Supplementary Information

Supplementary Note 1

The image analysis algorithm has been implemented in C++ and uses Eigen (*1*) for the linear algebra routines, libigl (*2*) for basic geometry processing, and PolyFEM for finite element simulation (*3*). The reference implementation used to generate the results is attached to the submission and will be released as an open-source project.

The input of our algorithm is a chronological sequence of images $I(x, y, z, t)$ acquired by confocal microscopy containing, for each timestamp, a stack of 3D gray-scale images revealing the location of the bleached pattern. The output are time dependent traction stresses σ sampled at the nodes of a triangular mesh representing the bleached pattern. An intermediate result of the algorithm is a sequence of time dependent positions for each vertex V^t of the mesh with faces F, which are used to compute the traction stresses and, optionally, displacement and strains in the BM.

The algorithm (Algorithm 1) has three main components: (1) detection of the markers and lattice mesh generation for the first frame (line 1), (2) mesh warping to fit moved markers on the next time frame (line 18), and (3) volumetric finite element (FE) analysis to compute traction forces (line 26).

At a high level, our algorithm is based on a numerical method designed to find the location and size of a marker by identifying cylinder-like structures θ parametrized by a position θ_{xyz} and a radius θ_r (the height of the cylinder is fixed to 1.0 micrometer) in a 3D image volume (Figure 1). Our algorithm requires an initial position and radius which we sample with n_x, n_y, n_z, and n_r uniform samples (line 5). In our implementation, by default, we sample 10000 xy locations for every z-slice, and for each location we evaluate the energy for $\theta_r = 3, 4, 5, 6$ pixels. The initial uniform sampling narrows down the possible locations for θ; to further refine the position and radius we minimize an energy function using a quasi-Newton method (line 8, Section 2). Note that, since the quasi-Newton method requires the image to be differentiable; in the first step we fit a quadratic B-spline to every frame's 3D image (lines 2 and 19, Section 1), which we use as a proxy for the original image data. We then cluster the optimized cylinders Θ (line 12, Section 3): the clustered cylinders will correspond to regions with a high density of candidate positions. Since the optimization (line 8, Section 2) refines only the xy-location and the radius, we run another optimization to find the optimal z-location of every clustered cylinder (line 13). Finally, we fit a regular lattice mesh to the detected markers (line 14) using an iterative closest point algorithm. The result is the optimal cylinders Θ and a regular lattice F connecting them.

We track and refine the previously optimized cylinder over different time frames. To estimate the new location of the cylinders, we use optical flow (line 20, Section 5) to avoid repeating the computationally expensive uniform sampling in the first frame. To further increase the accuracy of the result estimated by optical flow, we optimize the moved cylinder (line 22). The result is a new cylinder set Θ optimized for the new time frame. Note that we do not need to compute a new lattice as all frames share the same one. This procedure is repeated for all the

Algorithm 1 Overview of our algorithm to compute traction stresses.

function INITIALIZATION($I(x, y, z, t)$)
  $S^0(x, y, z) \leftarrow$ FIT_SPLINE($I(x, y, z, 0)$)
  $w, h, d \leftarrow$ IMAGE_SIZE(I)
  $\Theta \leftarrow \emptyset$
  for $x \in [0, \ldots, n_x], y \in [0, \ldots, n_y], z \in [0, \ldots, n_z], r \in [0, \ldots, n_r]$ **do**
    $\theta \leftarrow (w/x,\ h/y,\ d/z,\ 3 + 4/r)$
    if $E(\theta, S^0) < \varepsilon_g$ **then**
      $\theta \leftarrow$ FIT_CYLINDER_XYR(θ, S^0)
      $\Theta \leftarrow \Theta \cup \theta$
    end if
  end for
  $\Theta \leftarrow$ CLUSTER(Θ)
  $\Theta \leftarrow$ FIT_CYLINDER(Θ, S^0)
  $F \leftarrow$ MESH_ICP(Θ_{xyz})
  return Θ, F
end function

