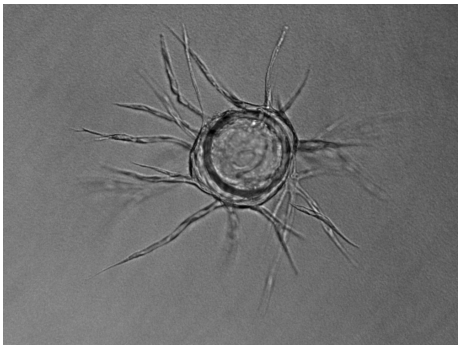


Figure 1: Single Bead Image from the Hughes Lab Images

Components of an image are described as follows. A *bead* refers to the Cytodex bead. The *sprouts* are the multicellular capillary structures formed by endothelial cells. The *sprout count* refers to the number of sprouts per bead. For quantitative analysis, the objective of the system is to yield sprout counts that are consistent with an expert observer.

Previously, the number of sprout counts per bead were measured by manual counting on the obtained images. Furthermore, the sprout length was ambiguously measured in arbitrary units [4]. Our system relieves the laborious, manual analysis by quickly automating the detection and reliably analyzing of imaged assay features.

Our system, the High-Level Sprout Geometry (HLSG) Extractor, is designed to detect high-level object and analyze the objects of imaged assays. Objects for detection include the Cytodex bead and its associated multicellular capillaries. Due to the noise and depth of the image, the HLSG Extractor uses structural inpainting to infer which sprout

segments emerge belong the same sprout. After structural inpainting, the HLSG Extractor quantitatively analyzed the images using Sholl Analysis to determine the number of primary sprouts, the average length and the average branching factor for each bead in the imaged assay.

In addition to the High-Level Sprout Geometry (HLSG) Extractor, a driver and report generator are implemented to drive functionality on sample images and generate reports on the analyses respectively.

The following sections will proceed as follows. Section 2 will describe an overview of the HLSG feature detection and analysis mechanisms. Section 3 will describe the data structures used to represent the HLSG and its properties. Section 4 shows how the system is designed for modularity and robustness.

2. METHODOLOGY

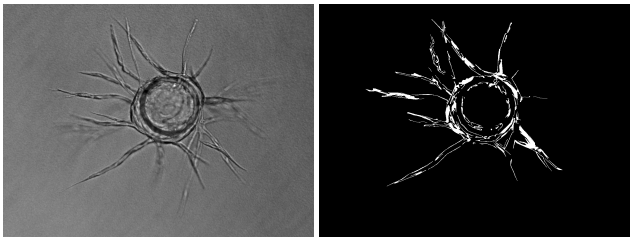
Our system enables detection of high-level features and quantitative analysis of morphometrics which can be decomposed into four stages. The first stage is preprocessing: obtain a black and white image such that the foreground only contains two objects of interest: beads and sprout segments. The second stage is bead detection where the system finds the origin and radius of each bead in the image. The third stage is non-bead detection where the system collects all sprout segments from the image. Once the sprout segments are collected, the system adjusts the sprout segments to reduce noise in the analysis stage. The system begins by restoring holes and small gaps within sprout segments. Second, spurious arcs and noise are pruned. Finally, the resulting image is analyzed through Sholl Analysis which collects the number of crossings for a set of concentric circles with a fixed origin as the origin of a detected bead.

2.1 Preprocessing

In the preprocessing step, the system converts the image into a black and white image where white represents the foreground and black represents the background. Objects in the foreground are elements of interest. Specifically, a foreground object can belong one of three higher-order objects: a bead, a sprout segment or noise. Noise is an artifact generated from the original image as well as the image processing techniques. We use the Canny edge detection algorithm [2] to obtain the black and white images such the edges represent the contours of the elements of interest.

2.2 Bead Detection

Bead detection is a three-step process. To reduce noise, the system first smooths the image using a Gaussian blur. Next, the system obtains a black and white version of the image through the edges of the image using the Canny edge detection algorithm. Finally, circles in the image are detected using the circular Hough Transform [7]. The circles



(a) Original (b) Edges

Figure 2: Edge Detection Outline

detected correspond to the beads in the imaged assay. It follows that the origin and radius of the bead are obtained.

