Supplementary Material

Includes Supplementary Figures, Supplementary Movie Legends, Supplementary Text, Strain and plasmid table, and Strain construction

Supplementary Figures

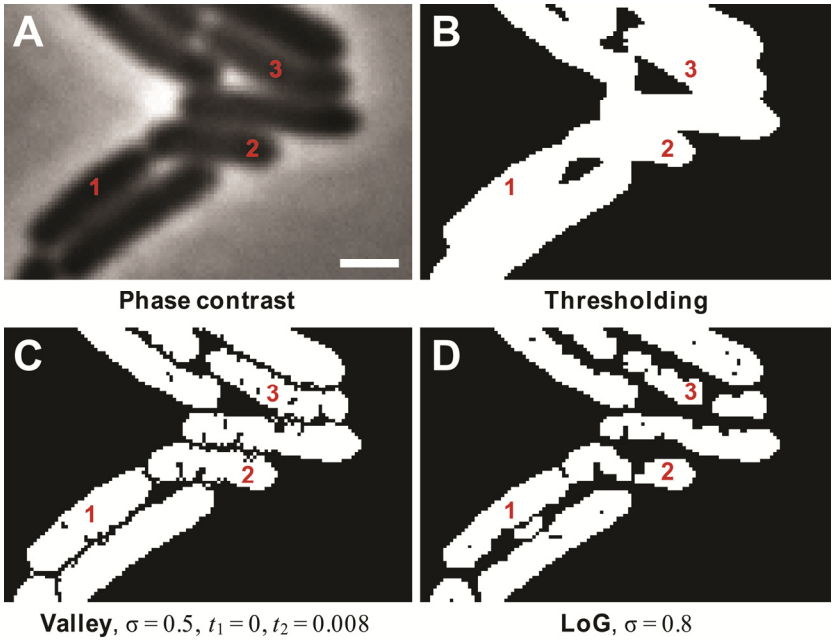

Fig. S1. Performance of thresholding and edge detection algorithms on a cell cluster. A. Phase contrast image of *E. coli* cells touching each other (strain MC1000/pWM1409/pFx40). **B.** The same image after thresholding with an automatically determined threshold. **C.** The same image after performing thresholding and valley detection with optimal parameters. **D.** The same image after performing thresholding and edge detection using Laplacian of Gaussian algorithm with optimal parameters. A typical problem shown, when cells 1 and 2 are not separated, and the cells 2 and 3 are fragmented at the same time. Here, σ – width of the Gaussian filter, t_1 and t_2 – weak and strong thresholds of the valley detection algorithm, respectively. Bar: 1 μm.

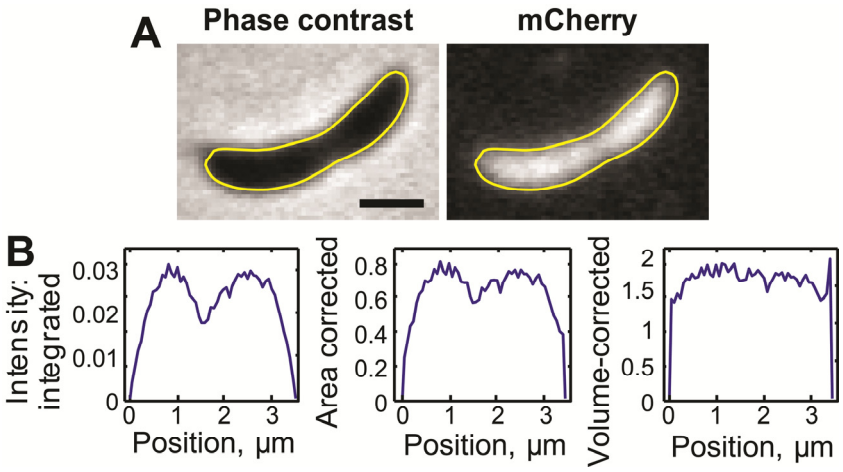

Fig. S2. Normalization of a fluorescence profile. A. Phase contrast and fluorescence images are shown for a *C. crescentus* cell expressing the fluorescent protein mCherry (strain CJW3097, induced for 2 h with 500 μM vanillic acid). **B.** The integrated (uncorrected) intensity profile of the cell in (A) and the same profile corrected by normalization to the area and the volume of each segment. The profile is the flattest in the volume-corrected case, indicating equal concentration of the protein inside the cell, but the errors near the poles are the highest (resulting from less precise volume estimation). The area-corrected case provides partial compensation for the segment size difference and corresponds to the average intensity in a segment as seen on the images. Bar: 1 μm.