function NEXT_FRAME($I(x, y, z, t), \Theta^{t-1}, t$)
  $S^t(x, y, z) \leftarrow$ FIT_SPLINE($I(x, y, z, t)$)
  $d \leftarrow$ OPTICAL_FLOW($I(x, y, z, t-1), I(x, y, z, t)$)
  $\Theta \leftarrow (\Theta^{t-1}_{xyz} + d,\ \Theta^{t-1}_{r})$
  $\Theta \leftarrow$ FIT_CYLINDER(Θ, S^t)
  return Θ
end function

function ANALYSIS($I(x, y, z, t), R, r_x, r_y, r_z$)
  $\Theta^0, F \leftarrow$ INITIALIZATION($I(x, y, z, t)$)
  $V^0 \leftarrow \Theta^0_{xyz}$
  $\sigma \leftarrow \emptyset$
  for $t \in [1, \ldots, t_{\max}]$ **do**
    $\Theta^t \leftarrow$ NEXT_FRAME($I(x, y, z, t), \Theta^{t-1}, t$)
    $V^t \leftarrow$ REMOVE_GLOBAL_DISPLACEMENT(V^0, Θ^t_{xyz}, R)
    $\sigma \leftarrow \sigma \cup$ FEM($V^0, F, V^t - V^0, r_x, r_y, r_z$)
  end for
  return σ
end function

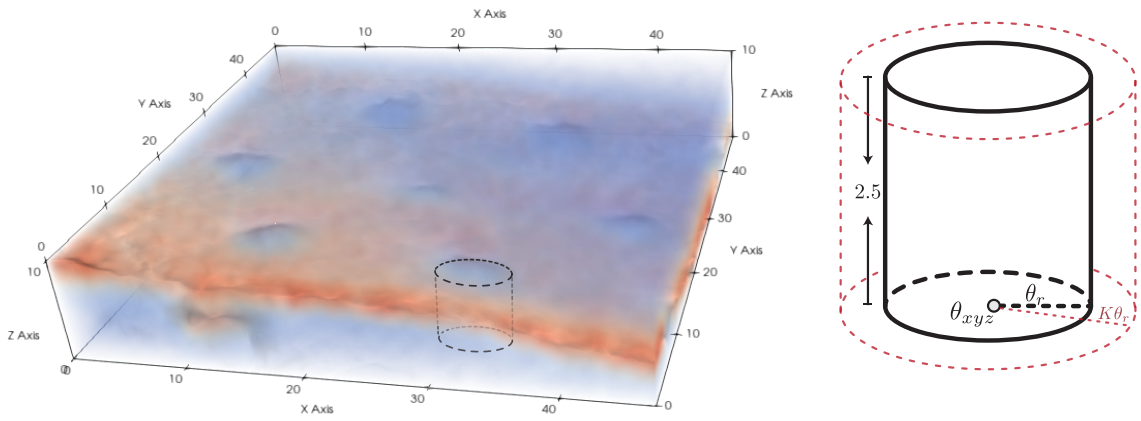

Fig. 1. Example of an image I with a bleached marker and its corresponding cylinder θ.

time-frames of the sequence (line 30), obtaining a 3D reconstruction of the BM.

Before solving the FE analysis to reconstruct the traction stresses that induced the observed displacement (line 33, Section 6), we remove the global displacement of the BM using a reference region R specified by the user (line 32). To compute traction stresses, we create a volumetric tetrahedral mesh of the image volume, and use the tracked mesh as a Dirichlet boundary condition for an elastic deformation, from which we can extract the traction stresses required to induce such a deformation.

1 Spline Approximation

The input image I is normalized with a maximum cutoff brightness value to remove bright outliers (we use the 99.5% quantile of the first frame). As our algorithm requires the computation of first derivatives of the image, we convert each time-frame t into a 3D quadratic B-spline $S^t(x, y, z)$. We set the number of control points in each dimension as 70% the number of pixels in that dimension to avoid overfitting the image. We expose this parameter to the user to allow lowering the control point density for stronger smoothing effect and faster computation. To compute the spline's control c^t_{ijk} points we solve

$$\min_{c^t_{ijk}} \|I(x_i, y_i, z_i, t) - \sum_i \sum_j \sum_k c^t_{ijk} B_i(x_i) B_j(y_i) B_k(z_i)\|^2,$$

with x_i, y_i, z_i the voxel coordinates of I in least square sense using the conjugate gradient algorithm with inverse diagonal preconditioner. To speedup the convergence we use the closest voxel as the initial guess for c^t_{ijk}. Figure 2 shows how the image I is smoothed with different numbers of control points.