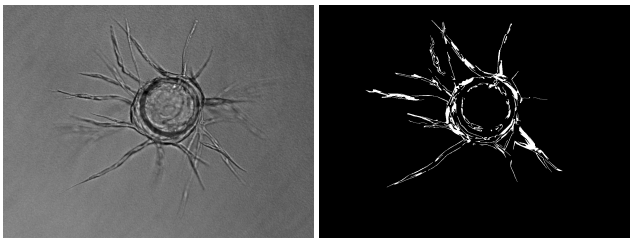
(a) Edges (b) Bead Detection

Figure 3: Bead Detection Outline

In order to avoid false bead detections, we enforce that the minimum distance between the origin of every pair of detected circles be four times the average bead radius while preferring beads with the more votes in the accumulator i.e. each detected bead should at least be a bead apart. We ignore the degenerate case in which two or more beads are within the distance constraint because associating sprouts to beads becomes inaccurate.

2.3 Non-Bead (Sprout) Detection

The system begins by obtaining a black and white image from the grayscale imaged assay. Since the edges of the images represent the contours of beads, sprouts and noise, we mask the detected beads such that only the sprout and noise contours remain. The noise is reduced in the sprout restoration phase.

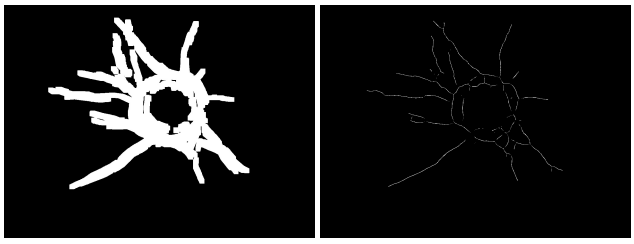


(a) Original (b) Edges

Figure 4: Edge Detection Outline

Since the borders of the sprouts are detected as edges, they may appear to be distinctly segmented objects. We call such objects *sprout segments*. Subsequently, it is necessary that the system links the sprout segments to reconstruct the original sprout.

The system approaches the reconstruction by first thickening the sprout segments until the lumen disappears. Next, the thickened structure is is skeletonized by thinning. Since the skeleton represents the fundamental morphology of the structure, and approximate centerline for each of the sprouts are generated. We now utilize the disjoint center-



(a) Thickening (b) Thinning and Pruning

Figure 5: Sprout Detection Outline

lines as sprout segments.

2.3.1 Sprout Segment Connectivity

Due to the discontinuity of sprouts in the assay, we begin by collecting all sprout segments and represent them as line segments. We distinguish the end points of each line segment as either an inner point or outer point. The inner point, I, is the end point on the line segment such that it is closer to the origin of the line segment's closest bead. The outer point, O, is the end point on the line segment such that it is farther from the origin of the line segment's closest bead.

To determine which line segments belong to the same sprout, we use Euclidean distance between their inner and outer points enforced by the constraint that the end point must be closer to the origin of its closest bead than the target start point. For example, two line segments are part of the same sprout if the distance between the start and end points are within a specified distance parameter, d.

Once the sprouts segments have been connected, we refer to the connected structures as sprouts.

2.3.2 Sprout Restoration

In this phase, there are two primary sources of degradation observed. First, noise is still in the foreground after sprouts segments are connected. Second, holes may exist in the morphology of the sprouts after thickening and thinning the structures. We describe our methods for resolving each problem independently.

Noise in the image can be described in two common forms: isolated points and spurious arcs. Isolated points refer to the small objects detected in the foreground that have no apparent relation to sprout segments. Spurious arcs are auxiliary stubs that are generated from skeletonization. To remove the noise from the black and white image, we apply several hit-and-miss transformation that masks isolated points and spurious arcs. Hence, only the sprout segments remain.

$$\begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

Figure 6: Hit-and-Miss Transformation Kernel for Isolated Points

In contrast with noise, holes are points that can be found within foreground structures that should have been detected as foreground. These holes are primarily artifacts of the Canny edge-detection algorithm since the contours of a capillary encompass the lumen which is detected as background. By the morphology of these structures, we simply find the contours which enclose small holes and perform inpainting by polygon approximation using the Ramer-Douglas-Peucker algorithm.

2.4 Sholl Analysis

Sholl Analysis is a quantitative method for quantitatively analyzing morphological characteristics of neurons [6]. We use Sholl Analysis because the morphology of our imaged assays closely resembles neuron images after our preprocessing steps.

