Supplementary Information

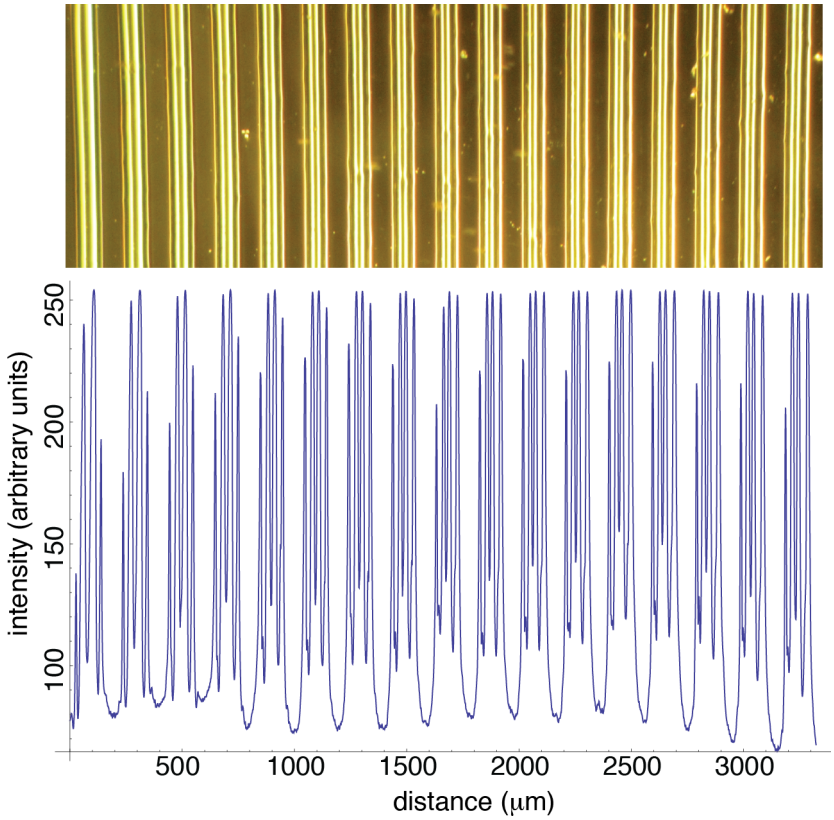


Figure S1: Substrates have less than 4% variability in measured features. Top: XY image of substrate under phase microscopy. Bottom: Intensity profile of micrograph used to measure interfeature distance.

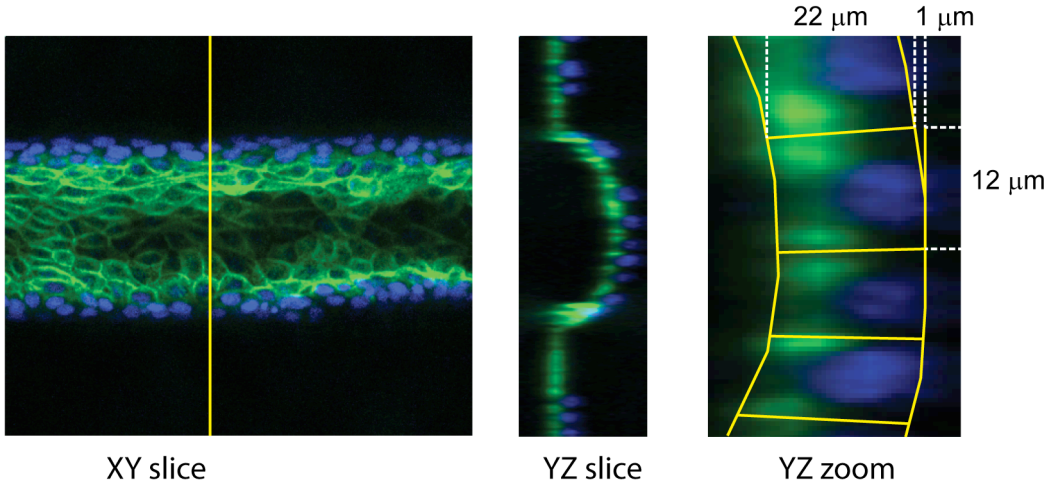


Figure S2: Height variation of a single cells at the nadir of curved substrate. Orthogonal views were manually measured to map out a cell-height profile. Height variation is 1 μm across a 12 μm wide cell.

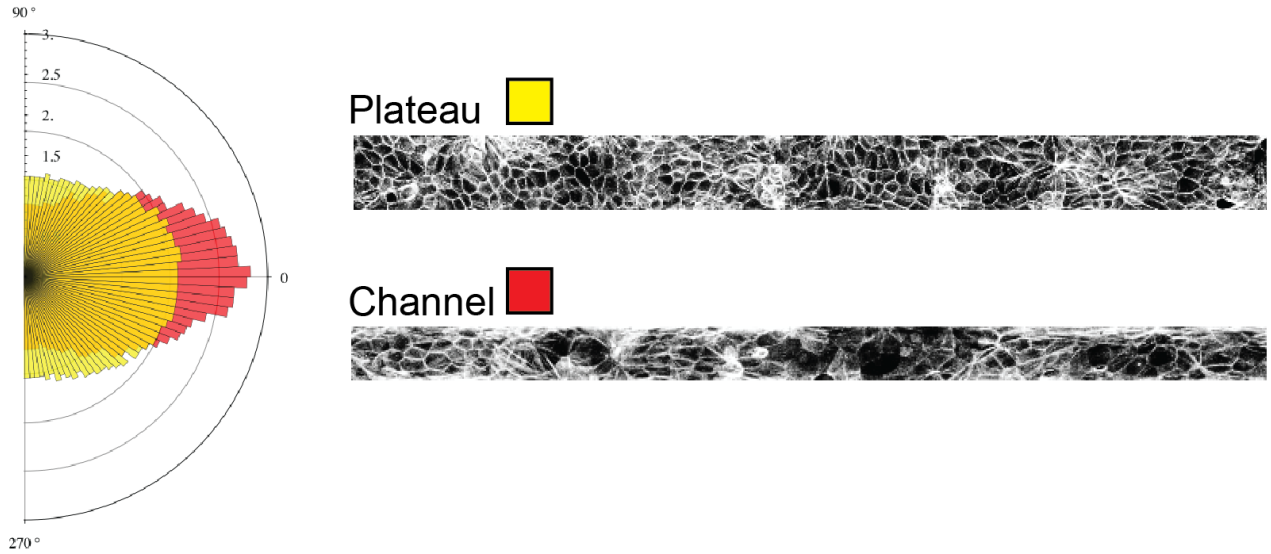


Figure S3: Alignment of actin on patterned substrates, especially in channels. Alignments calculated using Script S1. Representative sections of actin staining shown right.

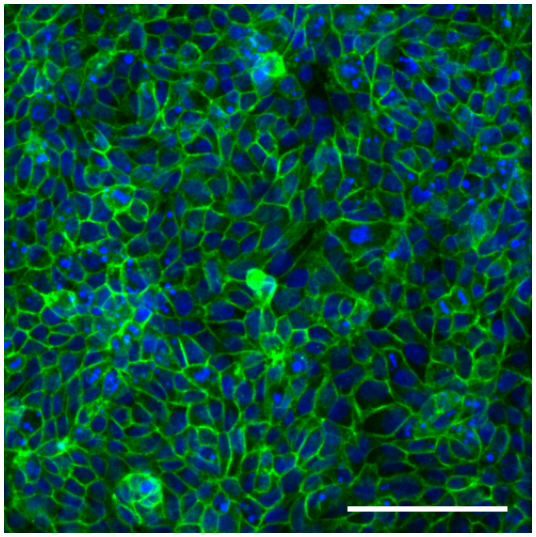


Figure S4: Cells are confluent and stable indefinitely on flat PDMS substrates prepared in the same way as curved substrates. Scale bar: 100 µm.

