vertX3D Documentation

Version 0
June 20, 2019

1. INSTALLATION

To install the vertissue3D, you need a C++ compiler with C++11 support (supported by most new compilers, GCC, Intel ICPC, ...). To enable parallel computing, the OpenMP library is required. Additionally, CMAKE version 3.5.1 is required.

Please follow these 3 steps to install vertX3D:

1) Clone the latest source from https://www.github.com/Shvartsman-Lab/vertX3D in a directory of your choice.

```
git clone https://www.github.com/Shvartsman-Lab/vertX3D.git
```

2) Enter the build directory and use CMAKE to generate a MakeFile:

```
cd vertX3D
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
```

If instead you are debugging the code, replace Release with Debug. Be warned the code will run slower in debug mode.

3) Compile the examples:

```
make
```

If for any reason, the build folder is deleted, the initial folder and its contents must be re-downloaded from the repository, and a new output folder must be created.

1.1 RUNNING EXAMPLES

The general form of running the code is as follows

```
./name_of_executable [array_max] [tmax] [seed] [num_threads]
```

If a segmentation fault is encountered when running the code, it is likely that array_max is too small, and an array requires more entries than allocated. seed is used to seed the random number generator which for example is utilized in simulations with T1 transformations. tmax specifies how long the simulation will run and if num_threads is greater than 1 the code will run in parallel.

For the included example simulations, please use the following commands to run them:

Spreading on a flat surface
```
./vertX3D_spreading_flat 100000 600 1 4
```

Spreading on a sphere
```
./vertX3D_spreading_sphere 100000 600 1 4
```

Homeostasis
```
./vertX3D_homeostasis 100000 20 1 4
```

By default, the homeostasis example will run using the overcrowded initial configuration. If you wish to run using the stretched or normal initial configurations, in `homeostasis.cpp` replace the line:

```
set_initial_fromFile("./initial/overcrowded.vt3d");
```

with

```
set_initial_fromFile("./initial/stretched.vt3d");
```

for the stretched configuration or

```
set_initial_fromFile("./initial/normal.vt3d");
```

for the normal configuration. After modifying the code run `make` in the build folder to re-compile.

Fluidization

```
./vertX3D_fluidization 200000 2000 1 4
```

By default, the fluidization example will run the using annealing on the rate of T1 transformations. To run using a fixed rate of T1s, in `fluidization.cpp` comment out the line:

```
T1_spont_act(0.1,300-300*Time/tmax);
```

and uncomment the line:

```
T1_spont_act(0.1,kT1);
```

`kT1` can be set at the top of `fluidization.cpp`. After modifying the code run `make` in the build folder to re-compile.

Sphere crumpling

```
./vertX3D_sphere 300000 100 1 4
```

1.2 VISUALIZATION

The code will output VTK files which can be easily read by ParaView, an interactive visualization tool. ParaView version 5.6.1 was used during development, however newer versions will likely work too. ParaView can be downloaded from https://www.paraview.org/download/. If you are using Linux, make sure to download “ParaView-5.6.1-MPI-Linux-64bit.tar.gz” not “ParaView-5.6.1-osmesa-MPI-Linux-64bit.tar.gz”.

Once you have ParaView launched, output files can be imported using the upper left folder icon or from the File>Open dropdown menu. The output is separated into 3 layers: the apical, basal and lateral sides which can be viewed separately or all at once. Use the animation controls to play your simulation.

2. INPUT FILE

In the input file, the user specifies the number of basal vertices, the number of basal edges, the number of cells, the size of the simulation box, positions of the basal vertices (x_i, y_i, z_i), normal vector to the basal vertices (n_{xi}, n_{yi}, n_{zi}), cell heights h_i, the ids of head- and tail-vertices of basal edges v_{i1}, v_{i2}, and polygonal classes of cells n_i followed by oriented edge ids oe_{ij} (in anticlockwise direction), where j=1,…n_i. List of oriented edge ids of cell i is followed by 12-n_i zeros (0). Structure of the input file is shown bellow.

```
//fileForm.vt3d
# of basal vertices    # of basal edges       # of cells

x dimension of simulation box          y dimension of simulation box          z dimension
of simulation box

x1      y1      z1      nx1     ny1     nz1     h1
x2      y2      z2      nx2     ny2     nz2     h2
x3      y3      z3      nx3     ny3     nz3     h3
…
xV      yV      zV      nxV     nyV     nzV     hV

v11     v12
v21     v22
v31     v32
…
vE1     vE2

n1      oe11    oe12    oe13    oe14    oe15    …       0       0       0
n2      oe21    oe22    oe23    oe24    oe25    …       0       0       0
n3      oe31    oe32    oe33    oe34    oe35    …       0       0       0
…
nC      oeC1    oeC2    oeC3    oeC4    oeC5    …       0       0       0
```