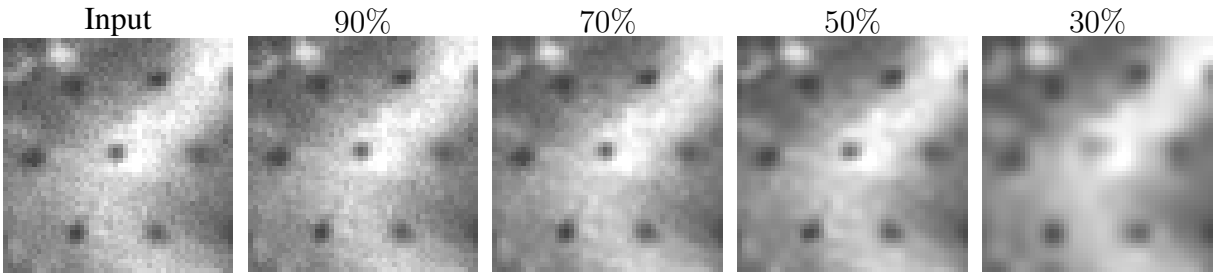

Fig. 2. Example of B-spline smoothing for different number of control points.

2 Markers Parametrization and Optimization

Marker Parametrization The BM marks have an approximately cylindrical shape, which we parameterize as (x, y, z, r) and the height is fixed to be 2.5 pixels, or roughly 1.0 micrometer in our case. $(x, y, z) \in \mathbb{R}^3$ is the coordinate of the bottom center of the cylinder and $r \in \mathbb{R}$ is the radius (Figure 1). Due to the protocol we use for bleaching and the physiology of the zebrafish embryo, we know that a cylinder θ needs to satisfy a set of properties:

1. It is not too close to the image boundary. In particular, a larger cylinder at position θ_{xyz} whose radius is $K\theta_r$ must be completely inside the image.
2. The radius θ_r is larger than 2 pixels, which correspond to 0.65 micrometer in our setup.
3. The marker is in the volume of the basement membrane, which we detect by enforcing the surrounding of a marker to be bright. We also allow the user to optionally provide a binary mask m identifying the membrane.

These properties help us filter out other features of the image that resemble cylinders, such as random tiny speckles. We define a cylinder that satisfies all these properties as *valid*.

Energy Function The energy function characterizes an ideal cylinder by considering the sharp brightness change inside the cylinder and in the circular region with radius $K\theta_r$ outside it (Figure 1). It is defined as the weighed subtraction of the density of the inner part and outer part of a cylinder θ for an interpolated image S

$$E(\theta, S) = (1 - \alpha)m_c(\theta, S)/V(\theta) - \alpha m_a(\theta, S)/V_a(\theta)$$

where

$$V(\theta) = 2.5\theta_r^2\pi \qquad V_a(\theta) = 2.5\theta_r^2\pi(K^2 - 1),$$

$$m_c(\theta, S) = \int_0^{2.5} \int_0^{\theta_r} \int_0^{2\pi} S(\theta_x + \ell \sin(\beta),\ \theta_y + \ell \cos(\beta),\ \theta_z + h)\, \mathrm{d}\beta\, \mathrm{d}\ell\, \mathrm{d}h,$$
$$m_a(\theta, S) = \int_0^{2.5} \int_{\theta_r}^{K\theta_r} \int_0^{2\pi} S(\theta_x + \ell \sin(\beta),\ \theta_y + \ell \cos(\beta),\ \theta_z + h)\, \mathrm{d}\beta\, \mathrm{d}\ell\, \mathrm{d}h,$$

and $\alpha \in [0, 1]$ is a hyper-parameter. Intuitively, the ratio K controls how large is the peripheral annulus and the scalar α controls how severe we penalize the annulus. The default radius ratio K is $\sqrt{2}$ and α is set to be $1/2$ and, we did not modify them throughout this project.