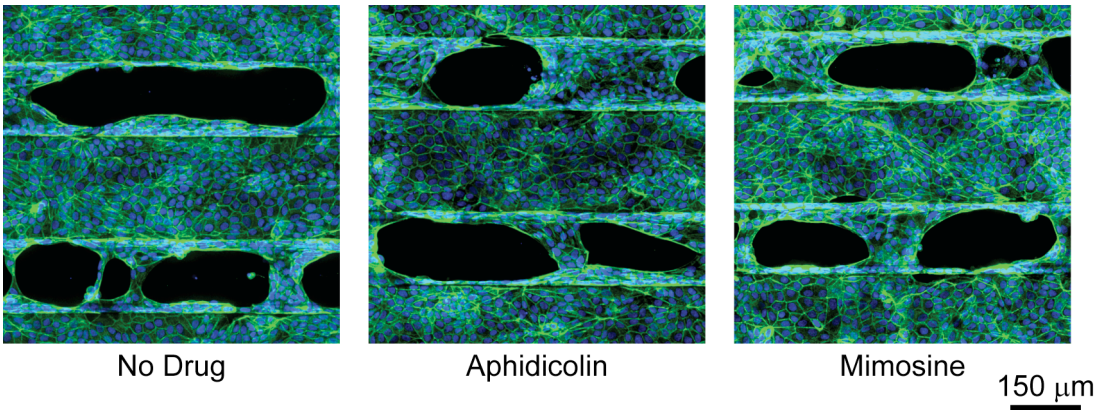


Figure S5: Inhibition of mitosis using aphidicolin (10 μm) or mimosine (1 mM) had no observable effect on lifting or clearing phenotypes.

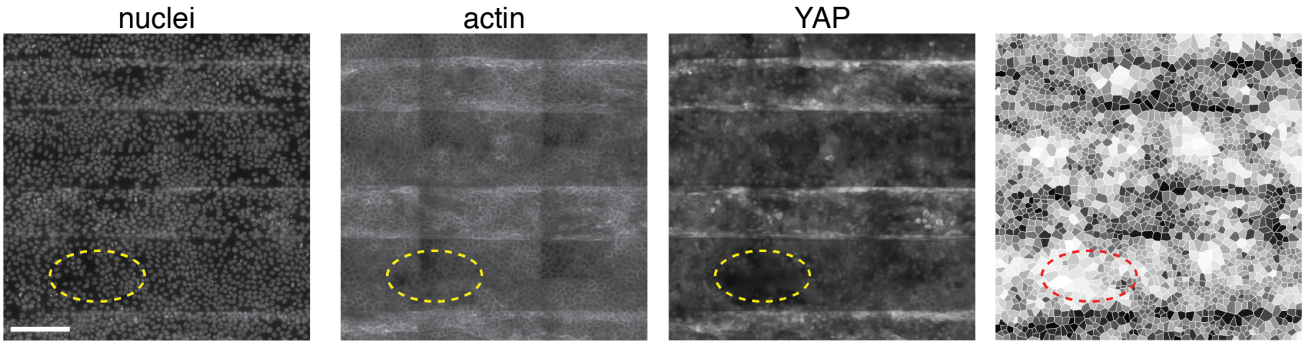


Figure S6: Method for quantifying nuclear yap in each cell. Cells were identified by finding nuclei and used to seed a Voronoi diagram. Within each Voronoi cell, YAP-associated fluorescence intensity was measured and compared to the proportion of that intensity that colocalized with nuclear staining. A heatmap was then generated where color correlated with % nuclear YAP staining within that cell (Script S3). Circled region shows one area within a plateau where nuclear YAP concentration is lower than average. Scale bar: 100 μm.

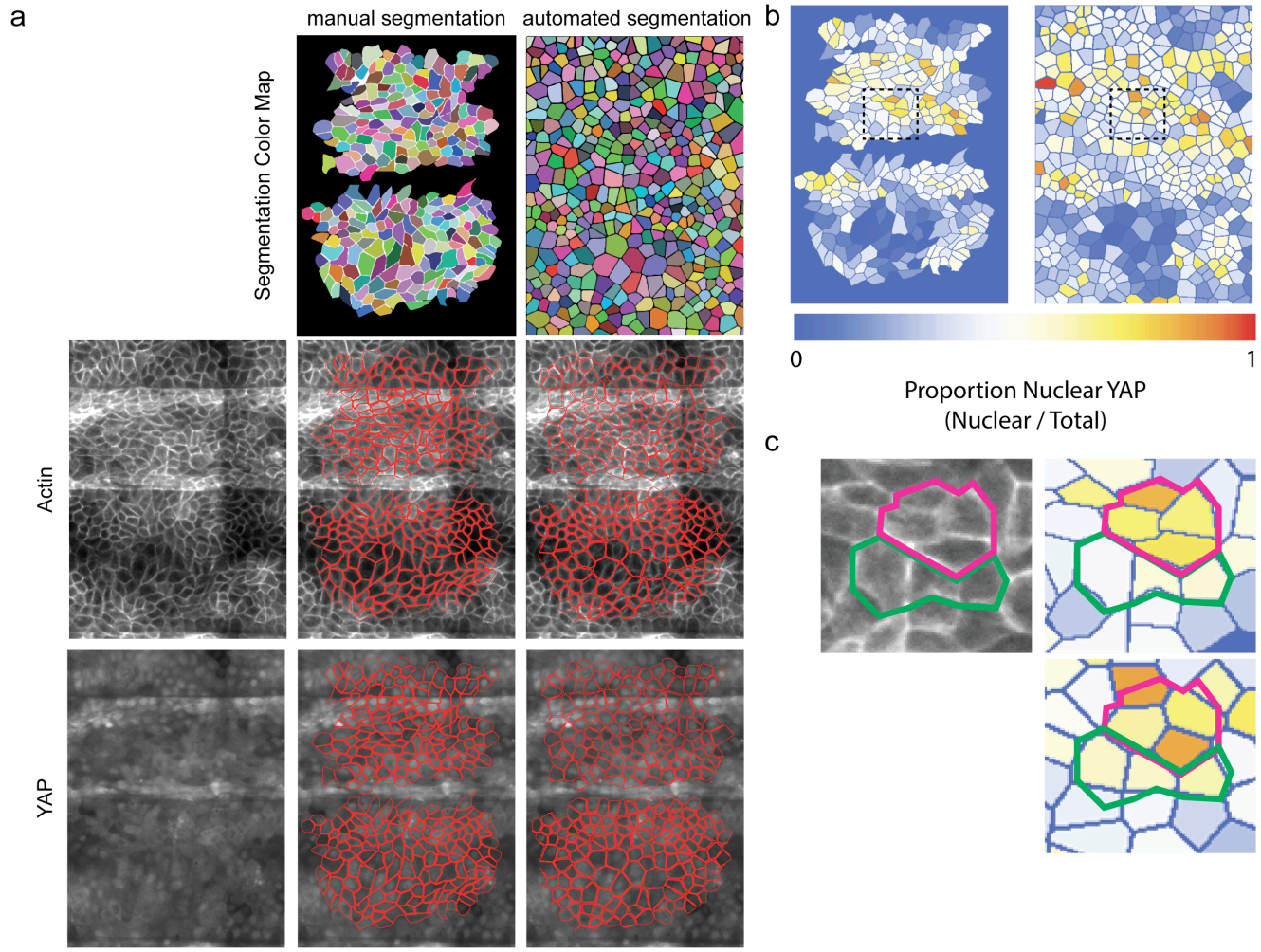


Figure S7: Comparison of segmentation methods for measuring nuclear YAP levels. (A) 300 cells were measured for YAP ratio after being hand-segmented (Script S4) and compared with the corresponding regions as segmented using the computational method. (B) Heat maps representing YAP translocation for hand (left) and computational (right) segmentation methods. (C) Close-up of heat map compared with actin staining showing two corresponding sets of cells (outlined).