An example input file regHex16.vt3d creates a flat tissue containing 16 regular hexagonal cells (32 basal vertices and 48 basal edges) in a simulation box of dimensions `perioXYZ`=(3.33073, 2.8845, 10) with periodic boundary conditions. Basal vertices lie in z = 4.0 plane and cell heights are 1.665366.

```
//regHex16.vt3d
32      48      16

3.33073         2.8845          10.

0.080000  0.080000  4.000000  0.000000  0.000000  1.000000  1.665366
0.496342  0.320375  4.000000  0.000000  0.000000  1.000000  1.665366
0.496342  0.801125  4.000000  0.000000  0.000000  1.000000  1.665366
0.080000  1.041500  4.000000  0.000000  0.000000  1.000000  1.665366
0.912683  0.080000  4.000000  0.000000  0.000000  1.000000  1.665366
1.329025  0.320375  4.000000  0.000000  0.000000  1.000000  1.665366
1.329025  0.801125  4.000000  0.000000  0.000000  1.000000  1.665366
0.912683  1.041500  4.000000  0.000000  0.000000  1.000000  1.665366
1.745367  0.080000  4.000000  0.000000  0.000000  1.000000  1.665366
2.161708  0.320375  4.000000  0.000000  0.000000  1.000000  1.665366
2.161708  0.801125  4.000000  0.000000  0.000000  1.000000  1.665366
1.745367  1.041500  4.000000  0.000000  0.000000  1.000000  1.665366
2.578050  0.080000  4.000000  0.000000  0.000000  1.000000  1.665366
2.994392  0.320375  4.000000  0.000000  0.000000  1.000000  1.665366
2.994392  0.801125  4.000000  0.000000  0.000000  1.000000  1.665366
2.578050  1.041500  4.000000  0.000000  0.000000  1.000000  1.665366
0.080000  1.522250  4.000000  0.000000  0.000000  1.000000  1.665366
0.496342  1.762625  4.000000  0.000000  0.000000  1.000000  1.665366
0.496342  2.243375  4.000000  0.000000  0.000000  1.000000  1.665366
0.080000  2.483750  4.000000  0.000000  0.000000  1.000000  1.665366
0.912683  1.522250  4.000000  0.000000  0.000000  1.000000  1.665366
1.329025  1.762625  4.000000  0.000000  0.000000  1.000000  1.665366
```

```
1.329025  2.243375  4.000000  0.000000  0.000000  1.000000  1.665366
0.912683  2.483750  4.000000  0.000000  0.000000  1.000000  1.665366
1.745367  1.522250  4.000000  0.000000  0.000000  1.000000  1.665366
2.161708  1.762625  4.000000  0.000000  0.000000  1.000000  1.665366
2.161708  2.243375  4.000000  0.000000  0.000000  1.000000  1.665366
1.745367  2.483750  4.000000  0.000000  0.000000  1.000000  1.665366
2.578050  1.522250  4.000000  0.000000  0.000000  1.000000  1.665366
2.994392  1.762625  4.000000  0.000000  0.000000  1.000000  1.665366
2.994392  2.243375  4.000000  0.000000  0.000000  1.000000  1.665366
2.578050  2.483750  4.000000  0.000000  0.000000  1.000000  1.665366
```

```
14  1
1  2
2  3
3  4
4  15
2  5
5  6
6  7
7  8
8  3
6  9
9  10
10  11
11  12
12  7
10  13
13  14
14  15
15  16
16  11
30  17
17  18
18  19
19  20
20  31
18  21
21  22
22  23
23  24
24  19
22  25
25  26
26  27
27  28
28  23
26  29
29  30
30  31
31  32
32  27
4  17
8  21
12  25
16  29
20  1
24  5
28  9
32  13
```

```
6  1  2  3  4  5  -18  0  0  0  0  0  0
6  6  7  8  9  10  -3  0  0  0  0  0  0
6  11  12  13  14  15  -8  0  0  0  0  0  0
6  16  17  18  19  20  -13  0  0  0  0  0  0
6  21  22  23  24  25  -38  0  0  0  0  0  0
6  26  27  28  29  30  -23  0  0  0  0  0  0
6  31  32  33  34  35  -28  0  0  0  0  0  0
6  36  37  38  39  40  -33  0  0  0  0  0  0
6  -4  -10  42  -26  -22  -41  0  0  0  0  0  0
6  -9  -15  43  -31  -27  -42  0  0  0  0  0  0
6  -14  -20  44  -36  -32  -43  0  0  0  0  0  0
6  -19  -5  41  -21  -37  -44  0  0  0  0  0  0
```

```
6  -24  -30  46  -6  -2  -45  0  0  0  0  0  0
6  -29  -35  47  -11  -7  -46  0  0  0  0  0  0
6  -34  -40  48  -16  -12  -47  0  0  0  0  0  0
6  -39  -25  45  -1  -17  -48  0  0  0  0  0  0
```