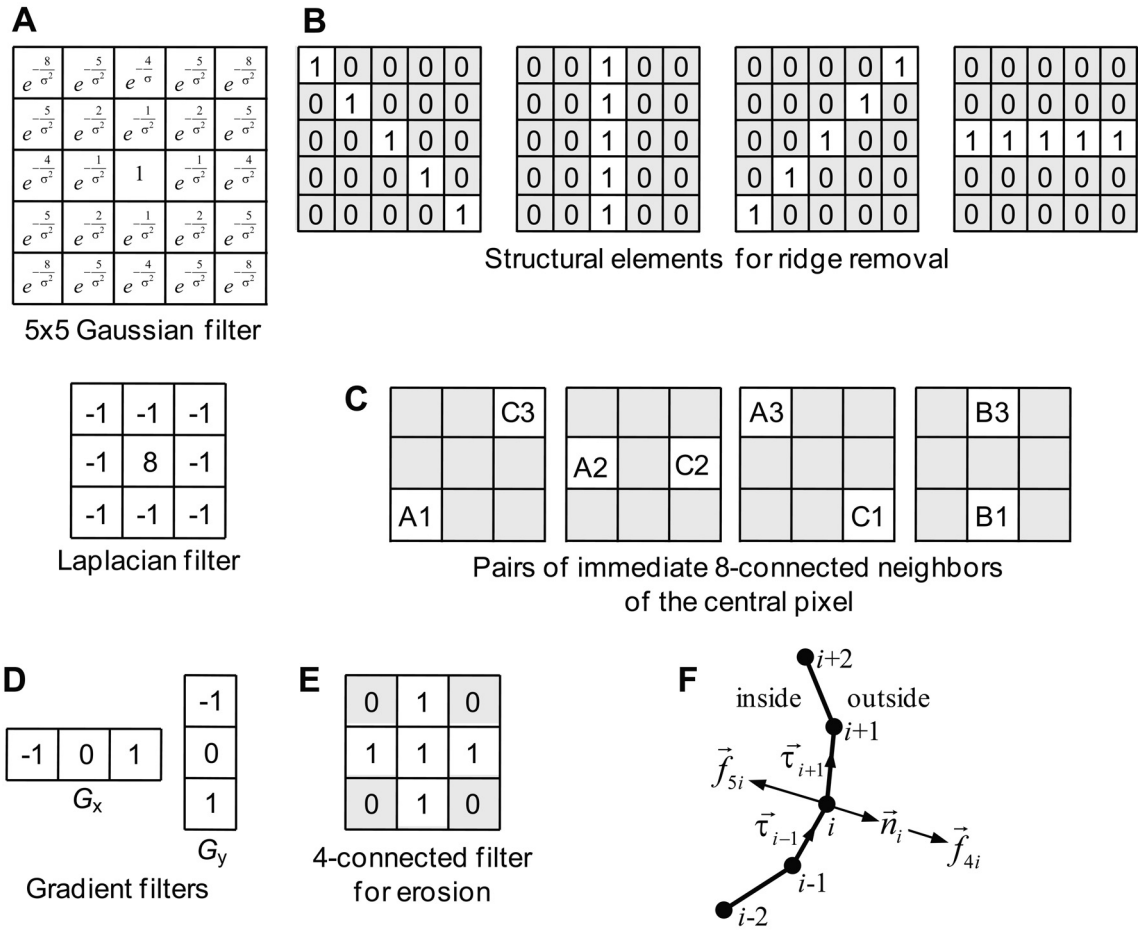

Fig. S3. Morphological operations and force calculation. A. Gaussian and Laplacian filter matrices used in edge detection. The Gaussian matrix depends on the width parameter σ, the size of the matrix is adjusted to ~ 4 σ. **B.** Structural elements for ridge removal. **C.** Pairs of neighbors of the central pixel used in the valley detection algorithm. **D.** Gradient matrices used in image force calculation. **E.** Four-connected filter matrix, used for erosion operation in calculating distance maps. **F.** Non-gradient-based components of the image forces.

Supplementary Movie Legends

Movie S1. Outlining of *C. crescentus* cells expressing FtsZ-YFP during growth and division on a pad. *C. crescentus* cells expressing FtsZ-YFP under the control of vanillic acid-inducible promoter P_{van} (strain MT196) were grown in M2G medium with vanillic acid for 4 h prior to 10h-imaging on agarose pads containing the same medium. The cells were imaged in a time-lapse series; the progeny of a single cell is shown. The timestamp is indicated in hours : minutes.

Movie S2. Oscillations of YFP-MinD in *min+ E. coli*. Time-lapse sequence of YFP-MinD in a group of *E. coli* cells outlined by MicrobeTracker. Strain MC1000/pWM1409/pFx40 was grown in M9-maltose, preinduced with 10 μM IPTG for 4 h and imaged on agarose-padded slides containing M9-maltose and IPTG. The images were taken at 2.5 s intervals for 20 min. The timestamp is indicated in minutes : seconds.

Movie S3. Oscillations of YFP-MinD in a *min⁻ E. coli* cell. Time-lapse sequence of YFP-MinD in a short *E. coli* cell outlined by MicrobeTracker. Strain CJW3672 was grown in M9-glycerol, preinduced with 20 μM IPTG for 4 h and imaged on agarose-padded slides containing M9-glycerol and IPTG. The images were taken at 3-s intervals for 30 min. The cell selected is the same as in Fig. 5A *Top*. The time stamp is indicated in minutes : seconds.

Movie S4. Oscillations of YFP-MinD in an elongated *min⁻ E. coli* cell. Time-lapse sequence of YFP-MinD in an elongated *E. coli* cell outlined by MicrobeTracker. Strain CJW3672 was grown in M9-glycerol, preinduced with 20 μM IPTG for 4 h and imaged on agarose-padded slides containing M9-glycerol and IPTG. The images were taken at 3 s intervals for 30 min. The cell selected is the same as in Fig. 5A *Bottom*. The timestamp is indicated in minutes : seconds.

