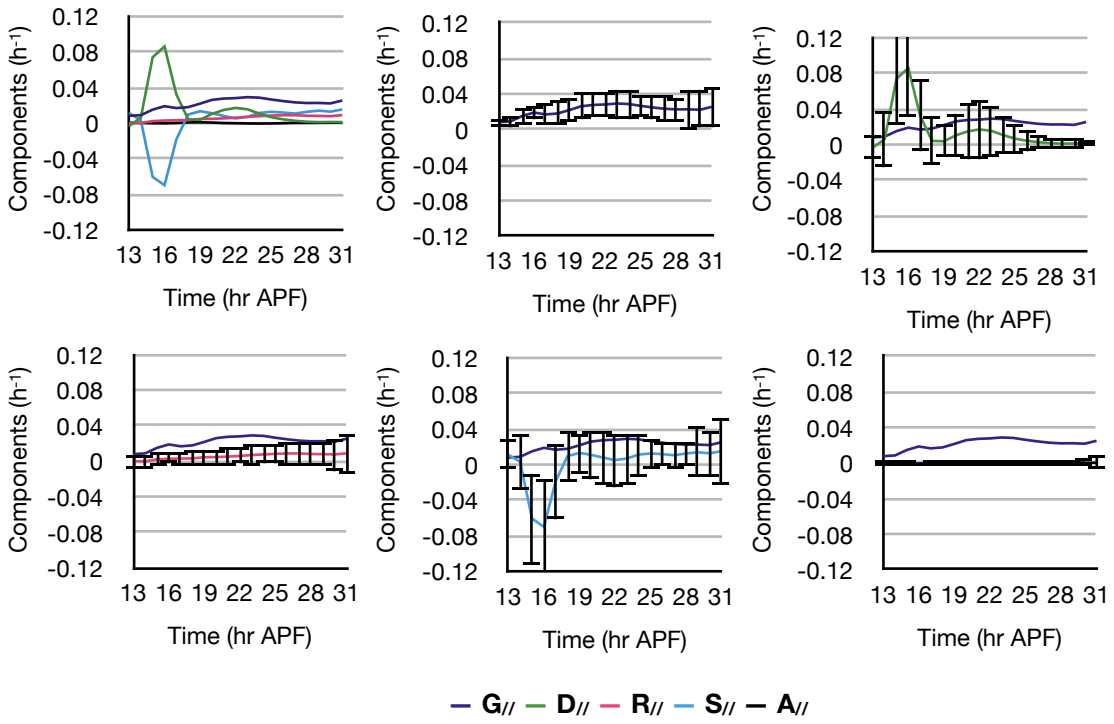


Fig. S1. Variance of effective contribution of cellular processes at each time point. Plots show time evolution of effective contribution of cellular processes in the *Drosophila* notum and standard deviation of them.