3. GLOBAL VARIABLES

Memory for each global variable is allocated by `allocate()` and deallocated by `deallocate()`.

3.1 Energy-related parameters

`double bet`
Basal surface tension (same for all cells).

`double alph`
Apical surface tension (same for all cells).

`double kV`
Reciprocal isothermal compressibility of cells (same for all cells).

`double V0`
Preferred cell volume (same for all cells).

3.2 Number of geometric elements

`size_t Nv`
Number of vertices (apical+basal).

`size_t Nv_pass`
Number of passive vertices, i.e. vertices at the ceters of cell sides (apical+basal+lateral).

`size_t Ne`
Number of edges (only basal).

`size_t Nf`
Number of triangular facets (apical+basal+lateral).

`size_t Nc`
Number of cells.

3.3 Geometric elements

`double v[Nv+1][4]`
Vertices.
`v[i][0]`=vertex id (if =0, vertex does not exist), `v[i][1]`=x_i, `v[i][2]`=y_i, `v[i][3]`=z_i.

`double v_pass[Nv_pass+1][4]`
Passive vertices.
`v_pass[i][0]`=passive vertex id (if =0, passive vertex does not exist), `v_pass[i][1]`=x_i, `v_pass[i][2]`=y_i, `v_pass[i][3]`=z_i.
`int e[Ne+1][3]`

Basal edges.
`e[i][0]`=edge id (if =0, edge does not exist), `e[i][1]`=vertex 1, `e[i][2]`=vertex 2.

`int f[Nf+1][4]`
Triangular facets.
`f[i][0]`=facet id (if =0, facet does not exist), `f[i][1]`=vertex 1, `f[i][2]`=vertex 2, `f[i][3]`=vertex 3.

`int basal_edges[Nc+1][15]`
Cell bases (edges).
`basal_edges[i][1]`=cell id (if =0, cell does not exist), `basal_edges[i][2]`=number of cell sides, `basal_edges[i][3]`=oriented edge 1, `basal_edges[i][4]`=oriented edge 2, `basal_edges[i][5]`=oriented edge 3...

`int basal_vertices[Nc+1][15]`
Cell bases (vertices).
Derivative from `basal_edges[][]` and `e[][]` by `make_basal_vertices()`.
`basal_vertices[i][1]`=cell id (if =0, cell does not exist), `basal_vertices[i][2]`=number of cell sides, `basal_vertices[i][3]`=vertex 1, `basal_vertices[i][4]`=vertex 2, `basal_vertices[i][5]`=vertex 3...

`int basal_facets[Nc+1][15]`
Cell bases (facets).
`basal_facets[i][1]`=cell id (if =0, cell does not exist), `basal_facets[i][2]`=number of cell sides, `basal_facets[i][3]`=facet 1, `basal_facets[i][4]`=facet 2, `basal_facets[i][5]`=facet 3...

`int apical_facets[Nc+1][15]`
Cell apices (facets).
`apical_facets[i][1]`=cell id (if =0, cell does not exist), `apical_facets[i][2]`=number of cell sides, `apical_facets[i][3]`=facet 1, `apical_facets[i][4]`=facet 2, `apical_facets[i][5]`=facet 3...