Movie S5. Stochastic shifting of YFP-MinD in a *min⁻ E. coli* cell. Time-lapse sequence of YFP-MinD in an *E. coli* cell outlined by MicrobeTracker. This cell displays stochastic shifting of YFP-MinD signal distribution between two locations. Strain CJW3672 was grown in M9-glycerol, preinduced with 100 μM IPTG for 4 h and imaged on agarose-padded slides containing M9-glycerol and IPTG. The images were taken at 2.5 s intervals for 25 min. The timestamp is indicated in minutes : seconds.

Movie S6. Stochastic shifting of YFP-MinD in an elongated *min*$^-$ *E. coli* cell. Time-lapse sequence of YFP-MinD in an elongated *E. coli* cell outlined by MicrobeTracker. This cell displays stochastic shifting of YFP-MinD signal distribution between several locations. Strain CJW3672 was grown in M9-glycerol, preinduced with 100 μM IPTG for 4 h and imaged on agarose-padded slides containing M9-glycerol and IPTG. The images were taken at 2.5 s intervals for 25 min. The timestamp is indicated in minutes : seconds.

Movie S7. Origin replication and segregation in filamenting *C. crescentus* cells. Time-lapse sequence of phase contrast and corresponding fluorescence images of the cell block for cell division (also shown in Fig. 7A-B). For this experiment, CJW3673 cells (CB15N *ftsZ*::*Pxyl-ftsZ parB*::*cfp-parB*) were depleted of FtsZ following synchrony, after which CFP-ParB was imaged by time-lapse microscopy. The outline produced by MicrobeTracker is shown in yellow with the "stick" bar indicating the old pole. The timestamp is indicated in hours : minutes.

MicrobeTracker program

The package was developed in MATLAB with insertions in C++. It includes the main graphical user interface-based program MicrobeTracker and a set of accessory components, both graphical and command line based. To run, the package requires MATLAB with the Image Processing Toolbox. The description of the operation principles of the main MicrobeTracker program and of its accessory program SpotFinder are presented below. The program itself, usage examples, and a detailed user guide for all components of the program are available at http://www.microbetracker.org.

Preparation of images

Before the analysis, the images have to be converted to a standard format, in which cells are light on a dark background. This is achieved automatically when using a diffuse fluorescent dye to label cells. Phase contrast microscopy images have to be simply inverted (i.e., light colors replaced with dark ones).

DIC images can also be processed into the aforementioned format, but this operation requires integration of the image and is very sensitive to slight variations of background intensity. In order to do so, we developed an accessory program. First, this program finds the high contrast areas of the image and dilates them to cover all cells and other objects in the image. A mask covering the remaining part of the image (cell-free mask) is therefore produced. Second, the program assumes that the DIC image represents a derivative of the image to be reconstructed in a certain direction (principal direction) plus a constant, which is determined by integrating the DIC image along the principal direction between two points of the cell-free mask and assuming that the background is flat and that all intensity variance is coming from cells. If the principal direction is not known (typically it is diagonal

in the field of view, but could be changed by rotating the camera), the program repeats the second step iteratively to maximize the contrast of the reconstructed image. In practice, such processing of DIC images provides suitable quality only for sparsely distributed cells.

Pixel-based operations

Here we present the sequence of pixel-based operations used to segment the image and to separate cells. These operations are performed on the initial stage of cell detection and consist of thresholding, edge detection, and splitting large cells as follows:

1. Thresholding

To compute the threshold, we use Otsu's method (Otsu, 1979) minimizing the intraclass variance of 'black' and 'white' pixels, as it is implemented in MATLAB (MATLAB's "greythresh" function, Fig. S1B). Optionally, the threshold can be adjusted by the user by multiplying it by a user-supplied factor.

2. Edge detection

An important problem with initial segmentation of an image by thresholding alone is that cells in a cluster (cells touching each other) cannot be resolved (Fig. S1B). To solve this problem, we empirically evaluated various standard edge detection algorithms provided in MALAB (MATLAB's "edge" function). In these tests, the Laplacian of Gaussian (LoG) algorithm proved to be the most efficient one for separating touching cells. This method sequentially computes the convolution of the image with two matrices, called Gaussian and Laplacian filters (Fig. S3A), and finds the points of zero crossing of the resulting image.

However, the LoG algorithm fails in some instances, in which cases the user can use an optional valley detection algorithm that we developed. Though this method requires more parameters which are estimated manually, it can produce superior results (Fig. S1C-D). The valley detection algorithm is a version of ridge detection algorithms that depends on three parameters: smoothing parameter and strong and weak thresholds. First, the image is smoothed with a Gaussian filter of the width determined by the smoothing parameter. Then, for each pixel, 4 pairs of neighboring pixels are identified (Fig. S3C). For example, using a chessboard analogy, the pair of neighbors for the pixel B2 would be: A1 and C3, A2 and C2, A3 and C1, and B1 and B3. Then, the 'valley depth' is determined for the pixel as the maximum for all pairs of neighbors of the minimum difference between the pixels of the pair and the pixel in consideration, i.e. for pixel B2, the valley depth V would be: $V = \max[\min(I_{A1}-I_{B2}, I_{C3}-I_{B2}), \min(I_{A2}-I_{B2}, I_{C2}-I_{B2}), \min(I_{A3}-I_{B2}, I_{C1}-I_{B2}), \min(I_{B3}-I_{B2}, I_{B1}-I_{B2})]$, where I_x is the intensity of the pixel x. The pixels with the valley depth above the strong threshold are always considered a part of the 'valley'. The pixels with the valley depth between the weak and the strong thresholds are considered a part of the valley if they are immediately adjacent (using the 8 pixels neighborhood) to a pixel with the valley depth above the strong threshold.