Numerical Integration. The evaluation of the energy function entails the computation of cylindrical numerical integration in m_c and m_a. Since the integration along the height of the cylinder is independent with respect to the one in the cylinder's plane, by virtue of the Fubini's theorem, we decompose the cylindrical integration into a disk integral and a 1D line integral.

The disk integration is computed with the Cools-Kim circular quadrature rules of order 17 (*4*), extracted from the library (*5*). We experimentally found that order 17 provides the optimal compromise between number of quadrature points and accuracy. We compared different quadrature order with a reference order of 59, and discovered that at order 15 the difference is around 10^{-4} which is acceptable; we added 2 orders for safety.

The 1D integration in the z-direction is computed with a standard Gaussian quadrature of order 7, which we choose as it has 4 sample points, that is roughly one sample every half pixel.

Algorithm 2 Minimization of the energy.

function FIT_CYLINDER_XYR(θ, S)
  $H \leftarrow$ Id
  for $i \in [1, \ldots, 50]$ **do**
    $d \leftarrow$ LINE_SEARCH($E(\theta, S, \nabla_{xyr} E(\theta, S), H)$
    $\theta = \theta - d$
    if $\|d\| < \epsilon$ **then**
      break
    end if
    $H \leftarrow$ UPDATE_HESSIAN(H, θ, d)
  end for
  return θ
end function

Minimization. We optimize the energy function with the LBFGS Quasi-Newton algorithm, using the library (*6*) (Algorithm 2). The gradients are computed using automatic differentiation (*7*). The optimization stops at either 50 LBFGS iterations, or when the norm of the gradient is smaller than 10^{-4} times the maximal entry in the current solution (`optimEpsilon` parameter in lbfgspp). The optimization uses backtracking line search with Armijo conditions. The only

modification to the standard LBFGS is in the line search (line 4): we reject a step if the resultant cylinder is invalid.

Algorithm 3 Algorithm to find the optimal depth.

1: **function** FIT_CYLINDER(θ, S)
2: $\theta \leftarrow$ SEARCH_Z($\theta, S, 4, 0.2$) ▷ first round: resolution is 0.2 pixel
3: $\theta \leftarrow$ SEARCH_Z($\theta, S, 0.2, 0.01$) ▷ first round: resolution is 0.01 pixel
4: **return** θ
5: **end function**
6:
7: **function** SEARCH_Z(θ, S, o, δ)
8: $\Theta \leftarrow \emptyset$, $E \leftarrow \emptyset$
9: **for** $z \in [\theta_z - o : \delta : \theta_z + o]$ **do**
10: $\bar{\theta} \leftarrow (\theta_x, \theta_y, z, \theta_r)$
11: $\bar{\theta} \leftarrow$ FIT_CYLINDER_XYR($\bar{\theta}, S$)
12: **if** $\|\theta_{xy} - \bar{\theta}_{xy}\| \leq 4$ **then**
13: $\Theta \leftarrow \Theta \cup \bar{\theta}$, $E \leftarrow E \cup E(\bar{\theta}, S)$
14: **end if**
15: **end for**
16: $E_s \leftarrow \emptyset$
17: **for** $i \in [2, \ldots,$SIZE(E)$-2]$ **do**
18: $t \leftarrow$ SORT($E[i-2, i+2]$)
19: $E_s \leftarrow E_s \cup (t[1] + t[2] + t[3])/3$ ▷ average excluding largest and smallest
20: **end for**
21: $i \leftarrow \arg\_\min(E_s)$
22: **return** $\Theta[i]$
23: **end function**

Fit Depth. We do not optimize the depth or height (z-coordinate) in the quasi-Newton minimization step, since the optimization is slow and there would be a large number of local minima if the height dimension is included. To overcome this limitation we design an algorithm that first samples the z-direction then fits x, y, and r for every sample (Algorithm 3). We start by uniformly sampling the depths centered at the current depth (line 9). First we sample with an offset o of 4 pixels at a distance $\delta = 0.2$ pixels (line 2), then we do a second pass at higher resolution ($o = 0.2, \delta = 0.01$, line 3). For each depth, we minimize its energy (line 11) and, if the resultant (x, y) is over 4 pixels away from the original one, we ignore that depth (line 12).