3.4 Vertex attributes

`double v_F[Nv+1][4]`
Forces on vertices.
`v_F[i][1]`=F_x, `v_F[i][2]`=F_y, `v_F[i][3]`=F_z. `v_F[][]` are calculated by `forces()` at each time step and reset to 0 after vertices are displaced.

`double v_normal_vector[Nv+1][4]`
Unit vector pointing from a basal vertex towards its apical counterpart.
`v_normal_vector[i][1]`=n_x,`v_normal_vector[i][2]`=n_y,`v_normal_vector[i][3]`=n_z.

`double v_height[Nv+1]`
Distance between basal vertex and its apical counterpart along the normal vector.

`int v_type[Nv+1]`
Type of vertex (=1 is for basal, =2 is for apical).

`int v_partner[Nv+1]`

Id of vertex' counterpart.

`int v_cell1[Nv+1]`
Id of cell 1 containing the vertex.
`int v_cell2[Nv+1]`
Id of cell 2 containing the vertex.

`int v_cell3[Nv+1]`
Id of cell 3 containing the vertex.

`int v_cell4[Nv+1]`
Id of cell 4 containing the vertex.

`int v_reserved[Nv+1]`
=0 if vertex id is free to use or =1 if it is reserved and thus is not allowed to be used.

`int v_T1dir[Nv+1]`
Id of a cell towards which a `valence_reduction()` on the vertex will create a new edge.

`int v_vertT1[Nv+1]`
Id of the vertex that is dissolved in `merge_vertices()` during T1 transformation.

`int v_edgeT1[Nv+1]`
Id of the edge that is dissolved in `merge_vertices()` during T1 transformation.

`double v_clock[Nv+1]`
Stopwatch that measures the time since two vertices were merged and the vertex became 4-way.

3.5 Passive-vertex attributes

`int v_pass_type[Nv_pass+1]`
Type of passive vertex (=1 is for basal, =2 is for apical, =3 is for lateral).

`int v_pass_cell[Nv_pass+1]`
Id of the cell containing the passive vertex.

`int v_pass_edge[Nv_pass+1]`
Id of the basal edge belonging to the lateral side that contains the passive vertex.

3.6 Edge attributes

`int e_v_pass[Ne+1]`
Id of the passive vertex belonging to the lateral side that contains the basal edge.

`int e_lateral1[Ne+1]`
Id of lateral facet 1 above the edge.

`int e_lateral2[Ne+1]`
Id of lateral facet 2 above the edge.

`int e_lateral3[Ne+1]`
Id of lateral facet 3 above the edge.

`int e_lateral4[Ne+1]`
Id of lateral facet 4 above the edge.
`int e_cell1[Ne+1]`
Id of cell 1 containing the edge.

`int e_cell2[Ne+1]`
Id of cell 2 containing the edge.

`int e_reserved[Ne+1]`
=0 if edge id is free to use or =1 if it is reserved and thus is not allowed to be used.

`double e_length[Ne+1]`
Edge length.
Set to edge length when a new edge is created by `make_edge()` and is NOT updated afterwards unless the user updates it by `e_length[i]=edge_length(i)`.

3.7 Facet attributes

`int f_type[Nf+1]`
Type of facet (=1 is for basal, =2 is for apical, =3 is for lateral).

`int f_cell[Nf+1]`
Id of cell containing the facet.

`int f_edge[Nf+1]`
Id of edge under the lateral-type facet.

`double f_T[Nf+1]`
Surface tension at facet.

3.8 Cell attributes

`int c_cent_basal[Nc+1]`
Id of the passive vertex at the center of the basal side of the cell.

`int c_cent_apical[Nc+1]`
Id of the passive vertex at the center of the apical side of the cell.

`double c_V0[Nc+1]`
Preferred cell volume.

`double c_kV[Nc+1]`
Cell reciprocal isothermal compressibility.

3.9 Miscellaneous

`double perioXYZ[3]`
Dimensions of the simulation box. Periodic boundary conditions are prescribed to the boundaries. `perioXYZ[0]` is the dimension in the *x*-direction, `perioXYZ[1]` is the dimension in the *y*-direction, and `perioXYZ[2]` is the dimension in the *z*-direction.

`double h`
Time step for the integration of the equation of motion.
`h` is set in `main()`.

`double Time`
Simulation time.
Runs during the simulation and increases by `h` at every simulation step within `eqOfMotion()`.