The images obtained by thresholding (with 'true' corresponding to cells and 'false' to the background) and edge detection (with 'false' corresponding to edges) are combined using the logical pixel-wise AND operation.

3. Splitting

The resulting image is then processed region-by-region, eliminating regions smaller than the minimum allowed area and trying to split those exceeding the maximum area. The regions that have to be split are segmented into two subregions in a single step using an algorithm we developed based on the watershed algorithm (Meyer, 1994). As the first step, the algorithm finds the lowest 'mountain pass' (a saddle point of image intensity) connecting any two 'mountain peaks' (local maxima of image intensity) in the region. As the second step, the algorithm separates the two peaks by creating a dividing line between them along the two 'valleys' exiting the path. The splitting continues until all resulting regions become smaller than the maximum size or until the maximum number of iterations has been reached.

4. Intermediary output

In some cases, the user may desire the output already at this point, which may be useful when irregularly shaped or atypical cells are used. In this case, each region is outlined along its border. Then, the outline is Fourier-transformed, keeping a fixed number of descriptors; this number is preset in the parameters and has to be increased with increasing irregularity in cell shape. The remaining descriptors are Fourier-transformed back, creating a smoothed outline, for which a mesh can be generated (see below for mesh generation).

Image forces

The shape of a cell in active contour methods is described by a contour, which can smoothly move in a way that finally converges to the ideal cell shape (Kass *et al.*, 1988). The motion is driven by the action of 'image forces' sometimes called 'external forces'. We have tested multiple methods of force generation. For the actual program, we use a linear combination of the following 5 components, for which the relative weights can be controlled by the user. While the first three forces are computed as gradients of energy at the nodes, the remaining two are calculated directly based on the image intensity in the vicinity of the nodes:

$$\vec{f} = \underbrace{-\nabla a\left(\left|\nabla I\right|\right)}_{\substack{\text{attraction to}\\ \text{high gradient}}} + \underbrace{\nabla b(I - I_0)}_{\substack{\text{attraction to}\\ \text{threshold}}} + \underbrace{\nabla c(D)}_{\substack{\text{attraction to}\\ \text{edge}}} + \underbrace{\vec{n}d(I / I_0)}_{\substack{\text{attraction to}\\ \text{filled areas}}} - \underbrace{\vec{n}d(I / I_0)}_{\substack{\text{repulsion from}\\ \text{background}}}.$$

The first force component is the attraction to the areas of high local intensity gradient. The corresponding energy is a monotonic function (*a*) of the absolute image (*I*) gradient value.

The second force component is attraction to the areas of intensity close to the threshold. The corresponding energy is a monotonic function (*b*) of the absolute value of the difference between the image intensity and the threshold (I_0).

The third force component is the attraction to the detected edge. The edges were found the same way as before; the corresponding energy is a monotonic function (*c*) of the distance map to the closest edge (*D*).

The fourth force component is the attraction to the areas with intensity above threshold (i.e., cell) outside of the contour, while the fifth force component is the repulsion from areas below threshold (i.e., background) inside of the contour. These forces are directed outwards or inwards from the contour, respectively, along the outwards

normal $\vec{n}$ and increase as a monotonic function (d) of the ratio of the image intensity and threshold intensity in a region around the given point. The regions for these two force components were chosen outside and inside of the contour, respectively. These forces are usually set to produce a small contribution to the total force when the contour is close to its equilibrium position, but they help to resolve the cases when the contour is significantly away from the boundaries of the cell, for example when the guess was produced with a significant error on the pixel-based stage or if the cell moved or deformed noticeably in a time-lapse sequence.

The exact way of calculating the forces from the discretized image is presented below. The first three force components are obtained as the gradients to the energy $W=W_1+W_2+W_3$ calculated on-lattice. The image matrix name is I, all algebraic operations here are performed pixel-wise.

1) Local intensity gradient:

$$W_1 = -\frac{w_1}{q^2 + w_1}, \text{ where } w_1 = \left(G_x \otimes I\right)^2 + \left(G_y \otimes I\right)^2$$

Here matrices G_x and G_y are the gradient filters, see Fig. S3D; symbol $\otimes$ denotes the convolution operation, also called filtering, constraining the dimensions of the image matrix I; q is obtained by averaging $\sqrt{w_1}$ over the image.

2) Difference from the threshold intensity (scalar threshold t is calculated as in the Thresholding section of the Online Methods):

$$W_2 = \left(I - I_0\right)^2$$

3) Distance from the edge (E – edge matrix, I_0 and α – coefficients with the default values I_0=2, α=2; R – 4-connected erosion filter, see Fig. S3E; logical true=1 and false=0):

$$W_3 = \frac{w_3^\alpha}{I_0^\alpha + w_3^\alpha}, \text{ where } w_3 = \sum_{i=0}^{\infty} \underbrace{R \otimes (R \otimes (R \otimes ...}_{i} (\text{not } E)...)))$$

The resulting force components are calculated as:

$$F_{123x} = -G_x \otimes \left(W_1 + W_2 + W_3\right)$$

$$F_{123y} = -G_y \otimes \left(W_1 + W_2 + W_3\right)$$

The values of the force at the contour nodes are obtained from these force matrices:

$$f_{123ix} = F_{123x}\left(x_i\right),$$

$f_{123iy} = F_{123y}(x_i)$,

Here, the values between pixels are calculated by bilinear interpolation.