The result of the z-sampling is a set of cylinders along the z-direction and their optimized energy. To find the optimal z-position we locate the minimum of the energy curve (line 21) of a cylinder moving across the depth (Figure 3). Intuitively, as the cylinder "moves" over its correct position, the energy first decreases to the minimum and then increases when the

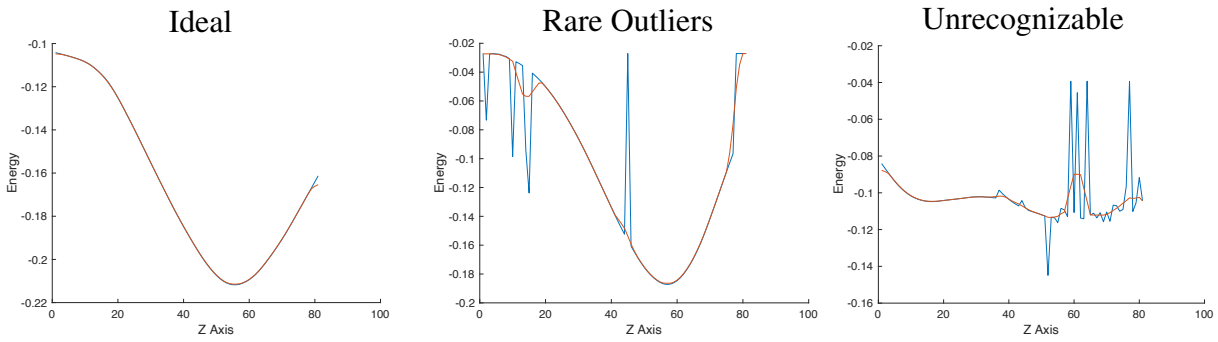

Fig. 3. Example of energy E (blue) profiles along the z-direction, Smoothing the curves (orange) successfully removes outliers and makes finding the minimum easy.

cylinder moves further. To reduce the interference of random outliers, we smooth the energy along the z-direction by a mean filter, excluding the two extrema, applied with a window size of 5 (line 17).

In practice, most of the markers can be properly optimized as long as the quality of the image is reasonable. We discovered that the image regions where the algorithm struggles usually correspond to region that are impossible to annotate even for a human expert.

3 Filtering and Clustering.

We filter and merge the optimized cylinders to finalize the 3D locations of the markers in the first frame. For every bleached marker, there should be only one cylinder remaining after this step. To filter and group the cylinders Θ, we (1) remove outliers, (2) create clusters, (3) filter clusters, and (4) merge each clusters in a *single* cylinder.

Remove outliers. The optimized position from the initial uniform sampling might contain several outliers. To exclude them we design three heuristics based on the current protocol. Note that, the heuristic can be changed in the UI if the bleached pattern differs from ours. We first exclude any cylinder whose energy is larger than ε_E, which we pick as -0.1 and whose radius is larger than $\varepsilon_r = 6$. The energy function E (Section 2) is smooth and for a normal configuration, the LBFGS usually converge in no more than 10 iterations. We exploit this consideration to exclude any cylinder that took more than 49 iterations to converge. Finally, since the markers are bleach marks on the BM, we exclude cylinders whose location are not on the BM.

Create cluster. In our experiments we noticed that if there is a marker nearby, most of the sampled cylinders converge to the marker's location with the similar correct radii. Exploiting this observation we cluster the cylinders whose distance in the xy-plane is smaller than 0.01 pixel. This operation is done independently for every depth layer.

Filter clusters. We observe that the number of cylinders is a good metric to estimate how reliable a cluster is. Thus we exclude clusters based on their size; in our experiment we exclude any cluster with less than 4 cylinders per unit area. In case there exists a wrong large cluster, our tool provides a handy functionality allowing the user to manually delete clusters by clicking on them. After this step, all the remaining groups must represent true markers.

Depth clustering. Since we can now trust the clusters, we merge all the clusters with varying depths but similar xy-coordinate to one large cluster. For every large cluster, we project it into a unique cylinder by averaging the positions and radii of its compositing cylinders.