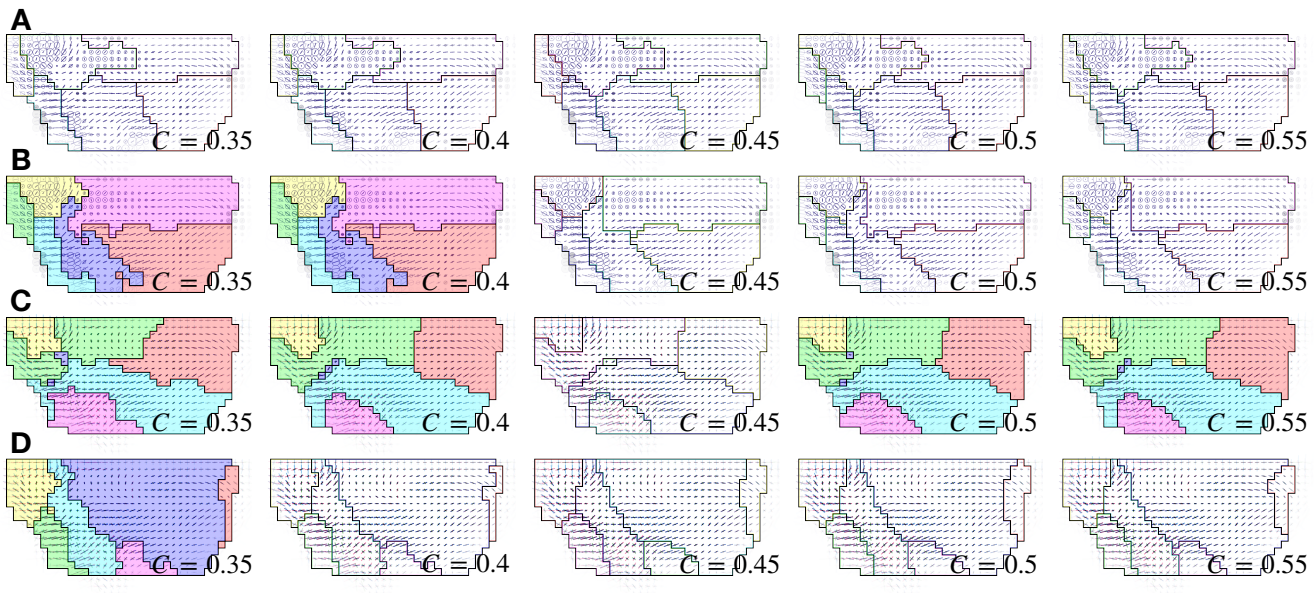


Fig. S2. Boundary smoothing with various minimum circularities. The *Drosophila* no-tum was divided based on time-average tissue deformation rate (**A**), time-evolution of tissue deformation rate (**B**), time-average cellular processes effective contributions (**C**), and time-evolution of cellular processes effective contributions. They were smoothed with the minimum circularity C ranging from 0.35 to 0.55. Some of them were colored for visibility.