4-5)The next two force components are calculated directly off-lattice and are directed along the normal to the contour, calculated in the node *i* as $\vec{n}_i = \left(\vec{z} \times \vec{\tau}_{i-1/2} + \vec{z} \times \vec{\tau}_{i+1/2}\right) \Big/ \left|\vec{z} \times \vec{\tau}_{i-1/2} + \vec{z} \times \vec{\tau}_{i+1/2}\right|$, where $\vec{z}$ is the normal to the image plain, $\vec{\tau}_{i-1/2}$ and $\vec{\tau}_{i+1/2}$ - vectors connecting nodes *i*-1 to *i* and *i* to *i*+1, respectively, the orientation of the vertices is counterclockwise if looking along $\vec{z}$ (i.e., when $\vec{z}$ is directed into the image plain), and $\times$ denotes the vector cross product (Fig. S3F).These forces are split into two components, attraction ($\vec{f}_{4i}$) and repulsion ($\vec{f}_{5i}$):

$\vec{f}_{4i} = \dfrac{\vec{g}_{4i}^{\beta}}{1 + \vec{g}_{4i}^{\beta}}$, where $\vec{g}_{4i} = \vec{n}_i \cdot \sum_{j=0}^{\gamma} I\left(\vec{x}_i + j\vec{n}_i\right) / I_0$;

$\vec{f}_{5i} = \dfrac{1}{1 + \vec{g}_{5i}^{\beta}}$, where $\vec{g}_{5i} = \vec{n}_i \cdot \sum_{j=0}^{\gamma} I\left(\vec{x}_i - j\vec{n}_i\right) / I_0$.

Here β and γ – coefficients with the default values β=4, γ=4. The values of the image intensity in the points between pixels are obtained by bilinear interpolation.

Point Distribution Model

Refinement of cells contours and cell detection in time-lapse image series are performed by one of the two algorithms of active contour models (Kass *et al.*, 1988) which differ in the way of imposing constraints on cell shapes: the Point Distribution Model and the Manually Constraint Contour. Consequently, these models differ in the speed of operation, in range of cell types they can handle, and in the method of adjusting their parameters to fit a new cell type.

Point Distribution Model (PDM) was proposed as a general tool to identify objects in an image belonging to a certain class (Cootes *et al.*, 1995). This model requires ‘training’ at the beginning on a set of objects known to belong to the class of interest. MicrobeTracker provides a button in the GUI to perform training. The protocol for the training procedure is provided in the program’s help.

In practice, training only needs to be performed for very different cell types. We provide two ‘standard’ training files obtained for *C. crescentus* CB15N and *E. coli* MC1000 strains, which we processed and tested. These sets will also work for any type of cells that are visually similar, which is essentially most strains of non-filamentous rod-shaped bacteria. The training procedure uses a set of cells outlined by other methods, usually sparsely located cells outlined by simple thresholding. In this procedure, the program extracts the cell pole positions and distributes a fixed number of points between the two poles. The number of these points is set in the

parameters and should approximately equal to the size of the cells in pixels; we used 52 pairs of points for the provided training sets. Then, all the contours are shifted and aligned, calculating the ‘average cell’ by averaging the coordinates of corresponding points. The alignment procedure minimizes the mean square distance from the contour points to the corresponding average cell points. For each cell, a difference from the ‘average cell’ is calculated and the set of difference vectors is analyzed using the Principal Component Analysis (PCA) method in order to find the most variable coordinate combinations. MicrobeTracker keeps only a predefined number of the most important combinations returned by PCA, with their weights described by a set of descriptors of equal length. We used 11 descriptors for the tested set; this number has to be increased with the complexity of the cell shape. This procedure reduces the dimensionality of the set of descriptors (in our case from $52 \times 2 = 104$ to 11 descriptors) and defines the implicit constraints for the cell shape.

In order to fit the PDA model to a cell, MicrobeTracker calculates a projection of the image forces on the principal components obtained during training. Thus, the motion is performed in the principal components space. The descriptors then change under the force until equilibrium is reached or a fixed number of iteration is performed.

For the initial guess, the average cell is aligned to the orientation of the segmented region and allowed to align to the region using the force produced by the distance map from the boundary alone. After fitting, the fitted model is converted back to a contour, which is used to generate a mesh (see below).

Cell detection by this method works well for wild type-looking cells (similar to those in the training set) and is essentially parameter free from the user's point of view. Nevertheless, this method depends on a few other parameters (independent of the PCA procedure) that can be adjusted by hand in order to improve cell detection in complex situations (see the program help for full details). For cells that diverge significantly from wild-type morphology, especially when shape defects are studied, the method may enforce the shape to be similar to the training set cells and therefore is not recommended.

Manually Constrained Contour Model

An alternative to PDM is the Manually Constrained Contour (MCC) model. As the name suggests, this algorithm is based on manually preset constraints instead of those determined automatically from a training set. This method was initially designed for filamentous cells, but was successfully applied to most other cell types. One advantage of this method is that it can accommodate shapes that are impossible to represent by a linear combination of descriptors (e.g., for filamentous cells). Another advantage is that this method does not require training and can instead be adjusted manually by a set of straightforward changes in a few parameters, saving time when working with a new cell type. In addition, if a particular property of the shape is measured or a particular defect of cell shape is expected, the constraints on that property can be manually relaxed.