`double max_move`
Maximal move of any vertex during the whole simulation.
`h` is set within `eqOfMotion()` after each time step.

`double wA`
The total surface energy.
`wA` is calculated at each time step by `forces()` and reset to 0 after each time step.

`double wV`
The volumetric part of the total energy.
`wV` is calculated at each time step by `forces()` and reset to 0 after each time step.

`double A_tot`
The total surface area of cells.
`A_tot` is calculated at each time step by `forces()` and reset to 0 after each time step.

`int v_free_id`
The lowest vertex id that is currently free to use.

`int e_free_id`
The lowest edge id that is currently free to use.

`int v_pass_free_id`
The lowest passive vertex id that is currently free to use.

`int e_free_id`
The lowest edge id that is currently free to use.

`int f_free_id`
The lowest facet id that is currently free to use.

`int c_free_id`
The lowest cell id that is currently free to use.

`FreeId v_freeId`
Vector of free ids in the list of vertices `v[][]`.

`FreeId v_pass_freeId`
Vector of free ids in the list of passive vertices `v_pass[][]`.

`FreeId e_freeId`
Vector of free ids in the list of edges `e[][]`.

`FreeId f_freeId`
Vector of free ids in the list of facets `f[][]`.

`FreeId c_freeId`
Vector of free ids in the list of cells `basal_edges[][]`.

`size_t array_max`
Size of arrays.

4. HEADERS

functions.h

Includes definitions of all global variables (see above) and links to all header (.h) files (see below).

freeid.h

```
class FreeId
```

```
void FreeId::add(const int id)
```
Adds `id` in the list of free ids.

```
void FreeId::get(int id0)
```
Returns a free id or `id0` if there are no free ids.

allocate.h

```
int read_code_arguments(int argc, char *argv[])
```
Reads arguments of the code and sets the corresponding global variables accordingly.

```
void allocate()
```
Allocates space for global arrays.

```
void deallocate()
```
Deallocates space for global arrays.

```
void reset_arrays()
```
Sets values of all global arrays to 0.

rndom.h

```
double rnd()
```
Returns a random double between 0 and 1.

```
int rnd_int(int intnum)
```
Returns a random integer between 1 and `intnum`.

```
int rnd_H()
```
Returns 1 or -1 (with equal probabilities).

torus.h

`void torus_dx_dy_dz(double *dxdydz, int vert_id, int vert_ref_id)`
Checks if vertices `vert_id` and `vert_ref_id` are on the same side of simulation-box boundary. If yes, vector `dxdydz[]` is set to 0, otherwise it is set to the displacement vector by which vertex `vert_id` needs to be moved so as to be at the same side of simulation-box boundary as vertex `vert_ref_id`.

`void torus_vertex(int i)`
Moves vertex *i* back into the simulation box.

`void torus_vertex_pass(int i)`
Moves passive vertex *i* back into the simulation box.

`void expand_box(double dpx, double dpy, double dpz)`
Expands the simulation box by performing a linear transformation on all vertices:
(*x*,*y*,*z*) --> ((1+`dpx`)*x*,(1+`dpy`)*y*,(1+`dpz`)*z*). perioXYZ[] is corrected as well. `dpx`, `dpy`, and `dpz` can be negative, which corresponds to compression of the simulation box.

distances.h

`double edge_length(int i)`
Returns the mean apico-basal length of edge *i* taking into account periodic boundary conditions.

`void output_edge_lengths(char *fileName)`
Outputs lengths of all basal edges to a file named `fileName`.

`double dist(int i, int j)`
Returns the distance between vertex *i* and vertex *j* (not taking into account periodic boundary conditions).

`double dist_x(int i, int j)`
Returns the distance between vertex *i* and vertex *j* along the *x*-direction (not taking into account periodic boundary conditions).

`double dist_y(int i, int j)`
Returns the distance between vertex *i* and vertex *j* along the *y*-direction (not taking into account periodic boundary conditions).

`double dist_z(int i, int j)`
Returns the distance between vertex *i* and vertex *j* along the *z*-direction (not taking into account periodic boundary conditions).

vert_edg_fac.h

`int make_vertex(double x, double y, double z)`

Creates new vertex by setting `v[][0]` to `v_free_id`, and `v[][1]`, `v[][2]`, and `v[][3]` to `x`, `y`, and `z`, respectively. If vertex id is larger than `Nv`, `Nv` is increased by 1. `v_free_id` is set to the next available vertex id. Id of the new vertex is returned.