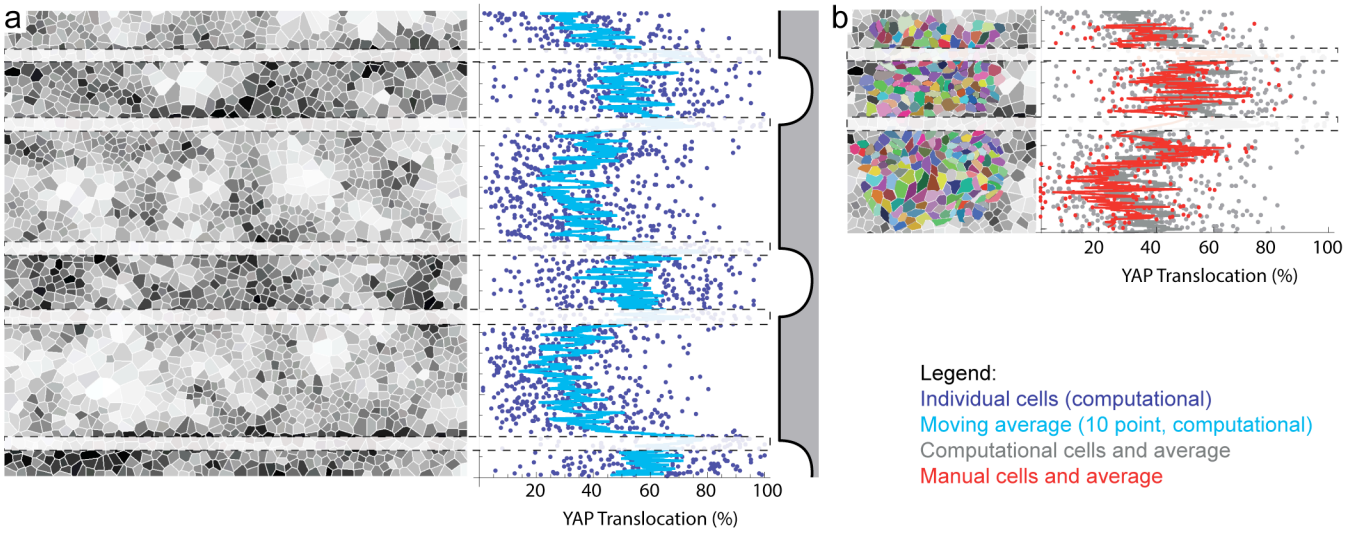


Figure S8: (A) Nuclear YAP levels for each cell plotted relative to position along the y-axis (blue). 10 point moving average (cyan) shows that channels possess elevated levels of nuclear YAP relative to plateaus. (B) Overlay of hand segmented nuclear YAP levels (red) compared with computationally measured levels (gray). Data from the two methods correspond on average.

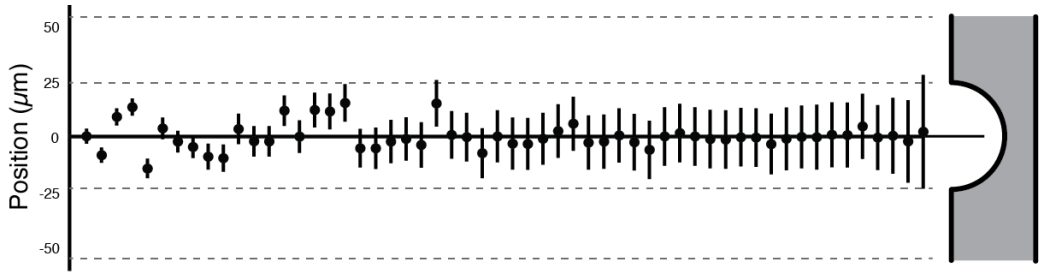


Figure S9: Tears are predominantly centered over the center of channels. 59 tears from 5 separate experiments were outlined by hand and their centroid positions were plotted relative to the local channel center in order of increasing tear width. Tear width was defined as the y dimension of a bounding box around the tear. The mean centroid position was 0 ± 5 μm.

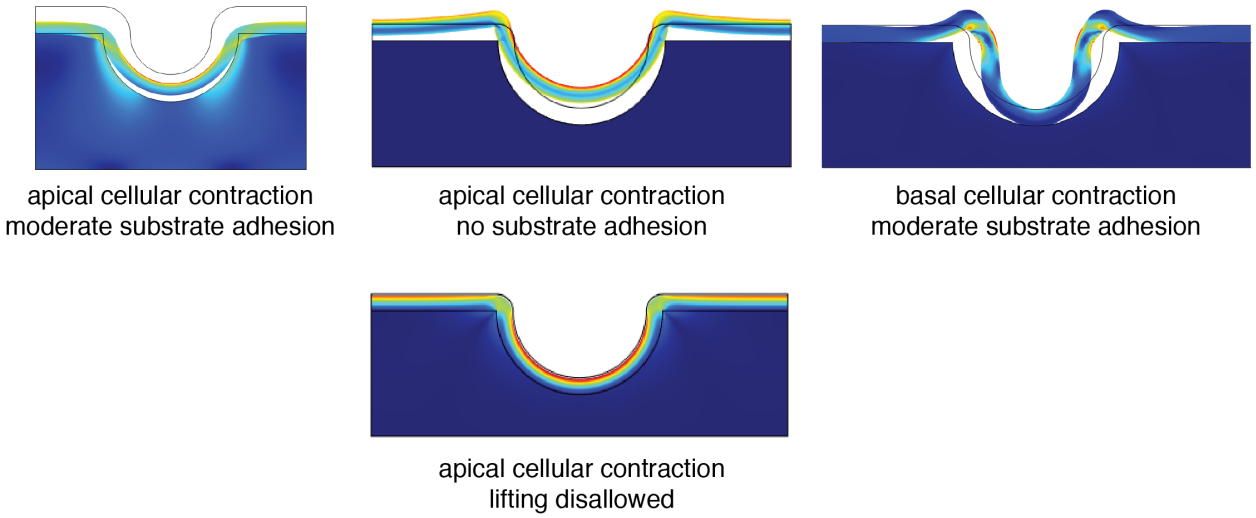


Figure S10: FEM model predictions showing that lifting only occurs when there is apical tissue contraction along with substrate adhesion. Modeling with lifting disallowed shows areas predicted to have high static stress in the tissue prior to lifting or clearing.

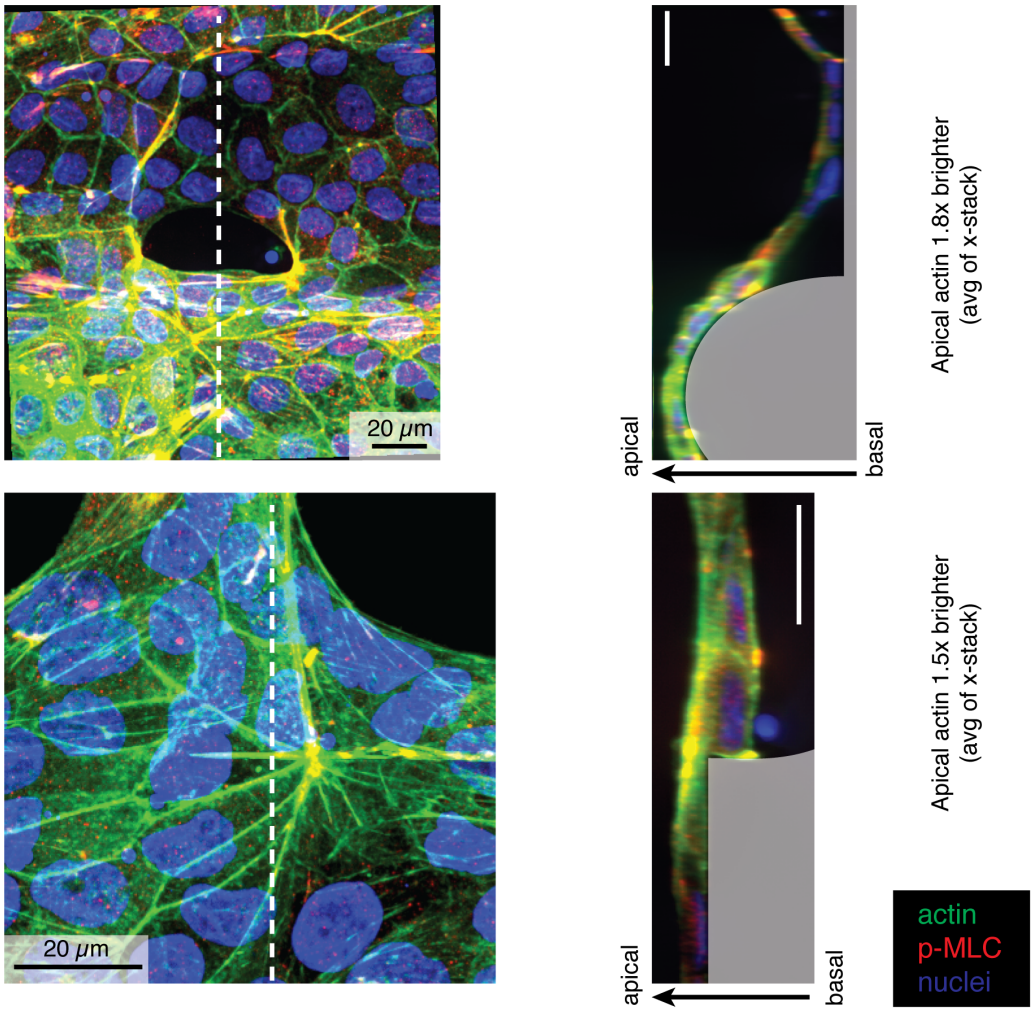


Figure S11: Actin and phosphorylated myosin light chain are enriched on the apical faces of lifting sheets in the context of ridges (top) and valleys (bottom).