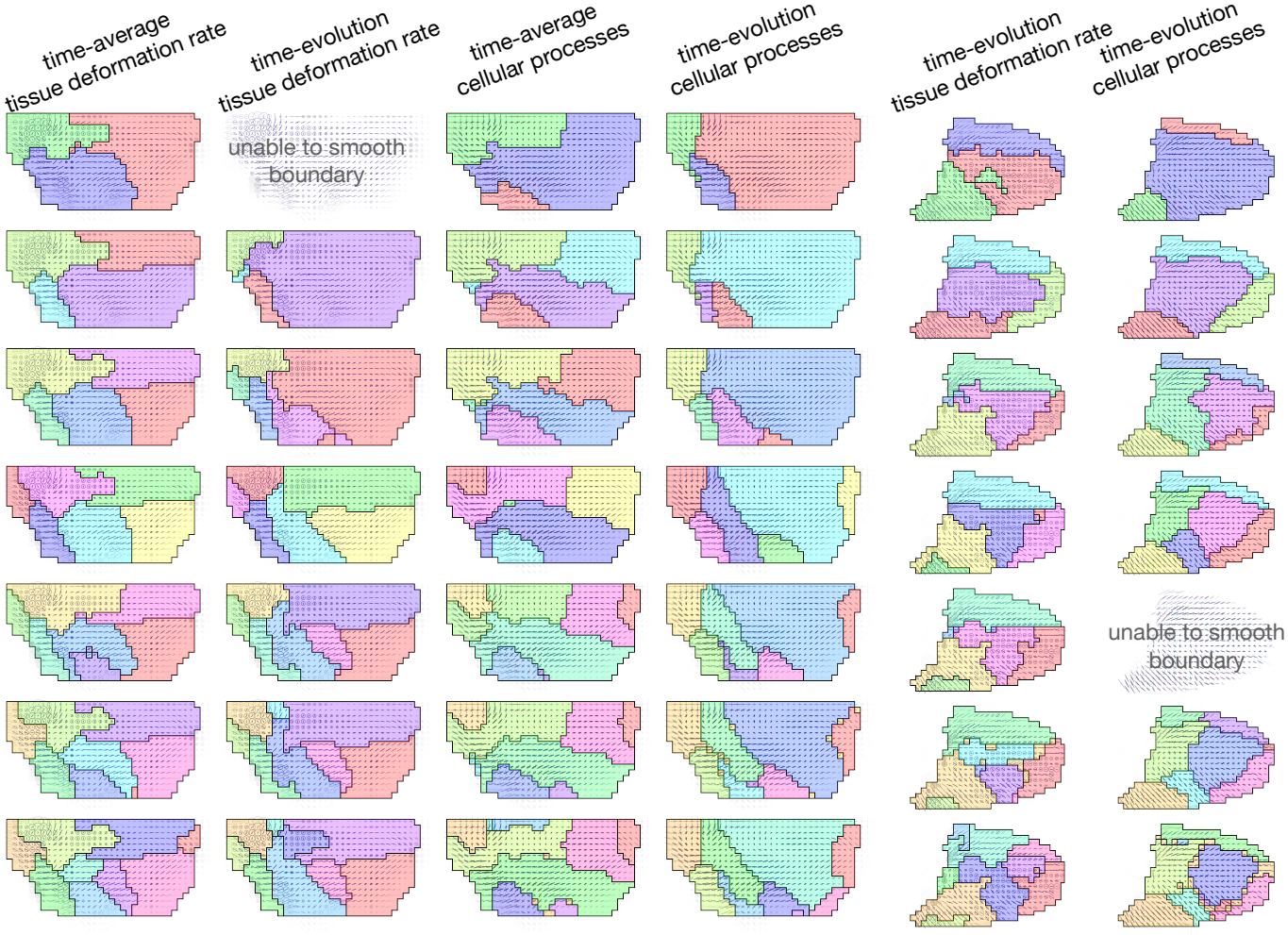


Fig. S3. Segmentations in different number of regions. Dorsal thorax (first to fourth columns) and wing blade (fifth and sixth columns) were divided into 3-9 regions. First column: segmentations based on time-average tissue deformation rate. Second column: segmentations based on time-evolution of tissue deformation rate. Third column: seg-mentations based on time-average cellular processes effective contributions. Fourth col-umn: segmentations based on time-evolution of cellular processes effective contributions. Fifth column: segmentations based on time-evolution of tissue deformation rate. Sixth column: segmentations based on time-evolution of cellular processes effective contribu-tions. The tissues were divided into 3 to 9 regions (from top to bottom rows). The regions were colored for visibility. When the number was too large and a result of the initial label propagation included a too small region, the small region tended to disap-pear in the cellular Potts model smoothing, and thus the final label propagation tried to integrate regions fewer than the final segmentation, sometimes resulted in undesired dis-connected regions (third column bottom row and fourth column sixth row). For dividing the dorsal thorax into three regions based on time-evolution of tissue deformation rate and wing blade into seven regions based on time-evolution of cellular processes effective contributions, it failed to screen the parameters (the screening algorithm pursued to a too low temperature which would freeze any change in the cellular Potts model, second column first row and sixth column fifth row).