`int make_vertex_pass(double x, double y, double z)`
Creates new passive vertex by setting `v_pass[][0]` to `v_pass_free_id`, and `v_pass[][1]`, `v_pass[][2]`, and `v_pass[][3]` to `x`, `y`, and `z`, respectively. If passive vertex id is larger than `Nv_pass`, `Nv_pass` is increased by 1. `v_pass_free_id` is set to the next available passive vertex id. Id of the new passive vertex is returned.

`int make_edge(int v1, int v2)`
Creates new basal edge by setting `e[][0]` to `e_free_id`, and `e[][1]` and `e[][2]` to `v1` and `v2`, respectively. If edge id is larger than `Ne`, `Ne` is increased by 1. `e_free_id` is set to the next available edge id. Id of the new edge is returned.

`int make_facet(int v1, int v2, int v3)`
Creates new facet by setting `f[][0]` to `f_free_id`, and `f[][1]`, `f[][2]` and `f[][3]` to `v1`, `v2` and `v3`, respectively. If facet id is larger than `Nf`, `Nf` is increased by 1. `f_free_id` is set to the next available facet id. Id of the new facet is returned.

basal_network.h

`int make_cell(int poly_class, int e1, int e2, int e3, int e4, int e5, int e6, int e7, int e8, int e9, int e10, int e11, int e12)`
Creates new cell (i.e. polygon within the basal network) by setting `basal_edges[][1]` to `c_free_id`, `basal_edges[][2]` to `poly_class`, and `basal_edges[][3-14]` to oriented edge ids (in the counterclockwise direction; =0 if does not apply). If cell id is larger than `Nc`, `Nc` is increased by 1. `c_free_id` is set to the next available cell id. Id of the new cell is returned.

`void set_v_cell(int vert_id, int i)`
Sets the vertex attribute `v_cell#[vert_id]=i`, where #=1,2,3, or 4, which returns the id of a cell that contains vertex `vert_id`. In this version of the code, a vertex can be at most 4-way, i.e. can be shared by not more than 4 cells.

`void make_basal_vertices(int i)`
For cell `i` it derives the list of basal vertices `basal_vertices[i][]` from `basal_edges[i][]`. `basal_vertices[i][1]` is set to cell id, `basal_vertices[i][2]` is set to polygonal class of cell `i`, and `basal_vertices[i][3-]` are set to ids of vertices (in the counterclockwise direction). For each vertex of cell `i` it also sets `v_cell#[]` by calling `set_v_cell()`.

basal_side.h

`int basal_center(int i, int make_or_not)`
Calculates coordinates of the center point of the basal side of cell `i`. If `make_or_not`=1, a new passive vertex is created at the center of the basal side and `v_pass_cell[]` for the new passive vertex and `c_cent_basal[i]` are set. If `make_or_not`=0, the existing passive vertex of the basal side of cell `i` is moved to the center point. Id of the passive vertex is returned.

`void set_e_cell(int edge_id, int cell_id)`
Sets `e_cell#[edge_id]=cell_id`, where #=1 or 2, which tells the id of a cell containing edge `edge_id`. An edge can be shared by only two cells.

`void make_basal_side(int i)`
Creates basal side of cell `i`.

copy_to_apical.h

`void set_normal_vector(int i)`
Resets `v_normal_vector[i][#]`, for #=1,2,3 and `v_height[i]`.

`int copy_vertex_to_apical(int vert_id)`
Creates the apical counterpart of the basal vertex `vert_id`.

`void copy_vertex_to_apical_ALL()`
Creates the apical counterparts of all basal vertices by calling `copy_vertex_to_apical()` for all basal vertices.

apical_side.h

`int apical_center(int i, int make_or_not)`
Calculates coordinates of the center point of the apical side of cell `i`. If `make_or_not`=1, a new passive vertex is created at the center of the apical side and `v_pass_cell[]` for the new passive vertex and `c_cent_apical[i]` are set. If `make_or_not`=0, the existing passive vertex of the apical side of cell `i` is moved to the center point. Id of the passive vertex is returned.

`void make_apical_side(int i)`
Creates apical side of cell `i`.