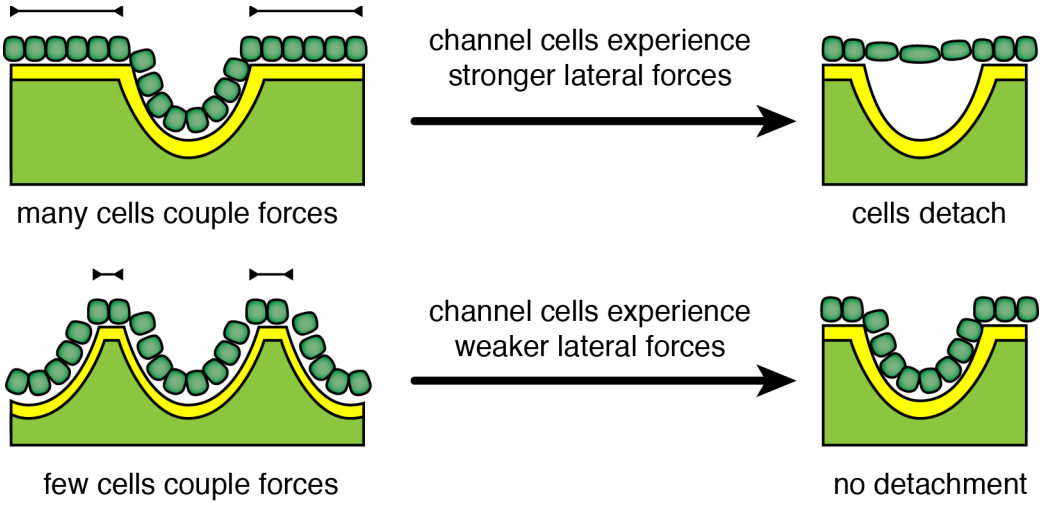


Figure S12: Schematic of proposed relationship between plateau width and tissue detachment. As more cells couple together on plateaus, more lateral force is generated. At some point, the apically directed component of the lateral forces in the channels becomes greater than the basally directed adhesive forces and the tissue detaches.

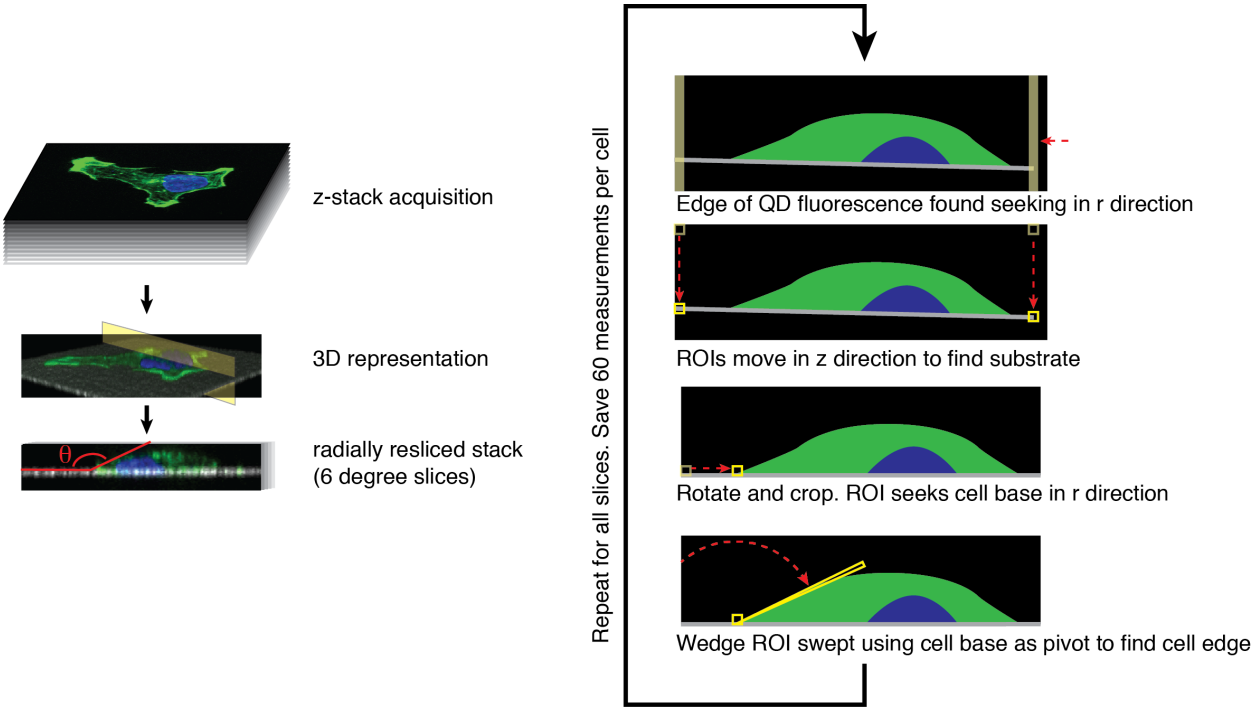
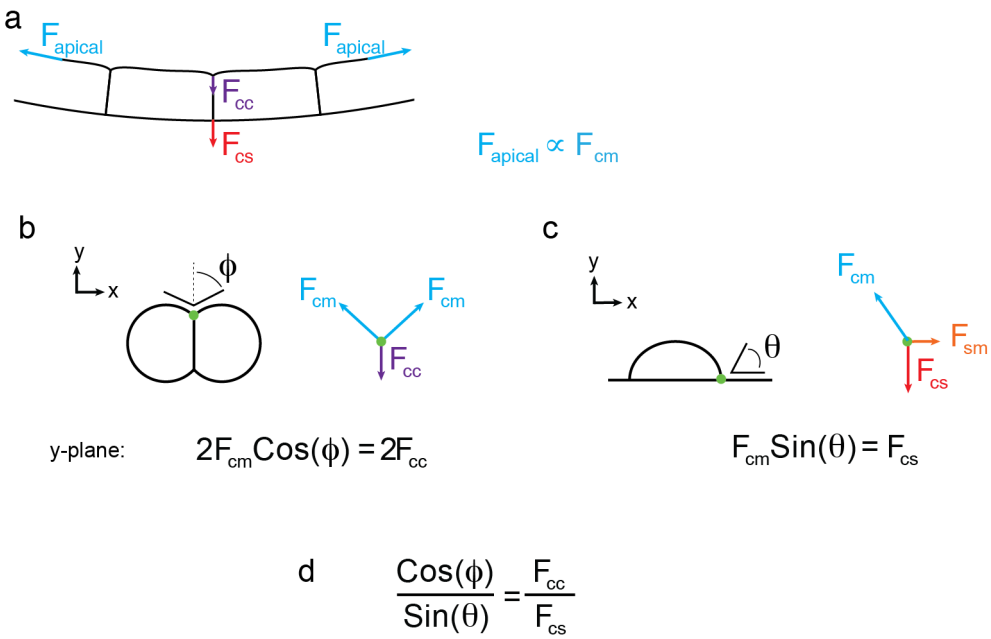


Figure S13: Algorithm for measuring cell-ECM contact angle. Z-stacks were resliced radially using 6 degree projections to improve signal to noise (Script S5). For each radial slice an ImageJ script rotated and cropped the image, then detected the angle between the advancing cell edge and the cell edge using threshold brightness values in regions of interest (ROIs). This was then run as a batch for each of >30 cells for each condition (Script S6).



d $$\frac{\mathrm{Cos}(\phi)}{\mathrm{Sin}(\theta)} = \frac{F_{cc}}{F_{cs}}$$

Figure S14: Illustration of cell-cell and cell-substrate forces estimated using contact angle measurement. (A) Forces experienced by cells in an intact epithelium. Apical epithelial forces are expected to be proportional to cell-medium forces in doublets based on topological similarity. (B-C) Projections of force along the y-plane, normal to the substrate, are the product of cell-medium force and either cos ϕ or sin θ. (D) The ratio of these measurements cancels the contribution from cell-medium interactions and provides a ratio of cell-cell and cell-substrate forces.