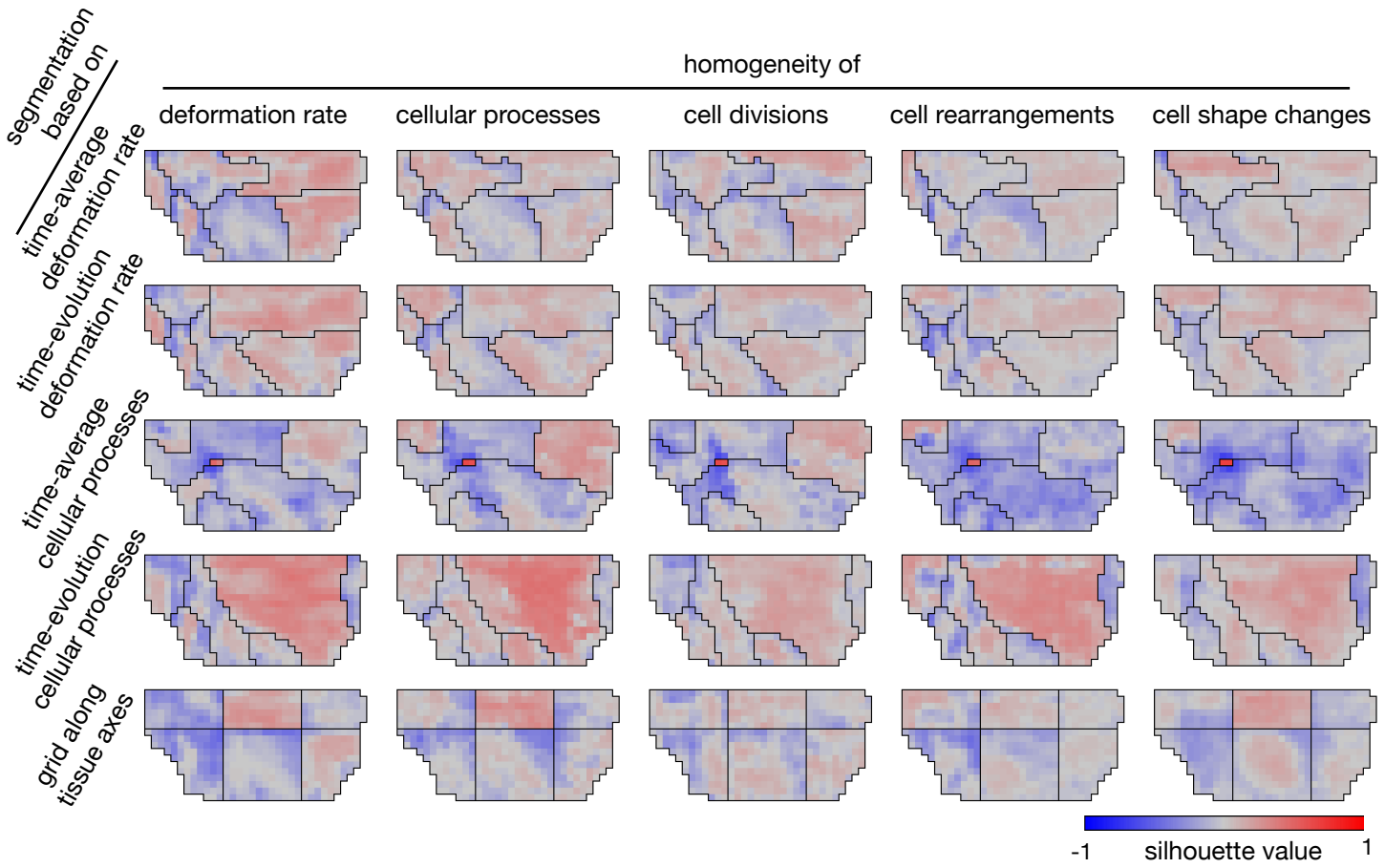


Fig. S4. Heat maps of silhouette value. First row: segmentation based on time-average tissue deformation rate. Second row: segmentation based on time-evolution of tissue deformation rate. Third row: segmentation based on time-average cellular processes effective contributions. Fourth row: segmentation based on time-evolution of cellular processes. Fifth row: conventional segmentation of large grid parallel to tissue axes. First column: silhouette values measured in the property space of time-evolution of de-formation rate. Second column: silhouette values measured by time-evolution of cellular processes. Third column: silhouette values measured by time-evolution of cell divisions. Fourth column: silhouette values measured by time-evolution of cell rearrangements. Fifth column: silhouette values measured by time-evolution of cell shape changes.