`void make_apical_side_ALL()`
Creates basal sides for all cells by calling `make_basal_side()` for all cells.

lateral_side.h

`int lateral_center(int i, int make_or_not)`
Calculates coordinates of the center point of the lateral side defined by basal edge `i`. If `make_or_not`=1, a new passive vertex is created at the center of the lateral side and `v_pass_edge[]` for the new passive vertex and `e_v_pass[i]` are set. If `make_or_not`=0, the existing passive vertex of the apical side of cell `i` is replaced to the center point. Id of the passive vertex is returned.

`void make_lateral_side(int i)`
Creates lateral side defined by basal edge `i`.

`void make_lateral_side_ALL()`
Creates lateral sides for all basal edges by calling `make_lateral_side()` for all basal edges.

output.h

`void out_Vertissue3D(char *fileName)`
Outputs the current tissue structure as a `.vt3d` file.

initial_structure.h

`void set_initial_regHex(size_t arrayMax=1)`
Creates the input structure. First, it creates a regular hexagonal basal network on a flat surface, i.e. vertices (by `make_vertex()`), edges (by `make_edge()`), and cells (by `make_cell()`). It also sets `v_normal_vector[][]` and `v_height[]` for all vertices. Next it creates 3D tissue from the basal network.
`void set_initial_fromFile(char *fileName)`
Creates the input structure. First, it creates the basal network, i.e. vertices (by `make_vertex()`), edges (by `make_edge()`), and cells (by `make_cell()`) and `v_normal_vector[][]` and `v_height[]` for all vertices as as given by the user in the input file `fileName`. Next it creates 3D tissue from the basal network. `perioXYZ[]` are also red from file and set.

force_area.h

`double f_SurfaceTension_force(const int i, int **_f)`
Calculates surface area of triangular facet `i` and contributions to the total force on vertices `f[i][1]` and `f[i][2]`. Total forces `v_F[v1][]` and `v_F[v2][]` are updated and surface area of the facet is returned.

force_volume.h

`double dV(int i, int vert_ref_id, int **_f)`
Calculates and returns volume of tetrahedron defined by triangular facet `i` and (0,0,0).

`double cell_volume(const int i, int **_f)`
Calculates and returns volume of cell `i`.

`void f_VolCompressibility_force(const int i, int cell_id, double sign, int vert_ref_id, double cell_vol, int **_f)`
Calculates contributions to the total force on vertices `f[i][1]` and `f[i][2]`. Total forces `v_F[v1][]` and `v_F[v2][]` are updated.

`double c_VolCompressibility_force(const int cell_id, int **_f)`
Calculates contributions to the total force on all vertices of cell `cell_id` by calling `f_VolCompressibility_force()` for all triangular facets of cell `cell_id`. Energy contribution of cell `cell_id` is returned.

constraints.h

`void constraints_displacement()`
Calculates displacement vector for all vertices such that constraints are satisfied. This is not yet generalized and needs to be modified manually when constraints are changed.

`void constraints_forces()`
Project forces on vertices such that constraints are satisfied after integration step. This is not yet generalized and needs to be modified manually when constraints are changed.

equation_of_motion.h

`void forces(Result _results)`
Calculates forces on all vertices. The first loop goes over all facets and surface-tension contribution is calculated by calling `f_SurfaceTension_force()` for all facets. The second loop goes over all cells and volumetric contribution is calculated by calling `c_VolCompressibility_force()` for all cells. Also surface energy `wA` and volumetric energy `wV` are calculated together with the total area of facets `A_tot`.

`void fix_central_vertices()`
Repositions passive vertices at the centers of cell sides by calling `basal_center()`, `apical_center()`, and `lateral_center()` for all basal, apical, and lateral passive vertices, respectively.

`double eqOfMotion(Result _results)`
First, vertices are projected on constraints. Next, forces on all vertices are calculated by calling `forces()`. This is followed by projection of forces such that constraints are satisfied after the integration step. Next, all vertices are moved according to the equation of motion, `max_move` is updated, and forces on vertices `v_F[][]` are reset to 0. Finally, passive vertices are replaced to the centers of cell sides by calling `fix_central_vertices()`.

dissolve.h

`void dissolve_vertex(int i, int reserved)`
Dissolves vertex `i`. `v[i][]` is set to 0 and so are all vertex attributes, except for `v_reserved[i]`, which is set to `reserved`. `v_free_id` is set to `i` if `i<v_free_id`.