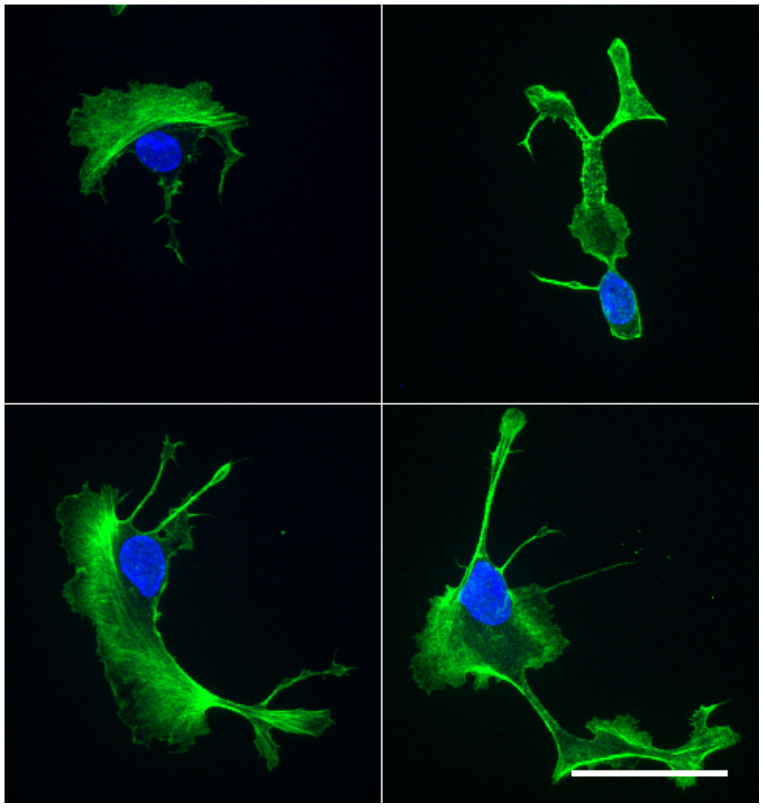
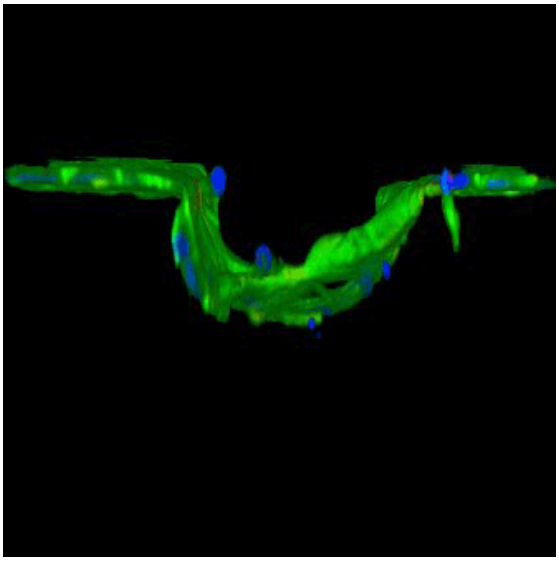


Figure S15. Blebbistatin-treated cells in isolation take on high surface-area morphologies. Scale bar: 40 µm.



Movie S1: Movie showing a growing discontinuity that has lifting along at its edge as it propagates.

Computational code

Script S1: Calculating average orientation of actin filaments (Mathematica)
It is recommended that code after each (*section break*) be placed in a new notebook section.

```
SetDirectory[
  SystemDialogInput["Directory",
   WindowTitle -> "Select directory with actin channel files"]];
files = FileNames["*.png"];

(* Import images, get local orientations and weight by local brightness \
gradient *)
dists = {};
imgs = {};
data = {};
Do[
  importedImage = Import[files[[index]]];
  grayImage = ColorConvert[importedImage, "Grayscale"];
  img = HistogramTransform[grayImage];
  AppendTo[imgs, img];
  o = Flatten@ImageData@GradientOrientationFilter[img, 1];
  w = Flatten@ImageData@Gradien1tFilter[img, 1];
  A = WeightedData[o, w];
  D = HistogramDistribution[A, {p/64}];
  AppendTo[dists, D];
  AppendTo[data, A];
  , {index, 1, Length[files], 1}];

(* Rescale weights manipulate arrays *)
weighted = {};
Do[
 weights = Floor[100*dists[[i, 2, 1]]];
 values = dists[[i, 2, 2]];
 tempweighted = {};
 Do[
  AppendTo[tempweighted, ConstantArray[values[[x]], weights[[x]]]],
  {x, 1, Length[weights], 1}];
 AppendTo[weighted, Flatten[tempweighted]];
 , {i, 1, Length[dists], 1}]

sowingBar[{{x0_, x1_}, {y0_, y1_}}, __] := (Sow[{x1 - x0, 100*(y1 - y0)}];
  Rectangle[{x0, y0}, {x1, y1}])

(* Generate histogram with ~3 degree bins and make mirrored data for 180∞ *)

histograms = {};
Do[
 {histogram, newdata} =
  Reap[Histogram[weighted[[x]], {p/64}, "Probability",
    ChartElementFunction -> sowingBar]];
 (*
 zeroed=newdata;
 zeroed[[1,All,2]]=0;
 AppendTo[histograms,Flatten[Join[newdata,zeroed],1]];
 *)
 AppendTo[histograms, Flatten[Join[newdata, Reverse[newdata]], 1]];
 , {x, 1, Length[weighted]}]

(* Section break *)
(* Generate radial plots *)
plots = {};
colors = {Red, Yellow};

Do[
  thisPlot = SectorChart[
    histograms[[i]],
    SectorOrigin -> -p/2,
    PlotRange -> Automatic,
    PolarAxes -> OddQ[i],
    PolarGridLines -> OddQ[i],
    PolarTicks -> {{{Pi/2, 0 ∞}, {2 Pi, 270 ∞}, {Pi, 90 ∞}, {3 Pi/2, 180 ∞}},
      Automatic},
```

```
    ChartBaseStyle -> EdgeForm[Thin],
    ChartStyle -> Directive[Opacity[0.75], colors[[Mod[i, 2, 1]]]],
    ImageSize -> Medium];
  AppendTo[plots, thisPlot];
  , {i, 1, Length[histograms], 1}];

(* Section break *)
(* Generate labeled plots and export *)
Do[
 label1 = StringDrop[files[[x]], Length[files[[x]]] - 4];
 label2 = StringDrop[files[[x + 1]], Length[files[[x + 1]]] - 4];
 output =
  Row[{Show[plots[[x]], plots[[x + 1]]],
    Column[{
      Show[imgs[[x      ]], ImageSize -> Medium,
       PlotLabel ->
        Style[Framed[label1], 16,
         Background -> Lighter[colors[[Mod[x, 2, 1]]]]]],
      Show[imgs[[x + 1]], ImageSize -> Medium,
       PlotLabel ->
        Style[Framed[label2], 16,
         Background -> Lighter[colors[[Mod[x + 1, 2, 1]]]]]]
      }, Center, Spacings -> 2]
    }, "  "];
 Print[output]
  (* Comment out this line if you don't want to export *)

  Export[files[[x]] <> ".pdf", output];
 , {x, 1, Length[plots], 2}
 ]
```

Script S2: Calculating portion of EdU and caspase-3 positive cells (Mathematica)
It is recommended that code after each (*section break*) be placed in a new notebook section.

```
(* Get a working directories for Nuclei and YAP *)
DAPIdirectory =
  SetDirectory[
   SystemDialogInput["Directory",
    WindowTitle -> "Select directory with DAPI channel"]];
DAPIfiles = FileNames["*.tif"];
EdUdirectory =
  SetDirectory[
   SystemDialogInput["Directory",
    WindowTitle -> "Select directory with EdU channel"]];
EdUfiles = FileNames["*.tif"];
Casp3directory =
  SetDirectory[
   SystemDialogInput["Directory",
    WindowTitle -> "Select directory with Casp3 channel"]];
Casp3files = FileNames["*.tif"];

(* Section break *)
(* Find nuclei and save their ceentroids *)
theDAPIs = {};
theNuclei = {};
DAPIcomps = {};
SetDirectory[DAPIdirectory];
Do[
  importedImage = Import[DAPIfiles[[i]]];
  grayImage = ColorConvert[importedImage, "Grayscale"];
  img = HistogramTransform[grayImage];
  binImage = Binarize[grayImage, .11];
  (* Find centers of regions of brightness *)
  distT = DistanceTransform[binImage, Padding -> 0];
  marker = MaxDetect[ImageAdjust[distT], 0.1];
  (* Use centers as seeds for component detection.
  Save centroids and radii. *)
  comp = WatershedComponents[GradientFilter[binImage, 3], marker,
    Method -> "Rainfall"];
  cells = SelectComponents[comp, "Count", 30 < # < 1000 &];
  measures =
   ComponentMeasurements[
    cells, {"Centroid", "EquivalentDiskRadius", "Label"}];
  AppendTo[theDAPIs, grayImage];
  AppendTo[theNuclei, measures];
  AppendTo[DAPIcomps, cells];
  , {i, 1, Length[DAPIfiles], 1}];
(* Uncomment to help choose binarization threshold
Row[{Show[ImageAdjust[grayImage]],Show[binImage]}]
*)

(* Section break *)
* This only needs to be run to quality check the nuclei analysis *)
Do[
 Print[
  Show[ImageAdjust[theDAPIs[[i]]],
   Graphics[
    {Red, Circle @@ # & /@ (theNuclei[[i]][[All, 2, 1 ;; 2]]),
     MapThread[
      Text, {theNuclei[[i]][[All, 2, 3]],
       theNuclei[[i]][[All, 2, 1]]}]}]
   ]
  ]
 , {i, 1, Length[DAPIfiles], 1}]

(* Section break *)
(* Import all EdU and Casp files and measure mean brightness in each \
of the nuclei above*)
SetDirectory[EdUdirectory];
theEdUs = Map[Import[#] &, EdUfiles];
EdUlvls =
  Table[ComponentMeasurements[{DAPIcomps[[i]], theEdUs[[i]]},
    "Mean"], {i, 1, Length[theNuclei]}];
SetDirectory[Casp3directory];
```

```
theCasp3 = Map[Import[#] &, Casp3files];
Casplvls =
  Table[ComponentMeasurements[{DAPIcomps[[i]], theCasp3[[i]]},
    "Mean"], {i, 1, Length[theNuclei]}];

(* Section break *)
(* Build a list of statistics about each file trio and present it in \
a table *)
stats = Table[{
    EdUfiles[[i]],
    eduPos = Count[EdUlvls[[i]][[All, 2]], u_ /; u > 0.025],
    casPos = Count[Casplvls[[i]][[All, 2]], u_ /; u > 0.17],
    numNuc = EdUlvls[[i]] // Length,
    100*N[eduPos/numNuc, 2],
    100*N[casPos/numNuc, 2]
    }
   , {i, 1, Length[EdUfiles]}];

Grid[PrependTo[
  stats, {"filename", "EdU positive", "Casp positive",
   "total Nuclei" , "% EdU positive", "% Casp positive"}],
 Frame -> All]
```