4 Lattice Mesh Generation

Iterative Closest Point (ICP) matches two point clouds by performing translational and rotational transformations (*8*). In order for ICP to work, prior knowledge of the reference point cloud is necessary. By default, the program uses a regular triangular-lattice as the reference set to find the best match with the detected marker locations.

Two challenges lie in the way of ICP matching: The first is that the detected cylinder pattern could be distorted. The second is that the edge length in the reference pattern needs to be determined because ICP cannot handle scaling. The program automatically estimates the edge length by averaging the distance of all markers to its closest marker. Aside from that, the user can manually translate, rotate, distort and scale the reference pattern.

5 Mesh Warping

We compute the Horn-Schunck (*9*) optical flow vector field between every two consecutive frames to estimate the new position of the markers. Optical flow computes the optimal 3D smooth vector field V to transform $I(x, y, z, t)$ to $I(x, y, z, t + 1)$. The smoothness can be controlled by a parameter α. We run the algorithm for 30 iterations, with a default smoothness factor $\alpha = 0.1$. We move every cylinder θ^t at time t using V (interpolated across voxels) to their new position at $t + 1$. As aforementioned, the final position of θ^{t+1} is further refined (Section 2) to obtain the optimized new location. To handle any potential optimization or tracking failures, the user can manually move the markers in any frame by selecting and dragging them with the mouse. It is worth noting that this functionality should only be used to guide the optimization to the correct direction. The user can initiate the marker optimization again after manually correcting the loccation of an incorrectly tracked cylinder.

6 Traction Stress Computation

The traction stresses are computed using a finite element (FEM) simulation. To compute such simulation we need to: (1) generate a tetrahedral mesh and (2) setup a material model and boundary conditions. The result is a displacement field defined everywhere in the mesh that can be used to compute the traction stresses.

We generate the tetrahedral mesh using TetGen (*10*) by extruding the BM mesh (V_0, F) along the positive and negative z-direction by one half the diagonal length of the mesh. To increase the accuracy of the simulation we generate a mesh with higher density than F by specifying the area parameter to TetGen. By default, the mean edge length of the tetrahedral mesh approximates the edge length of the BM mesh as if we upsample it twice.

We setup the problem by imposing zero Dirichlet boundary condition on the top/bottom flat side and the BM's displacement for the middle part. Note that, since the BM mesh is refined, we extend the displacement of the BM using radial basis function interpolation with thin-plate $r^2 \log(r)$ as radial function. To accurately compute the displacement in the volume, we approximate the FEM solution with quadratic tetrahedra. We use a linear material model

$$\text{div}(\sigma[u]) = 0,$$

where

$$\sigma[u] = 2\mu\epsilon[u] + \lambda \text{tr}(\epsilon[u])I \qquad \epsilon[u] = \frac{1}{2}\left(\nabla u^T + \nabla u\right),$$

with $\lambda = (E\nu)/((1+\nu)(1-2\nu))$, and $\mu = E/(2(1+\nu))$.

By solving the aforementioned equation for u we obtain a displacement in the whole volume of the cube, from which we compute the traction stresses on the BM as

$$F = \sigma[u]n,$$

where n is the normal of the face of a triangle on the BM.

References

1. Guennebaud, G., Jacob, B. *et al.* Eigen v3 (2010). URL http://eigen.tuxfamily.org.

2. Jacobson, A., Panozzo, D. *et al.* libigl: A simple C++ geometry processing library (2018). Https://libigl.github.io/.

3. Schneider, T., Dumas, J., Gao, X., Zorin, D. & Panozzo, D. Polyfem. https://polyfem.github.io/ (2019).

4. Cools, R. & Kim, K. A survey of known and new cubature formulas for the unit disk. *Journal of Applied Mathematics and Computing* **7**, 477–485 (2000).

5. Schlomer, N. Quadpy (2020). Https://github.com/nschloe/quadpy.

6. Qiu, Y. lbfgspp (2021). Https://lbfgspp.statr.me.

7. Jakob, W. Mitsuba autodiff (2021). Https://www.mitsuba-renderer.org/misc.html.

8. Arun, K. S., Huang, T. S. & Blostein, S. D. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.* **9**, 698–700 (1987). URL https://doi.org/10.1109/TPAMI.1987.4767965.