The constraints in the program are defined explicitly and result in constraint forces or internal forces. Here the contour in described by assuming that a cell is rod-shaped with approximately uniform width and rounded poles. The contour is represented by pairs of points on opposite sides of the cell connected by imaginary ‘ribs’.

The ribs are considered of equal length except for the polar regions, where they shorten according to a predefined law. The equilibrium shape of the cell is thus a straight rod of equal width with rounded poles. However, soft constraints allow for significant deviations from such shape.

The program tries to keep the pairs of points on the opposite sides of the cell, while keeping the cell from being too curved and too jagged etc. The particular forces are as follows:

a) cell width forces, equivalent to rigidity of the 'ribs';

b) rib bending forces that keep the ribs locally perpendicular to the centerline;

c) centerline rigidity;

d) contour rigidity;

e) equidistant location of contour points.

The explicit constraint components used in the MCC algorithm are the following:

a) Cell width forces, rigidity of the 'ribs':

$$\vec{f}_{ai} = \frac{\vec{r}_i}{|\vec{r}_i|}\left(w_i - |\vec{r}_i|\right),$$

where $\vec{r}_i = \vec{x}_i - \vec{x}_{\tilde{i}}$ is the vector connecting the node *i* at the position $\vec{x}_i$ and the position of its counterpart $\tilde{i}$ on the opposite side of the cell $\vec{x}_{\tilde{i}}$; w_i is the width of the corresponding rib connecting the nodes *i* and $\tilde{i}$, which is constant along the cell body and decreases smoothly at the poles.

b) Rib bending forces that keep the ribs locally perpendicular to the centerline

$$\vec{f}_{bi} = \left(\vec{\tau}_i - \vec{\tau}_{\tilde{i}}\right)\alpha\left(\vec{\tau}_i - \vec{\tau}_{\tilde{i}}, \vec{r}_i\right),$$

here $\vec{\tau}_i$ and $\vec{\tau}_{\tilde{i}}$ are the tangentials to the contour at the node *i* and at its counterpart $\tilde{i}$; $\alpha(\vec{v}_1, \vec{v}_2)$ is the angle function between vectors $\vec{v}_1$ and $\vec{v}_2$ (positive if the orientation of the nodes is counterclockwise when looking along $\vec{v}_1 \times \vec{v}_2$ and negative otherwise).

c) Centerline rigidity

$$\vec{f}_{ci} = \left[\vartheta_{-1}\left(\vec{c}_i, \vec{c}_{i+1}, \vec{c}_{i+2}\right) + \vartheta_0\left(\vec{c}_{i-1}, \vec{c}_i, \vec{c}_{i+1}\right) + \vartheta_1\left(\vec{c}_{i-2}, \vec{c}_{i-1}, \vec{c}_i\right)\right] \otimes K(w_b),$$

where $\vec{c} = \frac{\vec{x}_i + \vec{x}_{\tilde{i}}}{2}$ is the position of the centerline node corresponding to the node *i*, $\vartheta_k\left(\vec{v}_1, \vec{v}_2, \vec{v}_3\right)$ is the elastic force of a string produced by the spring at node $\vec{v}_2$ connected to nodes $\vec{v}_1$ and $\vec{v}_3$, at the *k*-th node, i.e.

$$\vec{\vartheta}_1(\vec{v}_1,\vec{v}_2,\vec{v}_3)=\vec{z}\times\frac{\vec{v}_1-\vec{v}_2}{|\vec{v}_1-\vec{v}_2|}\alpha(\vec{v}_1-\vec{v}_2,\vec{v}_2-\vec{v}_3),$$

$$\vec{\vartheta}_3(\vec{v}_1,\vec{v}_2,\vec{v}_3)=\vec{z}\times\frac{\vec{v}_2-\vec{v}_3}{|\vec{v}_2-\vec{v}_3|}\alpha(\vec{v}_1-\vec{v}_2,\vec{v}_2-\vec{v}_3),$$

$$\vec{\vartheta}_2(\vec{v}_1,\vec{v}_2,\vec{v}_3)=-\vec{\vartheta}_1(\vec{v}_1,\vec{v}_2,\vec{v}_3)-\vec{\vartheta}_3(\vec{v}_1,\vec{v}_2,\vec{v}_3),$$

where $\vec{z}=\frac{(\vec{v}_1-\vec{v}_2)\times(\vec{v}_2-\vec{v}_3)}{|\vec{v}_1-\vec{v}_2||\vec{v}_2-\vec{v}_3|}$; $K(w_b)$ is a sharp-peaked kernel of width w_b.

d) Contour rigidity. For the nodes away from the poles, this component is similar to the centerline rigidity, though with a narrower kernel:

$$\vec{f}_{di}=\left[\vartheta_{-1}(\vec{x}_i,\vec{x}_{i+1},\vec{x}_{i+2})+\vartheta_0(\vec{x}_{i-1},\vec{x}_i,\vec{x}_{i+1})+\vartheta_1(\vec{x}_{i-2},\vec{x}_{i-1},\vec{x}_i)\right]\otimes K(w_r)$$

For the polar regions, an offset to the equilibrium angle is added.

e) Equidistant positioning of contour nodes

$$\vec{f}_{ei}=\vec{\tau}_i\left[\vec{\tau}_i\cdot(\vec{x}_{i+1}+\vec{x}_{i-1}-2\vec{x}_i)\right]$$

All these components are added together with different user-supplied coefficients. In practice, the summation coefficients and the coefficients mentioned in the description rarely need to be changed, and a parameter set optimized empirically for the best fit quality, convergence time, and stability of the algorithm is supplied with the program and is used in most cases. Small adjustments, however, may need to be made to accommodate for a particular cell type and can be made in an obvious way. For example, if all the cells are straight rod-shaped, the centerline rigidity should be increased. Conversely, if curvature measurements are the point of the experiment, it should be reduced from the default values. In a similar fashion, filamentous cells are better recognized in dense regions if the cell width rigidity is high. However, it has to be reduced to detect cell constriction or to perform precise cell width measurements.