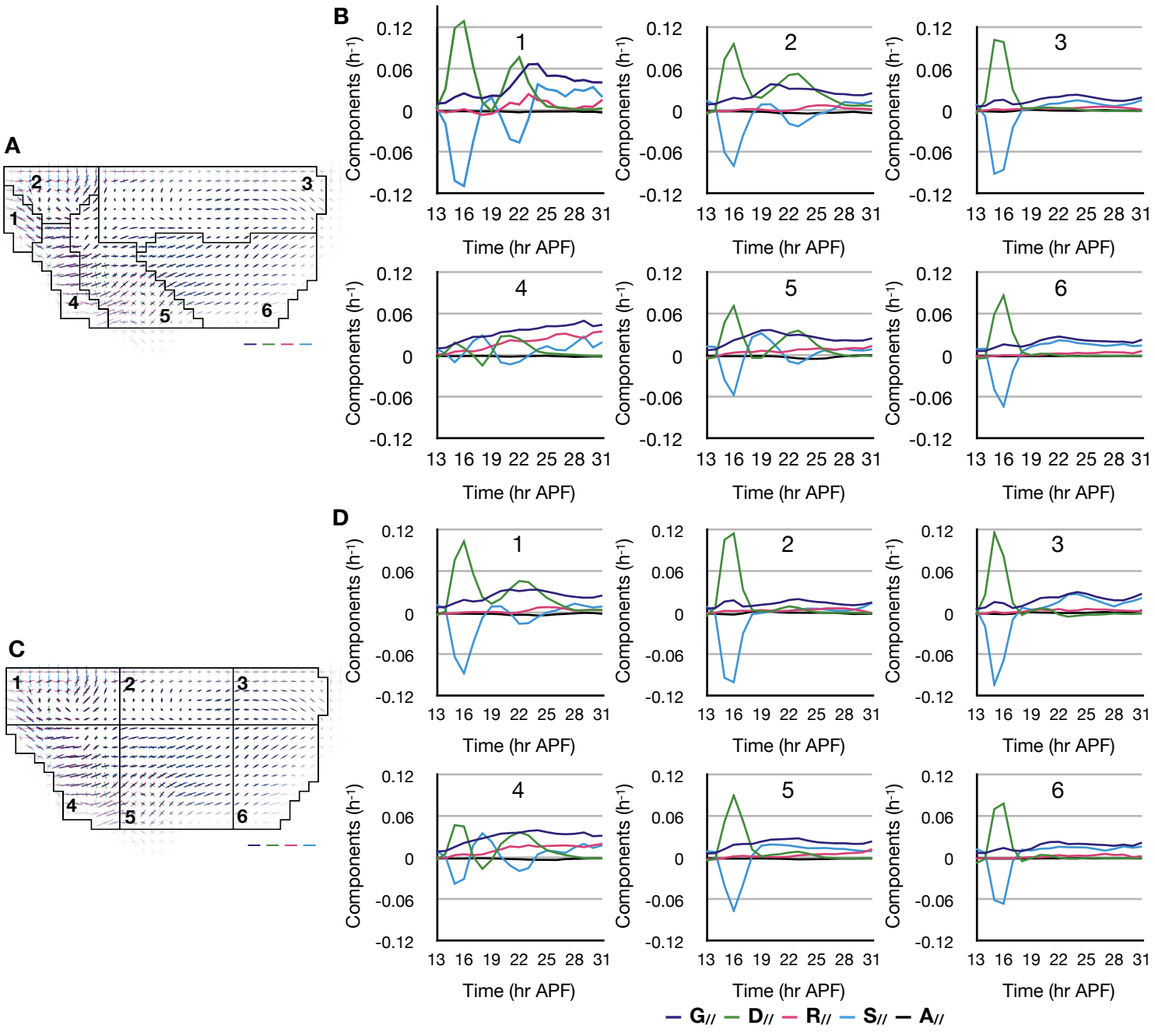


Fig. S5. Plots of cellular processes in the segmentations based on time evolution of tissue deformation rate and the conventional large grid. (**A**, **B**) The tissue segmentation based on time-evolution of tissue deformation rate (**A**) and plots of cellular processes effective contributions averaged in each region (**B**). The numbers indicate the regions. (**C**, **D**) The large grid (**C**) and plots of cellular processes in each region (**D**). Scale bars in **A** and **C** indicate deformation rate 0.02 h^{-1} with colors for tissue and cellular processes.