9. Horn, B. K. P. & Schunck, B. G. Determining optical flow. *Artif. Intell.* **17**, 185–203 (1981). URL https://doi.org/10.1016/0004-3702(81)90024-2.

10. Si, H. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.* **41** (2015). URL https://doi.org/10.1145/2629697.

Supplementary Table 1

Supplementary Table 1

Genotyping PCR primers

lamC1(sa9866)		
	Forward outer primer (5'-3')	TTCAGTTCATCGGGTTGC
	Reverse outer primer (5'-3')	GTAACAGTTAGGGCACTGC
	Forward inner primer (5'-3')	GTTTTCCTGCGTTGACGCTT
	Reverse inner primer (5'-3')	GGTGTCGAGCGGTTGTAGAA
tln1(d4)		
	Forward outer primer (5'-3')	CAAGTGGCTCCGCCTGTACT
	Reverse outer primer (5'-3')	ATAGGCCTAAAGGTATGCCAGC
	Forward inner primer (5'-3')	GAGTAGCAGTGGCACAGTCC
	Reverse inner primer (5'-3')	TGATGGACTCACGCTGGC
tln2a(i23)		
	Forward primer (5'-3')	CAGTTTGAGCCCTCAACGGCTGTATATGACGCATGCCcGGG
	Reverse primer (5'-3')	CCCATATTCTGAAGCTGAGG
tln2b(d10)		
	Forward outer primer (5'-3')	CAGGTGACCCCATAGACACG
	Reverse outer primer (5'-3')	TGCATTGGTCACCTCTCCAG
	Forward inner primer (5'-3')	TGTCCAAGGGTGTGAAGCTG
	Reverse inner primer (5'-3')	CCTCTCCAGACGTGGGCTC
itgb1a(d34)		
	Forward outer primer (5'-3')	GAGTTTCTGAAGCAGGGAG
	Reverse outer primer (5'-3')	ATGGTGTTGCTTTCACACGC
	Forward inner primer (5'-3')	AAAGAGGCTGCGCAGAAGAT
	Reverse inner primer (5'-3')	TTTCTGAGGCTGGATCTGCG
itgb1b(i70)		
	Forward outer primer (5'-3')	GATTGGACGCCGGGTATGTC
	Reverse outer primer (5'-3')	AAACAGGCTGGAACTCCTCG
	Forward inner primer (5'-3')	GGAATGTCACTCGTCTCC
	Reverse inner primer (5'-3')	CATGGTGTAAATGTTATCCTGC

CRISPR-Cas9 primers

The chimeric gRNA backbone primer (5'-3')

gttttagagctagaaatagcaagttaaaataaggctagtccgttatcaacttgaaaaagtggcaccgagtcggtgcggatc

tln1 (tln1 target sequences are underlined)

gRNA1 (5'-3')	TAATACGACTCACTATAACGACGCCTGTCGAATCATCgttttagagctagaaatagcaag
gRNA2 (5'-3')	TAATACGACTCACTATAACAGGCGTCGTACACCACTGgttttagagctagaaatagcaag
gRNA3 (5'-3')	TAATACGACTCACTATAGGCGCTGTCGCTGAAGATCGgttttagagctagaaatagcaag
gRNA4 (5'-3')	TAATACGACTCACTATATCTCTCCCTGATGATTCGACgttttagagctagaaatagcaag

tln2a (tln2a target sequences are underlined)

gRNA1 (5'-3')	TAATACGACTCACTATACATGCCGGGTCATCAGAGAGgttttagagctagaaatagcaag
gRNA2 (5'-3')	TAATACGACTCACTATATGTCTCTGAAGATCTGTGTGgttttagagctagaaatagcaag
gRNA3 (5'-3')	TAATACGACTCACTATACATGCAGTTTGAGCCCTCAAgttttagagctagaaatagcaag
gRNA4 (5'-3')	TAATACGACTCACTATAGAACCCTCTCTCTGATGACCgttttagagctagaaatagcaag

tln2b (tln2b target sequences are underlined)