Mesh construction

The mesh is stored as 2D coordinates of two semi-contours, corresponding to the ‘left’ and ‘right’ sides of the cell, counting from one of the poles. It is constructed in order to generate output in all methods as well as to create an initial shape to be used with the MCC model and to resample the points of this model before going to the next frame in a time-lapse sequence (see below).

The first step in mesh construction is Fourier-smoothing of the outline. In this procedure, the outline is Fourier transformed, a fixed number of the lowest frequency descriptors is kept and transformed back. The

number of descriptors kept has to depend on the complexity of the shape, increasing from the default of 16 for wild-type *C. crescentus* cells to about 100 for long filamenting cells.

Then the centerline has to be calculated. To do so, we first calculate the 'skeleton' of the shape by applying Voronoi transform (MATLAB's "voronoi" function) to the set of contour points. Then all the nodes of the transform outside the contour are removed and the remaining points are degraded from the ends until the remaining part of the skeleton has no branches. This shape represents the true centerline of the cell; however, it does not reach its poles.

In order to handle the poles, the centerline is extended using smoothing spline extrapolation. Then ribs are constructed as perpendiculars to the centerline, identifying their intersections with the contour. For each pair of points, the true center is found, from then on being a point of the refined centerline. This refined centerline is again spline-smoothed and the process of refinement repeated iteratively until the desired error between the old and the refined centerlines is reached.

The resulting mesh has all the ribs equidistantly spaced along the centerline (by 1 pixel by default) and oriented perpendicularly to the centerline.

Time-lapse series analysis

The first preparation step in analyzing time-lapse image sets is image alignment. It is performed by calculating the correlation coefficient between the two consecutive images, then shifting one of them by 1 pixel in either direction including diagonal and calculating the maximum correlation among all 9 pixels. The process is repeated until the correlation coefficient stops increasing, at this point the frames are considered aligned. To speed up the alignment process, the programs start with 4 pixel steps, reducing the step size to 2 pixels and then 1 pixel when the correlation coefficient stops growing. We have tested the method for different drift levels and demonstrated perfect alignment for a drift that is not too large. For the cases of extreme drift when correlation-based alignment methods are not feasible, we developed a manually-assisted alignment tool.

In time-lapse sequences, the morphological operations are only used on the first frame to produce the initial guess. In later frames, the active contour models use the shape determined on the previous frame as the initial guess.

This method has proved to be noticeably more efficient in resolving touching cells than *de novo* detection, i.e. starting from the morphological operations. In addition, this method automatically keeps the identity of each cell and of each cell pole, and is efficient in tracing cell lineages. In our experiments with *C. crescentus*, this method typically worked until the cells got noticeably deformed or pushed out of the focal plane because of overcrowding and thus could not be reliably separated neither by the program nor by eye. A possible drawback of the algorithm is that the cells could be misdetected in the case of significant drift (usually solved by image alignment) or when significant growth occurs between time frames (which can generally be solved by reducing the time interval between image acquisition events).

SpotFinder program

This tool identifies and quantifies small fluorescent spots. These spots result either from intracellular objects smaller than the diffraction limit, or from larger round-shaped structures.

First, the program applies 2D band-pass filtering that enhances the objects of correct size completely or partially, eliminating both pixel-to-pixel noise and large objects, such as whole cells (Fig. 1E, First step).

Second, the program applies a ridge-removal algorithm that we developed. This algorithm applies morphological erosion with the structural element shaped as a line oriented in four principal directions (horizontal, vertical and two diagonals; Fig. S3B), and then assigns the minimum difference of a pixel value and the corresponding value of the 4 eroded images. This algorithm favors round-shaped spots over elongated ones and efficiently eliminates ‘ridges’ – long lines of bright pixels typical for the centerlines of thin cells (Fig. 1E, Second step). The output of this second step provides an initial guess of the position, height and width of each spot.

Third, the initial guess for each spot is used to fit a 2D Gaussian plus a constant (background level) to the original image, varying the width, the coordinates of the center, and the background level (Fig. 1E, Third step). During this fitting procedure, the position of the Gaussian is constrained within a few pixels of the original guess obtained from the Second Step. The position and the magnitude (as well as the direct fit parameters: the width, height and background level) of each spot are added to a standard MicrobeTracker data structure.