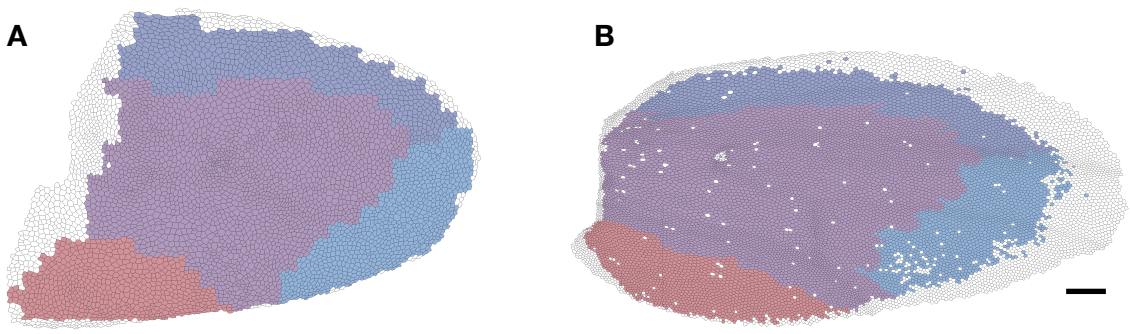
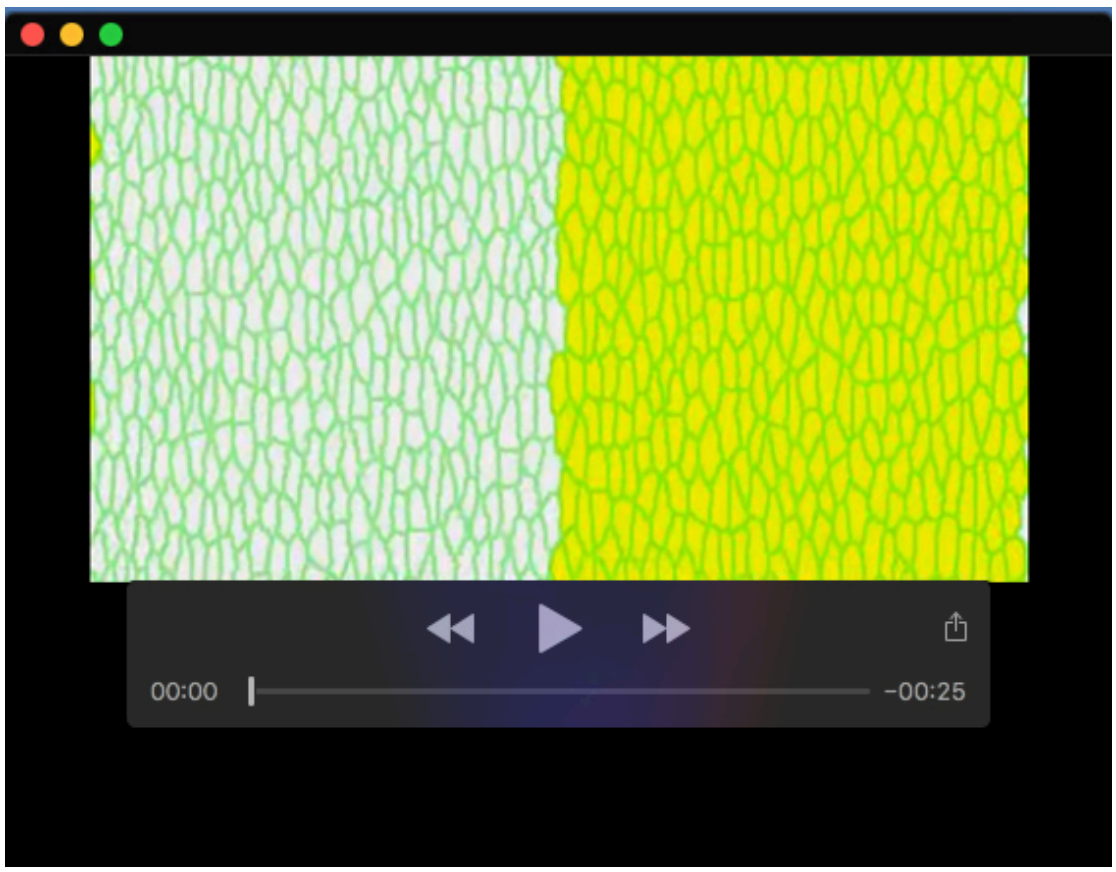


Fig. S6. Projection of the segmentation onto the wing blade cells. The segmentations based on time evolution of cellular processes were projected. (**A**, **B**) The segmentation was projected onto the wing blade cells at 15 hr (**A**) and 32 hr APF (**B**), where the regions were indicated by colors. Scale bars indicate 50 μm.