gRNA1 (5'-3')	TAATACGACTCACTATAGGCCCCTGTGAGGGTCCTCGgttttagagctagaaatagcaag
gRNA2 (5'-3')	TAATACGACTCACTATAAGCCGCGAGGACCCTCACAGgttttagagctagaaatagcaag
gRNA3 (5'-3')	TAATACGACTCACTATAAGAACCTGATGGGAGCCGCGgttttagagctagaaatagcaag
gRNA4 (5'-3')	TAATACGACTCACTATACAAGGGTGTGAAGCTGCTGGgttttagagctagaaatagcaag

itgb1a (itgb1a target sequences are underlined)

gRNA1 (5'-3')	TAATACGACTCACTATACCGGTGACCAACCGCAAGAAgttttagagctagaaatagcaag
gRNA2 (5'-3')	TAATACGACTCACTATAGAGAATCCTGAAGAATACACgttttagagctagaaatagcaag
gRNA3 (5'-3')	TAATACGACTCACTATAGATAAGATCGAGAACCCGCAgttttagagctagaaatagcaag

itgb1b (itgb1b target sequences are underlined)

gRNA1 (5'-3')	TAATACGACTCACTATACGTCATGCTCATGAGCTGAGgttttagagctagaaatagcaag
gRNA2 (5'-3')	TAATACGACTCACTATATGTAAATGTTATCCTGCAGAgttttagagctagaaatagcaag
gRNA3 (5'-3')	TAATACGACTCACTATAGCATCGTGCTTCCTAATGACgttttagagctagaaatagcaag

Whole mount in situ hybridization PCR primers

tln1

Forward (5'-3')	ATGGTACGGGGGCTGGAGAG
Reverse (5'-3')	ACCGCGCGAGCAGCAGCAGC

tln2a

Forward (5'-3')	TCCGGTATGTCAGGAGCAGC
Reverse (5'-3')	GGTTTCAACTGTCCCTCAGA

tln2b

	Forward (5'-3')	TCGACTCCGCTCTCAGTGCT
	Reverse (5'-3')	AATACTAATACGACTCACTATAGCACAAGCAGTTTCTTACTGG
itgb1a		
	Forward (5'-3')	GAAGCGGGAGAATCCAGAGG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGTCCATGGTCTTGACGACGTG
itgb1b		
	Forward (5'-3')	CCTACGTCTCCCACTGCAAG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGATTCGCACGTTCCACAAACG
itgb1b.1		
	Forward (5'-3')	AAAACCCCTGTTTTCCAAGCG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGCTCCGTTCTTGCAGTGGGAG
itgb1b.2		
	Forward (5'-3')	ATGTACTGAGCTTGACGGACG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGTGGACACACCATCAGGTAGC
itgb2		
	Forward (5'-3')	ATCCCCAAATCTGCAGTCGG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGCGTCGCATTCACAGTGTTCG
itgb3a		
	Forward (5'-3')	TCTGGGCAATAATCTGGCCG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGCGTGCCAACTGAAGGGTAGT
itgb3b		
	Forward (5'-3')	TCCAACCAGCAAAATGCACG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGATTTTGTCCTTGCACACGGC
itgb4		
	Forward (5'-3')	ACAATTTAGAATCGCGCTTCACC
	Reverse (5'-3')	AATACTAATACGACTCACTATAGCGTTGGGTTTTCGGGGTTTC
itgb5		
	Forward (5'-3')	GTCACCCGCTGTGGAAGGATG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGATAGCGAGAGGTCCATGAGGTAG
itgb6		
	Forward (5'-3')	AAGATGCGCCTCCAGCTTAG
	Reverse (5'-3')	AATACTAATACGACTCACTATAGGCTTCATGGAGTCGTTTCGC
itgb7		
	Forward (5'-3')	CATGTGCAGCTGTGACGAAG

	Reverse (5'-3')	AATACTAATACGACTCACTATAGTCTCATGACAGGAGCCGCTAC

itgb8

Forward (5'-3')	GCCACCTAGAGGACAACGTC
Reverse (5'-3')	AATACTAATACGACTCACTATAGGTAGTGCAGGACGAGGGTTC