Supplementary Tables

Table S1. Strain and plasmid table

Strain or plasmid	Relevant genotype or description	Reference or source
***C. crescentus* strains**		
CB15N	Synchronizable variant of CB15 (also NA1000)	(Evinger & Agabian, 1977)
CJW826	CB15N *divJ*::*divJ-yfp*	(Lam *et al.*, 2003)
CJW3097	CB15N *vanA*::pHL23-Pvan-mcherry-lacO$_{120}$tm	(Montero Llopis *et al.*)
CJW3673	CB15N *ftsZ*::pBGENTPxylftsZ *parB*::*cfp-parB*	This study
MT190	CB15N *parB*::*cfp-parB*	(Thanbichler & Shapiro, 2006)
MT196	CB15N *vanA*::pMT383	(Thanbichler & Shapiro, 2006)
MT16	CB15N Cori::CoriTetOpGent Ter::TerLacOpHygro, carrying plasmid pHPV472	(Viollier *et al.*, 2004)
***E. coli* strains**		
CJW3672	JS964 carrying pFx40 plasmid	This study
JS964	MC1061 *malPp*::*lacI*^{q} Δ(*minCDE*)::*kan*	(Pichoff *et al.*, 1995)
MC1000	K12 *ara*D139 Δ (*ara*, *leu*)7697 Δ*lac* X74 *galU galK strA*	(Casadaban & Cohen, 1980)
MC1061	MC1000 *galU*$^{-}$ *galK*$^{-}$ *hsr*$^{-}$ *hsm*$^{-}$	(Casadaban & Cohen, 1980)
	MC1000/pWM1409/pFx40	(Shih *et al.*, 2005)
Plasmids		
pBGENTPxylftsZ	Integrative vector carrying *ftsZ* under P*xyl* control	(Cabeen *et al.*, 2009)
pFx40	Replicative vector carrying *yfp-minD* and *minE* under P*lac* control	(Shih *et al.*, 2002)
pHPV472	Integrative vector carrying *lacI-cfp* and *tetR-yfp* under P*xyl* promoter	(Viollier *et al.*, 2004)
pMT383	Integrative vector carrying *ftsZ-yfp* under P*van* control	(Thanbichler & Shapiro, 2006)
pWM1409	Replicative vector carrying *ftsZ-cfp* under P*ara* control	(Shih *et al.*, 2005)

Strain construction

CJW3672: Plasmid pFx40 was purified from strain MC1000/pFx40/pWM1409 and electroporated into strain JS964. The clones were selected for ampicillin resistance and checked for chloramphenicol sensitivity.

CJW3673: The *ftsZ*::pBGENTPxylftsZ locus from strain CJW2591 (CB15N Δ*bla ftsZ*::pBGENTPxylftsZ, M. Cabeen and C. Jacobs-Wagner, unpublished) was moved by ΦCR30 phage transduction into strain CJW2069 (CB15N *parB*::*cfp-parB*) (Thanbichler & Shapiro, 2006).

References

Cabeen, M. T., G. Charbon, W. Vollmer, P. Born, N. Ausmees, D. B. Weibel & C. Jacobs-Wagner, (2009) Bacterial cell curvature through mechanical control of cell growth. *EMBO J* **28**: 1208-1219.

Casadaban, M. J. & S. N. Cohen, (1980) Analysis of gene control signals by DNA fusion and cloning in *Escherichia coli*. *J Mol Biol* **138**: 179-207.

Cootes, T. F., C. J. Taylor, D. H. Cooper & J. Graham, (1995) Active shape models - their training and application. *Comp Vis Image Under* **61**: 38-59.

Evinger, M. & N. Agabian, (1977) Envelope-associated nucleoid from *Caulobacter crescentus* stalked and swarmer cells. *J Bacteriol* **132**: 294-301.

Kass, M., A. Witkin & D. Terzopoulos, (1988) Snakes: active contour models. *Int J Comput Vis* **1**: 321-331.

Lam, H., J. Y. Matroule & C. Jacobs-Wagner, (2003) The asymmetric spatial distribution of bacterial signal transduction proteins coordinates cell cycle events. *Dev Cel* **5**: 149-159.

Meyer, F., (1994) Topographic distance and watershed lines. *Signal Proc* **38**: 113-125.

Montero Llopis, P., A. F. Jackson, O. Sliusarenko, I. Surovtsev, J. Heinritz, T. Emonet & C. Jacobs-Wagner, (2010) Spatial organization of the flow of genetic information in bacteria. *Nature* **466**: 77-81.

Otsu, N., (1979) A threshold selection method from gray-level histograms. *IEEE T Syst Man Cyb* **9**: 62-66.

Pichoff, S., B. Vollrath, C. Touriol & J. P. Bouche, (1995) Deletion analysis of gene *minE* which encodes the topological specificity factor of cell division in *Escherichia coli*. *Mol Microbiol* **18**: 321-329.

Shih, Y. L., X. Fu, G. F. King, T. Le & L. Rothfield, (2002) Division site placement in *E.coli*: mutations that prevent formation of the MinE ring lead to loss of the normal midcell arrest of growth of polar MinD membrane domains. *EMBO J* **21**: 3347-3357.

Shih, Y. L., I. Kawagishi & L. Rothfield, (2005) The MreB and Min cytoskeletal-like systems play independent roles in prokaryotic polar differentiation. *Mol Microbiol* **58**: 917-928.

Thanbichler, M. & L. Shapiro, (2006) MipZ, a spatial regulator coordinating chromosome segregation with cell division in *Caulobacter*. *Cell* **126**: 147-162.

Viollier, P. H., M. Thanbichler, P. T. McGrath, L. West, M. Meewan, H. H. McAdams & L. Shapiro, (2004) Rapid and sequential movement of individual chromosomal loci to specific subcellular locations during bacterial DNA replication. *Proc Natl Acad Sci USA* **101**: 9257-9262.