Movie 1. Cell rearrangements and cell shape changes after tissue compression. The cells with low and high surface tension were colored gray and yellow respectively. The cellular Potts model was run on an image of 480×270 lattice and included 600 cells. The movie is 7 fps and there were 5000 updates between the frames.

Supplementary Materials and Methods

1 Pseudo codes for tissue segmentation algorithms

In below pseudo codes show algorithms of the automatic tissue segmentation. Matlab custom functions and framework developed for this study are available at GitHub (http://doi.org/10.5281/zenodo.3626111). For details of the functions and framework, see its README file and comments in the codes.

1.1 Region growing algorithm

Algorithm 1 shows a pseudo code of the region growing image segmentation in Matlab-like syntax. It divides a bitmap image stored in a data object *dataMap*. In the algorithm, a number of regions, a limit to update the seeds, and a metric are given as parameters. With the parameters, supporting objects *seedList*, *meanList*, *regionsList*, *meter*, and *seeder* are allocated and initialized. The seedList, meanList, and regionsList are instances of data object with a property *var* representing seeds and means of regions and regions, shared among the supporting objects. The meter is an object measuring distance between the mean of region and a point adjacent to the region. A method *measure* returns the distance measured by the given metric. The seeder is an object choosing seeds of regions. Methods *initalSeeds* and *initialMeans* return indices of randomly chosen points and their values. Once the dataMap was divided into regions, methods *newSeeds* and *newMeans* return indices of points at center of the regions and mean values of the regions. A method *initalQueue* returns an array where its element represents a point adjacent to one of the seeds and holds the region and distance to the region's mean value. Inside a loop, a point in the queue with the smallest distance to the region's mean value is added to the region, and points adjacent to the point, returned by a method *neighborsOfPoint* of dataMap, are added to the queue.

In our tissue segmentation, a Matlab custom function *run_region_growing*() iterates this algorithm for given time, returning a stack of resultant partitions.

1.2 Label propagation on a consensus matrix

Algorithm 2 shows a pseudo code of the label propagation. It divides N objects into clusters based on an $N \times N$ consensus matrix M whose rows and columns correspond to the objects, and an element m_{ij} represents the frequency at which the i-th and j-th objects were included in a cluster among given clustering results. A parameter t_M indicates a threshold value, where elements in M smaller than t_M are ignored in the label propagation.

In the tissue segmentation, 50 results of region growing were converted to the consensus matrix and given to a Matlab custom function *run_label_propagation*() implementing the label propagation. The number of resultant regions is influenced by t_M, and thus a Matlab custom function *run_cm_thresholding_lp*() screens t_M values so that *run_label_propagation*() returns the same number of regions with the given partitions.

Algorithm 1: Region growing algorithm

```
input  : dataMap to be segmented and parameters.
% seedList, meanList, regionsList, allocatedList, meter, and seeder are
  supporting objects and variable initialized with the parameters.
seedList.var = seeder.initalSeeds;
meanList.var = seeder.initalMeans;
while loop counter is smaller than limit do
    % Initialize partition, allocated list, queue.
    regionsList.var(:) = false;
    allocatedList(:) = false;
    queue = seeder.initalQueue;
    while queue is not empty do
        point = queue(1);
        if allocatedList.var(point.index) == false then
            % Grow region to the point.
            regionsList.var(point.index, point.region) = true;
            allocatedList(point.index) = true;
            % Enqueue neighbors of the point.
            array = dataMap.neighborsOfPoint(point.index);
            for neighbor in array do
                neighbor.region = point.region;
                neighbor.distance = meter.measure(neighbor);
                queue = cat(1, queue, neighbor);
            % Remove the allocated point from queue.
            queue(1) = [];
            % Sort queue.
            [values, indices] = sort([queue.distance]);
            queue = queue(indices);
        else
            queue(1) = [];
    % Check convergnence.
    lastMeanList = meanList.var;
    seedList.var = seeder.newSeeds;
    meanList.var = seeder.newMeans;
    if isequal(lastMeanList, meanList.var) then
        break;
return regionsList.var
```