Script S3: Calculating per-cell YAP levels with computational segmentation (Mathematica)
It is recommended that code after each (*section break*) be placed in a new notebook section.

```
(* Get a working directories for Nuclei and YAP *)

DAPIdirectory = SetDirectory[SystemDialogInput["Directory", WindowTitle -> "Select directory with DAPI channel"]];
DAPIfiles = FileNames["*.png"];
YAPdirectory = SetDirectory[SystemDialogInput["Directory", WindowTitle -> "Select directory with YAP channel"]];
YAPfiles = FileNames["*.png"];
If[Length[DAPIfiles] != Length[YAPfiles], MessageDialog[
  "These directories don't match. Fix before continuting!"]
]

(* Section break *)
(* Find nuclei and save their centroids *)
theDAPIs = {};
theNuclei = {};
SetDirectory[DAPIdirectory];
Do[
  importedImage = Import[DAPIfiles[[i]]];
  grayImage = ColorConvert[importedImage, "Grayscale"];
  img = HistogramTransform[grayImage];
  binImage = Binarize[grayImage, Method -> "Cluster"];
  (* Find centers of regions of brightness *)

  distT = DistanceTransform[binImage, Padding -> 0];
  marker = MaxDetect[ImageAdjust[distT], 0.1];
  (* Use centers as seeds for component detection.
  Save centroids and radii. *)

  comp = WatershedComponents[GradientFilter[binImage, 3], marker, Method -> "Rainfall"];
  cells = SelectComponents[comp, "Count", 30 < # < 1000 &];
  measures = ComponentMeasurements[cells, {"Centroid", "EquivalentDiskRadius", "Label"}];
  AppendTo[theDAPIs, grayImage];
  AppendTo[theNuclei, measures];
  , {i, 1, Length[DAPIfiles], 1}];


(* Section break *)
(* This only needs to be run to quality check the nuclei analysis*)
Do[
 Print[
  Show[ImageAdjust[theDAPIs[[i]]],
   Graphics[
    {Red, Circle @@ # & /@ (theNuclei[[i]][[All, 2, 1 ;; 2]]),
     MapThread[Text, {theNuclei[[i]][[All, 2, 3]], theNuclei[[i]][[All, 2, 1]]}]}]
   ]
  ]
 , {i, 1, Length[DAPIfiles], 1}]


(* Section break *)
(* Generate a component analysis field corresponding to a Voronoi graph using nuclei as seeds. *)
theVoronois = {};
theOverlays = {};
Do[
 points = theNuclei[[i, All, 2, 1]];
 {xmax, ymax} = ImageDimensions[theDAPIs[[i]]];
 data2 = MapIndexed[Flatten[{##}] &, points];
 cellZones =
  Rasterize[
   ListDensityPlot[data2,
    MeshStyle ->
     Directive[GrayLevel[0], Opacity[0.5`], AbsoluteThickness[2]],
    InterpolationOrder -> 0, ColorFunction -> "Pastel", Mesh -> All,
    AspectRatio -> "Full", Frame -> False,
    ImageSize -> ImageDimensions[theDAPIs[[i]]],
    PlotRangePadding -> None, PlotRange -> {{0, xmax}, {0, ymax}}]];
 components = MorphologicalComponents[Binarize[cellZones]];
 AppendTo[theOverlays, cellZones];
 AppendTo[theVoronois, components];
 , {i, 1, Length[DAPIfiles], 1}]
```

```
(* Section break *)
(* This only needs to be run to quality check the correspondence of the Voronoi with the input data*)
testOverlays = {};
Do[
 AppendTo[testOverlays,
  ImageCompose[ImageAdjust[theDAPIs[[i]]], {theOverlays[[i]], 0.4}]]
 , {i, 1, Length[DAPIfiles], 1}]
Manipulate[testOverlays[[a]], {a, 1, Length[theDAPIs], 1}]


(* Section break *)
(* Import YAP data and separate into nuclear and non-nuclear channels. *)
totalYAPs = {};
cytoYAPs = {};
nuclearYAPs = {};

SetDirectory[YAPdirectory];
Do[
 (* Make masks corresponding to nuclear and non-nuclear regions (1 for ROI, else 0). *)
 threshold = FindThreshold[theDAPIs[[i]], Method -> "Cluster"];
 DAPIbin = Binarize[GaussianFilter[theDAPIs[[i]], 3, Padding -> 0], threshold];
 DAPImask = ImageData[ColorNegate[DAPIbin]];
 DAPIunmask = ImageData[DAPIbin];
 (* Import YAP image and convert to an array to multiply with the masks. Save out separated images.*)
 importedImage = Import[YAPfiles[[i]]];
 grayImage = ColorConvert[importedImage, "Grayscale"];
 YAParray = ImageData[grayImage];
 cytosolic = Image[MapThread[#1*#2 &, {DAPImask, YAParray}]];
 nuclear = Image[MapThread[#1*#2 &, {DAPIunmask, YAParray}]];
 AppendTo[totalYAPs, grayImage];
 AppendTo[cytoYAPs, cytosolic];
 AppendTo[nuclearYAPs, nuclear];
 , {i, 1, Length[YAPfiles], 1}]

(* Measure YAP intensity in nuclei and cytosol per Voronoi cell *)

theHeatMaps = {};
Do[
 Print["Building voronoi for image ", i];
 cytoYAPintensity = ComponentMeasurements[{theVoronois[[i]], cytoYAPs[[i]]}, "Total"];
 nuclearYAPintensity = ComponentMeasurements[{theVoronois[[i]], nuclearYAPs[[i]]}, "Total"];
 totalYAPintensity = ComponentMeasurements[{theVoronois[[i]], totalYAPs[[i]]}, "Total"];
 portionNuclear =
  Partition[Riffle[
    totalYAPintensity[[All, 1]],
    MapThread[Divide, {nuclearYAPintensity[[All, 2]],totalYAPintensity[[All, 2]]}]], 2];

 tmp = theVoronois[[i]];
 Do[
  tmp = Replace[tmp, i -> portionNuclear[[i, 2]], 2];
  , {i, 1, Length[portionNuclear], 1}];

 AppendTo[theHeatMaps, tmp];
 , {i, 1, Length[theVoronois], 1}]


(* Section break *)
(* This section makes the images for export from the data above.*)
heatOverlays = {};
ratioPlots = {};
Do[
 heatmap =
  ArrayPlot[theHeatMaps[[i]], ColorFunction -> "TemperatureMap",
   ImageSize -> ImageDimensions[theDAPIs[[i]]],
   PlotRangePadding -> None, Frame -> False, AspectRatio -> "Full"];
 AppendTo[ratioPlots, heatmap];
 AppendTo[heatOverlays,
  ImageCompose[ImageAdjust[totalYAPs[[i]]], {heatmap, 0.4}]]
 , {i, 1, Length[DAPIfiles], 1}]


(* Section break *)
(* This section exports the files from above from a user-selected directory *)
```

```
exportDir =
  SetDirectory[
   SystemDialogInput["Directory",
    WindowTitle -> "Select directory for export."]];
Do[
 ratioTitle =
  StringJoin[Characters[DAPIfiles[[i]]][[1 ;; 4]]] <>
   "YAPratios.png";
 Export[ratioTitle, ratioPlots[[i]]];
 overlayTitle =
  StringJoin[Characters[DAPIfiles[[i]]][[1 ;; 4]]] <> "YAPoverlay.png";
 Export[overlayTitle, heatOverlays[[i]]];
 , {i, 1, Length[ratioPlots], 1}]

(* This section generates a plot of YAP nuclear levels versus vertical position. *)
(* NOTE: THIS OVERWRITES VALUES FROM ABOVE SECTIONS. RUN LAST OR CHANGE VARIABLE NAMES APPROPRIATELY *)
Do[
 nuclearYAPintensity = ComponentMeasurements[{theVoronois[[i]], nuclearYAPs[[i]]}, {"Total", "Centroid"}];
 totalYAPintensity = ComponentMeasurements[{theVoronois[[i]], totalYAPs[[i]]}, {"Total", "Centroid"}];
 portionNuclear =
  Partition[
   Riffle[
    totalYAPintensity[[All, 2, 2, 2]],
    MapThread[Divide, {nuclearYAPintensity[[All, 2, 1]],totalYAPintensity[[All, 2, 1]]}]
    ]
   , 2];
, {i, 1, Length[theVoronois], 1}]

a = ListPlot[portionNuclear];
b = ListLinePlot[MovingAverage[Sort[portionNuclear], 10], PlotStyle -> Red];
Show[a, b]
```