Briefly, Sholl Analysis consists of counting the number of foreground to background crossings given concentric circles of a centered at the origin of a bead. The algorithm can then be decomposed into initialization, iteration and final stages. For the intialization, we select a bead, B. Next, let the initial circle radius, $r_{\min}$, be slightly larger than the detected radius of the bead and let the maximum radius, $r_{\max}$, be the minimum distance from the origin of the detected bead to either of the four image borders. For each radius, $r \in [r_{\min}, r_{\max}]$, we we define the number of crossings as $N(r)$. For further analysis, we define the critical value, $\mathrm{CV}(N)$, as the radius at which the maximum number of crossings occur. See A.1 for the pseudo code using an optimized Bresenham Circle Algorithm.

For posterior analysis, we define longest increasing subsequence (LIS) of $N(r)$ as $\mathrm{LIS}(N)$ with the domain constrained between $[r_{\min}, \mathrm{CV}(N)]$. The longest decreasing subsequence (LDS), $\mathrm{LDS}(N)$ is defined similarly with the domain constrained to $[\mathrm{CV}(N), r_{\max}]$. The LIS and LDS guarantee that we have strictly increasing counts to ignore gaps as noise.

From the number of crossings, we determine the sprout counts and average length. The sprout counts, c_N, are de-

termined by a median of the LIS(r). To determine the average length, l_N, we define a proecdure. First, we define by the radius at which LDS(N) has a value of 1 as $l_{\max}$ and we define the radius at which LDS(N) has a value of c_N as $l_{\min}$. Then, we calculate l_N as the average of $l_{\max}$ and $l_{\min}$.

3. SYSTEM ARCHITECTURE

The system requires Python version 2.7x with the SimpleCV package. The architecture of the system is based on our methodology for quantitatively analyzing *in vitro* angiogenesis. The system, however, incorporates modules for driving batch processes as well as a Read-Eval-Print-Loop (REPL) for console interaction. Finally, a module incorporated for generating CSV reports of the analysis. The sequence diagram for the system components are shown in Figure 8.

The REPL module controls the interaction between the user and the system. Commands available in the REPL are shown in Table 1.

Command	Output	Description
extract [file]	HLSG of file	Extracts the HLSG of the given file.
extract [files]	HLSG of files	Extracts the HLSGs of the given files.
exit	Goodbye	Exits the system.

Table 1: Commands

4. RESULTS

The results of our sprout enumeration show a standard error of 1.7 when compared manual sprout counts of an expert observer (see Table 2).

Treatment	Human Counts	HLSG Counts
10k 1 Day 7	7	10
10k 5 Day 7	10	10
1k 2 Day 7	11	9
1k 3 Day 7	14	14
20k 1 Day 7	15	10
20k 2 Day 7	15	12
20k 3 Day 7	6	8
20k 5 Day 7	9	10
Ang1 100 ng/ml 3 Day 7	7	9
Ang1 100 ng/ml 5 Day 7	9	12
Ang1 250 ng/ml 1 Day 7	8	7
Control 1 Day 7	10	12
Control 2 Day 7	12	13
Control 4 Day 7	8	8
Control 5 Day 7	7	9

Table 2: Expert Observer vs. HLSG Sprout Counts

5. DISCUSSION

Although the methodology is similar to AngioQuant, our method operates on two-ridge structures [5].

6. ACKNOWLEDGEMENTS

We would like to thank Dr. Ernie Esser and Anna Konstorum from the Department of Mathematics, University of California, Irvine, for valuable commentary and guidance throughout the research process. We would also like to acknowledge the laboratory of Dr. Chris Hughes in the Department of Molecular Biology & Biochemistry at UC Irvine for making available the image sets of their fibrin gel bead sprouting assays. This research was funded by the National Science Foundation (NSF) DMS-0928427.

7. REFERENCES

[1] Jack Bresenham. A linear algorithm for incremental digital display of circular arcs. *Commun. ACM*, 20(2):100–106, February 1977.

[2] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[3] Robert S. Kerbel. Tumor angiogenesis: past, present and the near future. *Carcinogenesis*, 21(3):505–515, 1999.

[4] Martin N. Nakatsu, Richard C.A. Sainson, Jason N. Aoto, Kevin L. Taylor, Mark Aitkenhead, Sofía Pérez del Pulgar, Philip M. Carpenter, and Christopher C.W. Hughes. Angiogenic sprouting and capillary lumen formation modeled by human umbilical vein endothelial cells (huvec) in fibrin gels: the role of fibroblasts and angiopoietin-1. *Microvascular Research*, 66(2):102 – 112, 2003.