Algorithm 2: Label propagation

```
input : Matrix M and threshold t_M.

% Cut elements in M smaller than the t_M.
M(M < t_M) = 0;
% Make labelArray representing labels on N vertices.
labelArray = (1:N)';

flag = true;
while flag do
    flag = false;
    % Enumerate vertices in random order and update their label.
    for i = randperm(N) do
        % Make labelMatrix representing labels on vertices.
        labelMatrix = labelArray == 1:N;
        % Choose label most weighted by edges incident to the i-th vertex.
         ,indices= max(sum(M(:,i) .* labelMatrix),1));
        if labels(i)  = indices(1) then
            % Update label of the i-th vertex.
            labelArray(i) = indices(1);
            flag = true;

% Convert labelArray to a matrix.
labelMatrix = labelArray == 1:N;
indices = any(labelMatrix,1);
partition = labelMatrix(:,indices);

return partition
```

1.3 Cellular Potts model

Algorithm 3 shows a pseudo code of the cellular Potts model. It simulates a deformation of regions (partition of dataMap) by giving small fluctuations. In the algorithm, an array of function handles, coefficients to combine the functions results, the system temperature, and the number of label updates are given as parameters. With the regions and parameters, supporting objects *analyser* and *dict* are allocated and initialized. The functions in the array calculate system energy with *analyser* and *dict*. For each fluctuation, one of points at regions rim returned by analyser *rim_points* is selected randomly, and a label of neighboring points is also selected randomly and copied. Connectedness of a region is checked locally, with a coordinate of neighboring points returned by dataMap *coordinates*.

In the tissue segmentation, a Matlab custom function *run_CPM_smoothing*() implement this algorithm with energy functions combining area constraint, surface tension, and total silhouette value. The coefficients and temperature influence resultant regions, and thus a Matlab custom function *run_CPM_fitting*() screens the parameters so that *run_CPM_smoothin*() returns smoothed regions with a circularity larger than the given value and the total silhouette value as large as possible.

Algorithm 3: Cellular Potts model with region homogeneity

```
input  : Partition, dataMap, and parameters
% regionList, analyser, dict, H_functions, coefficients, T, and counter are
  supporting objects and variables initialized with the parameters.

% Calculate the system energy.
H = 0;
for k = 1:length(H_functions) do
    fh = H_functions(k);
    H = H + fh(analyser,dict) * coefficients(k);

% Update labels for given times.
while true do
    % Select a point randomly.
    rim = analyser.rim_points;
    rim = find(rim);
    if isempty(rim) then
        % There is only one region.
        break;
    i = ceil(rand() * length(rim));
    i = rim(i);

    % Select a label from neighbors of the point.
    neighbors = dataMap.neighborsOfPoint(i);
    j = ceil(rand() * length(neighbors));
    j = neighbors(j);
    if any(regionsList.var(i,:) & regionsList.var(j,:)) then
        % The i-th and j-th points are in a region.
        continue;

    % Check connectedness.
    m = zeros(3,'logical');
    x0 = dataMap.coordinates(i).x - 2;
    y0 = dataMap.coordinates(i).y - 2;
    for k = neighbors do
        x = dataMap.coordinates(k).x - x0;
        y = dataMap.coordinates(k).y - y0;
        m(y,x) = any(regionsList.var(i,:) & regionsList.var(k,:));
    array = m([1,2,3,6,9,8,7,4]);
    brray = m([2,3,6,9,8,7,4,1]);
    if sum(array ~= brray) > 2 then
        continue;

    % Get a change of energy.
    oldLabel = regionsList.var(i,:);
```

```
        regionsList.var(i,:) = regionsList.var(j,:);
        newH = 0;
        k = 1:length(H_functions)
            fh = H_functionsk;
            newH = newH + fh(analyser,dict) * coefficients(k);
        dH = newH - H;
        % Adapt the change when possible.
        p = exp(-dH / T);
        if p > rand() then
            H = newH;
            counter = counter - 1;
            if counter < 1 then
                break;
        else
            regionsList.var(i,:) = oldLabel;
    return regionsList.var
```