Script S4: Calculating per-cell YAP levels using hand segmentation (ImageJ/Fiji then Mathematica)
Segmentation preparation:
Outline a cell using the lasso tool then run the following macro. Outline a new cell before hitting “next” each time.

```
for(var i=0;i<300;i++){
        waitForUser("Next");
        r=g=b=0;
        run("Enlarge...","enlarge=1");
        setForegroundColor(r, g, b);
        fill();
        r=g=b=255;
        run("Enlarge...","enlarge=-1");
        setForegroundColor(r, g, b);
        fill();
}
```

Mathematica:
It is recommended that code after each (*section break*) be placed in a new notebook section.

```
(* Get a working directories for segmentation, nuclei and YAP *)
segDir = SetDirectory[SystemDialogInput["Directory", WindowTitle -> "Select directory with segmentation"]];
theSegs = FileNames["*.tif"];
DAPIdirectory = SetDirectory[WindowTitle -> "Select directory with DAPI channel"]];
DAPIfiles = FileNames["*.tif"];
YAPdirectory = SetDirectory[SystemDialogInput["Directory", WindowTitle -> "Select directory with YAP channel"]];
YAPfiles = FileNames["*.tif"];

(* Section break  *)
(* Import segmentations and binarize *)
theVoronois = {};
SetDirectory[segDir];
thisSeg = Binarize[ Import[ theSegs[[1]] ], .99];

comps = MorphologicalComponents[thisSeg];

AppendTo[theVoronois, comps];

(* Section break  *)
(* Import Nuclei *)
theDAPIs = {};
SetDirectory[DAPIdirectory];
Do[
  importedImage = Import[DAPIfiles[[i]]];
  grayImage = ColorConvert[importedImage, "Grayscale"];
  AppendTo[theDAPIs, grayImage];
  , {i, 1, Length[DAPIfiles], 1}];

(* Section break  *)
SetDirectory[YAPdirectory];
Do[
 (* Make masks corresponding to nuclear and non-nuclear regions (1 for ROI, \
else 0). *)

 threshold = FindThreshold[theDAPIs[[i]], Method -> "Cluster"];
 DAPIbin = Binarize[GaussianFilter[theDAPIs[[i]], 3, Padding -> 0], threshold];
 DAPImask = ImageData[ColorNegate[DAPIbin]];
 DAPIunmask = ImageData[DAPIbin];
 (* Import YAP image and convert to an array to multiply with the masks. Save \
out separated images.*)
 importedImage = Import[YAPfiles[[i]]];
 grayImage = ColorConvert[importedImage, "Grayscale"];
 YAParray = ImageData[grayImage];
 cytosolic = Image[MapThread[#1*#2 &, {DAPImask, YAParray}]];
 nuclear = Image[MapThread[#1*#2 &, {DAPIunmask, YAParray}]];
 AppendTo[totalYAPs, grayImage];
 AppendTo[cytoYAPs, cytosolic];
 AppendTo[nuclearYAPs, nuclear];
 , {i, 1, Length[YAPfiles], 1}]
```

```
(* Section Break *)
(* Measure YAP intensity in nuclei and cytosol per Voronoi cell *)

theHeatMaps = {};
Do[
 Print["Building voronoi for image ", i];
 cytoYAPintensity =ComponentMeasurements[{theVoronois[[i]], cytoYAPs[[i]]}, "Total"];
 nuclearYAPintensity = ComponentMeasurements[{theVoronois[[i]], nuclearYAPs[[i]]}, "Total"];
 totalYAPintensity = ComponentMeasurements[{theVoronois[[i]], totalYAPs[[i]]}, "Total"];
 portionNuclear =
  Partition[Riffle[
    totalYAPintensity[[All, 1]],
    MapThread[Divide, {nuclearYAPintensity[[All, 2]], totalYAPintensity[[All, 2]]}]], 2];

   tmp = theVoronois[[i]];
   Do[
    tmp = Replace[tmp, i -> portionNuclear[[i, 2]], 2];
   , {i, 1, Length[portionNuclear], 1}];

 AppendTo[theHeatMaps, tmp];
 , {i, 1, Length[theVoronois], 1}]

(* Section Break *)
heatOverlays = {};
ratioPlots = {};
Do[
 heatmap =
  ArrayPlot[theHeatMaps[[i]], ColorFunction -> "TemperatureMap",
   ImageSize -> ImageDimensions[theDAPIs[[i]]], PlotRangePadding -> None,
   Frame -> False, AspectRatio -> "Full"];
 AppendTo[ratioPlots, heatmap];
 AppendTo[heatOverlays,
  ImageCompose[ImageAdjust[totalYAPs[[i]]], {heatmap, 0.4}]]
 , {i, 1, Length[DAPIfiles], 1}]

Manipulate[heatOverlays[[a]], {a, 1, Length[theDAPIs], 1}]

(* Section Break *)

(* Measure YAP intensity in nuclei and cytosol per Voronoi cell *)
(* NOTE: THIS OVERWRITES VALUES FROM ABOVE SECTIONS. RUN LAST OR CHANGE VARIABLE NAMES APPROPRIATELY *)

theHeatMaps = {};
Do[
 nuclearYAPintensity =
  ComponentMeasurements[{theVoronois[[i]], nuclearYAPs[[i]]}, {"Total",
    "Centroid"}];
 totalYAPintensity =
  ComponentMeasurements[{theVoronois[[i]], totalYAPs[[i]]}, {"Total",
    "Centroid"}];
 portionNuclear =
  Partition[
   Riffle[
    totalYAPintensity[[All, 2, 2, 2]],
    MapThread[
     Divide, {nuclearYAPintensity[[All, 2, 1]],
      totalYAPintensity[[All, 2, 1]]}]
    ]
   , 2];
 , {i, 1, Length[theVoronois], 1}]
a = ListPlot[portionNuclear, PlotRange -> {{0, 1100}, Automatic}];
b = ListLinePlot[MovingAverage[Sort[portionNuclear], 3], PlotStyle -> Red];
Show[a, b]
```