[5] A. Niemisto, V. Dunmire, O. Yli-Harja, Wei Zhang, and I. Shmulevich. Robust quantification of in vitro angiogenesis through image analysis. *Medical Imaging, IEEE Transactions on*, 24(4):549–553, 2005.

[6] D. A. Sholl. Dendritic organization in the neurons of the visual and motor cortices of the cat. *J Anat.*, 87(4):387–406, 1953.

[7] H. K. Yuen, J. Princen, J. Dlingworth, and J. Kittler. A comparative study of hough transform methods for circle finding. In *Proc. AVC*, pages 29.1–29.6, 1989. doi:10.5244/C.3.29.

APPENDIX

A. PSEUDO CODE

This section outlines the pseudo code for Sholl Analysis by using an optimized Bresenham Circle Algorithm for circular point generation.

A.1 Sholl Analysis

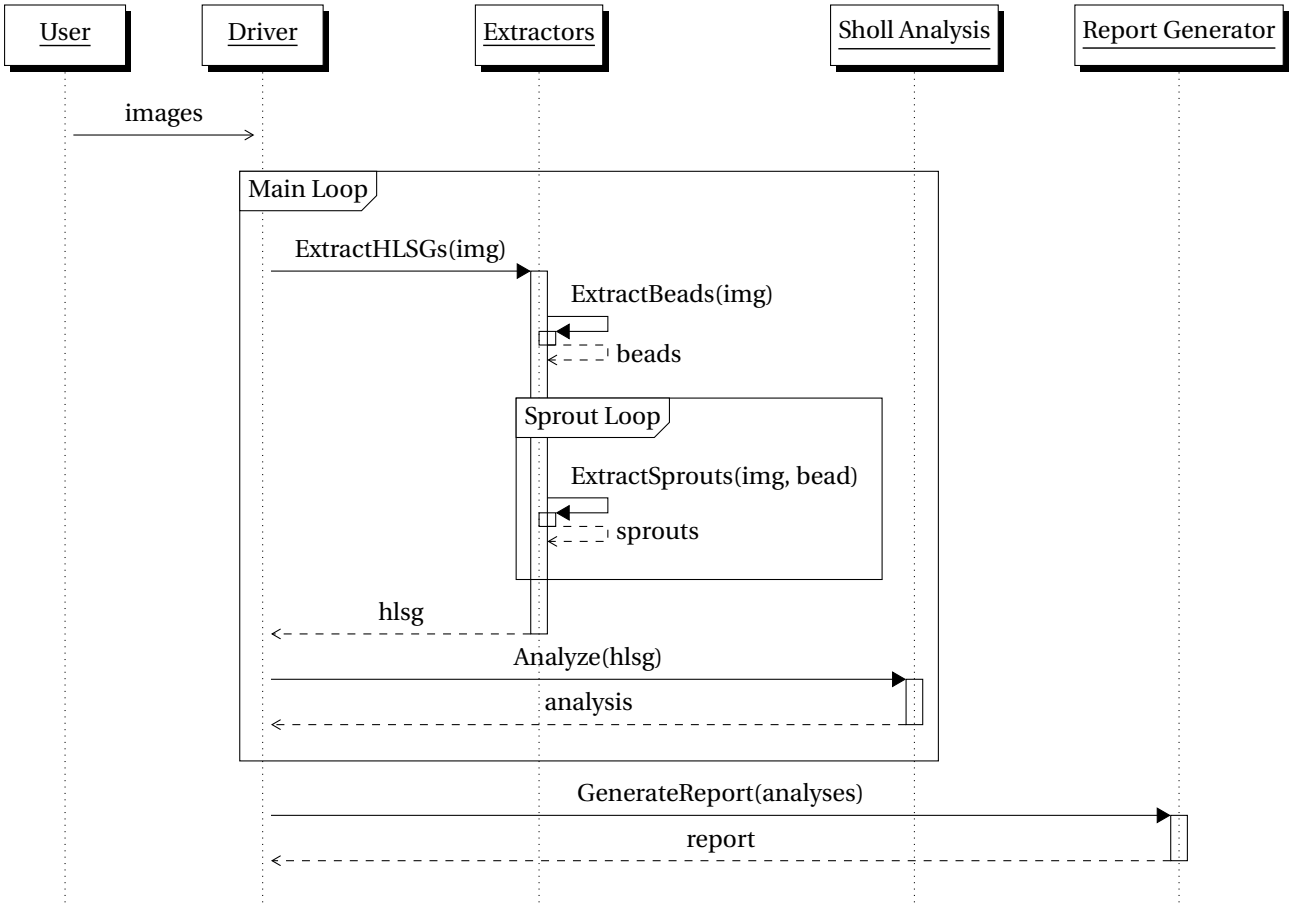


Figure 7: High-Level Architecture

Algorithm 1 Sholl Analysis

```
procedure SHOLLANALYSIS(img, origin, radius)
    r_min ← radius
    r_max ← min{origin.x, img.width - origin.x}
    r_max ← min{r_max, origin.y, img.height - origin.y}
    crossings ← empty dictionary
    for all r ∈ [r_min, r_max] do
        crossings[r] ← COUNTCROSSINGS(img, origin, r)
    end for
    return crossings
end procedure
procedure COUNTCROSSINGS(img, origin, radius)
    crossings ← 0
    for all pixel ∈ BRESENHAMCIRCLE(origin, radius) do
        if crossing detected at pixel then
            crossings ← crossings + 1
        end if
    end for
    return crossings
end procedure
```

A.2 Ordered Bresenham Circle Algorithm

Bresenham's circle algorithm is frequently used to determine which pixels should be selected to draw a circle. algorithm utilizes symmetry of a circle in order to reduce the running time by a factor of eight [1]. For our application with Sholl Analysis, the algorithm must return the points as an ordered set which is defined to be counterclockwise starting from the positive x-axis.