Script S5: BatchRadialReslice.ijm (FIJI or ImageJ)

```
/*
 * Kyle Broaders
 * Gartner Lab
 * This script expects a directory of 3-channel z-stack images. It has only been tested for .czi images.
 * The output of the script is each input cell resliced radially around a user-drawn line.
 *
 * The title parsing only looks at the string before the first "."
 * Make sure your file names only contain a "." for the extension.
 *
 */

//get input images
dir1 = getDirectory("Choose Source Directory ");
list = getFileList(dir1);
//get output folder
dir2 = getDirectory("Choose Destination Directory ");

//loop through images in source directory
setBatchMode(true);
for (x=0; x<list.length; x++) {
  showProgress(x+1, list.length);
  //only process images, not directories
  if(!endsWith(list[x],"/")){
  open(dir1+list[x]);

  // Get user input to draw a line through the cell.
  // The center of the line should be roughly the center of the cell
  setBatchMode("show");
  waitForUser("Draw the line to reslice");
  setBatchMode("hide");

  //split the image into composite channels and process them. This script assumes 3 channels.
  title=getTitle();
  //parse the title before the first "."
  titleNoExt=substring(title,0,indexOf(title,"."));
  getLine(x1, y1, x2, y2, lineWidth);
  run("Split Channels");

  for(i=1;i<=3;i++){
    selectWindow("C"+i+"-"+title);
    makeLine(x1, y1, x2, y2);
    run("Radial Reslice", "angle=360 degrees_per_slice=1 direction=Clockwise rotate_about_centre");
    run("Flip Vertically", "stack");
    selectWindow("C"+i+"-"+title);
    close();
  }
  //remerge images as a 3-channel r-stack and save to new directory.
  run("Merge Channels...", "c1=[Reslice of C1-"+titleNoExt+"] c2=[Reslice of C2-"+titleNoExt+"] c3=[Reslice of C3-
"+titleNoExt+"]");
  saveAs("tiff", dir2+list[x]);
  close();
  }
}
```

Script S6: BatchCAFinder (FIJI or ImageJ)

```
/*
 * Kyle Broaders
 * Gartner Lab
 * This script expects a directory of 3-channel r-stack images.
 * It has only been tested for the output of a batch reslice script that outputs TIFF files.
 *
 * The output of the script is a txt file containing only valid measurements of cell-ECM angle. There are at most
360/radstep measurements.
 *
 * The script expects a mac or unix style directory notation. Changes would need to be made to run on PC.
 * The title parsing only looks at the string before the first "." Make sure your file names only contain a "."
for the extension.
 */

//User input for input and output directories.
dir1 = getDirectory("Choose Source Directory ");
dir2 = getDirectory("Choose Destination Directory ");

//User input for surface channel and cell edge channel
QDchannel = getNumber("Select QD channel",1);
cellChannel = getNumber("Select cell channel",1);

list = getFileList(dir1);
setBatchMode(true);

//Loop through every file in the directory
for (filenum=0; filenum<list.length; filenum++) {
  showProgress(filenum+1, list.length);
  //Only process images that aren't directories
  if(!endsWith(list[filenum],"/")){
    open(dir1+list[filenum]);

    stacktitle = getTitle();
    condition = substring(stacktitle,0,indexOf(stacktitle,"."));

    //This is a hardcoded number of slices to project together for measurement.
    //It should divide evenly into 360 for best results.
    //Increasing the number improves SNR at the cost of cell edge.
    radstep=6;

    //Loop through each set of radslice
    for(radslice=1;radslice<=360;radslice+=radstep){
      selectWindow(stacktitle);
      run("Z Project...","start="+radslice+" stop="+radslice+radstep-1+" projection=[Max Intensity]");
      selectWindow("MAX_"+stacktitle);
      rename("WorkingSlice "+radslice);

      getDimensions(w,h,nChannels,slices,frames);
      Stack.setChannel(QDchannel);
      run("Set Measurements...", "  mean redirect=None decimal=3");

      //Hardcoded search window size in pixels
      QDsearchWidth=2;
      QDsearchHeight=7;

      //initialize values
      QDbrightestL=0;
      QDbrightestYL=0;
      //Find left X edge in case reslicing introduced empty space. Any fluorescence will exit loop.
      xEdgeL=-1;
      do{
        xEdgeL = xEdgeL+ 1;
        makeRectangle(xEdgeL,0,QDsearchWidth,QDsearchHeight);
        List.setMeasurements;
        brightness=List.get("Mean");
      } while(brightness<1);

      if(xEdgeL>0){
        xEdgeL = xEdgeL+QDsearchWidth+1;
      }

      //find brightest QDs in the Y direction on left side
```

```
for(y=0; y<h-QDsearchHeight; y++){
  makeRectangle(xEdgeL,y,QDsearchWidth,QDsearchHeight);
  List.setMeasurements;
  brightness=List.get("Mean");
  if(brightness>QDbrightestL){
    QDbrightestL=brightness;
    QDbrightestYL=y;
  }
  run("Select None");
}

QDbrightestR=0;
QDbrightestYR=0;
//Find right X edge in case reslicing introduced empty space. Any fluorescence will exit loop.
xEdgeR = w - QDsearchWidth + 1;
xEdgeR0 = xEdgeR-1;
do{
  xEdgeR = xEdgeR - 1;
  makeRectangle(xEdgeR,0,QDsearchWidth,QDsearchHeight);
  List.setMeasurements;
  brightness=List.get("Mean");
} while(brightness<1);

if(xEdgeR != xEdgeR0){
  xEdgeR = xEdgeR-QDsearchWidth-1;
}

//find brightest QDs in the Y direction on right side
for(y=0; y<h-QDsearchHeight; y++){
  makeRectangle(xEdgeR,y,QDsearchWidth,QDsearchHeight);
  List.setMeasurements;
  brightness=List.get("Mean");
  if(brightness>QDbrightestR){
    QDbrightestR=brightness;
    QDbrightestYR=y;
  }
  run("Select None");
}


//Rotate image based on line between brightest QD point on right and left side
makeLine(xEdgeL+QDsearchWidth/2,
  QDbrightestYL+QDsearchHeight/2,
  xEdgeR+QDsearchWidth/2,
  QDbrightestYR+QDsearchHeight/2
);
getSelectionCoordinates(X,Y);
run("Select None");
angle=-(180/3.14158)*atan((Y[1]-Y[0])/(X[1]-X[0]));
run("Rotate... ", "angle="+angle+" grid=1 interpolation=Bilinear enlarge stack");

//refind new bottom for crop to surface
QDbrightestM=0;
QDbrightestYM=0;
getDimensions(w,h,nChannels,slices,frames);
//find brightest QDs in the Y direction on left side
for(y=0; y<h-QDsearchHeight; y++){
  makeRectangle(w/2,y,QDsearchWidth,QDsearchHeight);
  List.setMeasurements;
  brightness=List.get("Mean");
  if(brightness>QDbrightestM){
    QDbrightestM=brightness;
    QDbrightestYM=y;
  }
  run("Select None");
}
makeRectangle(0,0,w,QDbrightestYM+QDsearchHeight/2);
run("Crop");

getDimensions(w,h,nChannels,slices,frames);
Stack.setChannel(cellChannel);

//Binarize to simplify problem
setAutoThreshold("Mean dark");
```

```
      run("Convert to Mask", "method=Default background=Dark black");

      //Find base of cell
      //Hardcoded search window for pivot
      ActinSearchWidth=3;
      ActinSearchHeight=3;
      //Initialize values
      pivotX=0;
      foundPivot=false;

      for(x=0;x<w/2;x++){
        makeRectangle(x,h-ActinSearchHeight,ActinSearchWidth,ActinSearchHeight);
        List.setMeasurements;
        brightness=List.get("Mean");
        if(brightness>128 && foundPivot==false){
          pivotX=x;
          foundPivot=true;
          x = w/2 + 1;
        }
        run("Select None");
      }

      //Hardcoded search window for cell edge
      ActinSearchRadius=25;
      ActinSearchAngle=PI/15;
      foundAngle=false;
      X0=pivotX;
      Y0=h-ActinSearchHeight/2;
      finX=-1;
      finY=-1;
      CA=-1;
      //Swing search window until threshold brightness reached
      for(a=PI;a>PI/20;a = a - PI/180){
        if(foundAngle==false){
          X1=X0+ActinSearchRadius*cos(a);
          Y1=Y0-ActinSearchRadius*sin(a);
          X2=X0+ActinSearchRadius*cos(a-ActinSearchAngle);
          Y2=Y0-ActinSearchRadius*sin(a-ActinSearchAngle);
          makePolygon(X0,Y0,X1,Y1,X2,Y2);
          List.setMeasurements;
          brightness=List.get("Median");
          //128 is a hardcoded threshold emperically found to work.
          if(brightness>128 && foundAngle==false){
            finX=X0+ActinSearchRadius*cos(a-ActinSearchAngle/2);
            finY=Y0-ActinSearchRadius*sin(a-ActinSearchAngle/2);
            CA=(180/PI)*(PI-a);
            foundAngle=true;
            a1=a;
          }
        run("Select None");
        }
      }

      //Save angle to results
      setResult(condition,nResults,CA);

      selectWindow("WorkingSlice "+radslice);
      close();
    }

    //Save out results per condition
    saveAs("Results", dir2+condition+".txt");
    //Clean up in preparation for next iteration
    run("Clear Results");
    selectWindow(stacktitle);
    close();
  }
}
```