The algorithm calculates points for all eight octants simultaneously. The octants are labeled with integers starting with zero. We label the octants from starting from the x-positive axis in the counterclockwise direction. The labels can be seen in figure 9. Because it is necessary that the points are ordered counterclockwise from the positive x-axis, we modify the algorithm. The algorithm calculates the points from ordered clockwise or counterclockwise from the closest coordinate axis. Hence, we reverse the points calculated for every odd octant and then concatenate their data points.

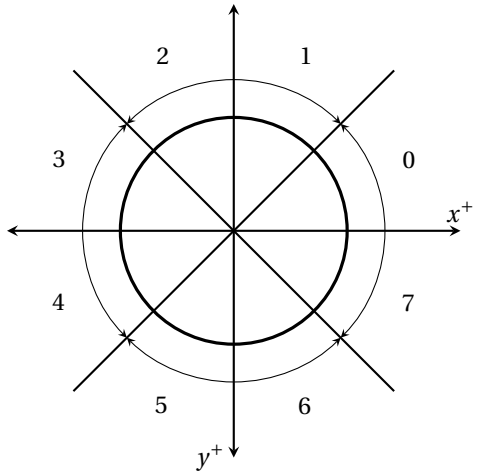


Figure 8: Octants in the Eight-Way Symmetry of a Circle

Algorithm 2 Ordered Bresenham Circle Algorithm

Ensure: points returned are ordered counterclockwise from the positive x-axis

```
procedure BresenhamCircle(origin, radius)
    x ← radius
    y ← 0
    r_error ← 1 − x
    octants ← empty array
    while x ≥ y do
        octants[0] ⇐ (x + origin.x, −y + origin.y)
        octants[1] ⇐ (y + origin.x, −x + origin.y)
        octants[2] ⇐ (−y + origin.x, −x + origin.y)
        octants[3] ⇐ (−x + origin.x, −y + origin.y)
        octants[4] ⇐ (−x + origin.x, y + origin.y)
        octants[5] ⇐ (−y + origin.x, x + origin.y)
        octants[6] ⇐ (y + origin.x, x origin.y)
        octants[7] ⇐ (x + origin.x, y origin.y)
        y ← y + 1
        if r_error < 0 then
            r_error ← r_error + 2y + 1
        else
            x ← x − 1
            r_error ← r_error + 2(y − x + 1)
        end if
    end while
    points ← []
    for all octant ∈ octants do
        if octant is odd then
            reverse the octant
        end if
        for all point ∈ octant do
            points ⇐ point
        end for
    end for
    return points
end procedure
```