`void dissolve_vertex_pass(int i)`
Dissolves passive vertex `i`. `v_pass[i][]` is set to 0 and so are all passive vertex attributes. `v_pass_free_id` is set to `i` if `i<v_pass_free_id`.

`void dissolve_edge(int i, int reserved)`
Dissolves edge `i`. `e[i][]` is set to 0 and so are all edge attributes, except for `e_reserved[i]`, which is set to `reserved`. `e_free_id` is set to `i` if `i<e_free_id`.

`void dissolve_apical_basal_sides(int i)`

Dissolves all facets and passive vertices of apical and basal sides of cell `i` by calling `dissolve_facet()` and `dissolve_vertex_pass()`.

`void dissolve_cell(int i)`
Dissolves apical and basal sides of cell `i` by calling `dissolve_apical_basal_sides(i)` and sets `basal_vertices[i][]` and cell attributes to 0.

`void delete_cell_from_basal_list(int i)`
Sets `basal_edges[i][]` to 0. `c_free_id` is set to `i` if `i<c_free_id`.

`void dissolve_lateral(int i)`
Dissolves all facets and passive vertex of lateral side `i` by calling `dissolve_facet()` and `dissolve_vertex_pass()`.

list_manipulation.h

`void remove_edge_from_basal_edges(int edg_id, int cell_id)`
Removes edge `edg_id` from `basal_edges[cell_id][]` and reduces `basal_edges[cell_id][2]` by 1.

`int insert_edge_to_basal_edges(int edg_id, int cell_id, int edg_prev, edg_next)`
Inserts edge `edg_id` into `basal_edges[cell_id][]` between edges `edg_prev` and `edg_next` and increases `basal_edges[cell_id][2]` by 1.

T1_transformation.h

`void shrink_edge(int v1, int v2)`
Puts vertex `v1` to the center point between vertices `v1` and `v2`. `v_normal_vector[v1][]` and `v_height[v1]` are also corrected and so is the position of vertex `v_partner[v1]`.

`int merge_vertices(int i)`
It merges vertices of edge `i` into a single vertex and returns its id. `v_vertT1[]`, `v_edgeT1[]`, and `v_T1dir[]` are prescribed to the remaining vertex, whereas the id of the dissolved vertex is reserved.

`int valence_reduction(int v1, int c1, fin_len)`
Valence of a four-way vertex `v1` is reduced by creation of a new edge `v_edgeT1[v1]` in the direction of cell `c1`. After a T1, length of the new edge is set to `fin_len`.

`int T1(int i, double fin_len)`
Performs T1 transformation on edge `i` by calling `merge_vertices()` and `valence_reduction()`. T1 is performed only if both cells that are shared by edge `i` have at least 4 edges (i.e. are at least quadrilaterals) and vertices of edge `i` are both 3-way vertices. After a T1, length of the new edge is set to `fin_len`. If T1 is performed successfully, the function returns 1, otherwise 0 is returned.

cell_growth.h

`void cell_growth(int i, double deltaV)`
Increases `c_V0[i]` by `c_grow[i]*deltaV`. If `c_grow[i]`=-1, cell wants to shrink, if `c_grow[i]`=1, cell wants to grow, and if `c_grow[i]`=0, cell wants to preserve volume.

cell_division.h

`int new_vertex_at_edge_center(int i)`
Creates new vertex at the center of edge `i` and returns its id.

`int divide(int i, int eid1, int eid2)`
Divides cell `i` by creating new edge (e.g. lateral side) connecting centers of edges `eid1` and `eid2`. One of the daughter cells gets the id `i`, whereas the id of the other is returned.

`int cell_division(int i)`
Divides cell `i` by calling `divide()` if its volume is above a threshold volume. `c_V0[]` and `c_grow[]` of both daughter cells are set to 1 and 0, respectively. In the current implementation, the orientation of the cleavage plane is random.

cell_extrusion.h

`void T2(int i)`
Performs T2 transformation on triangular cell `i`.

`int cell_extrusion(int i)`
Extrudes cell `i` by calling `T2()` if the cell has triangular base and its volume is below a threshold value. If cell base has more than 3 sides, forced T1 transformations are performed first, to reduce number of sides to 3.

5. INDEX

GLOBAL VARIABLES

Energy-related parameters

bet
alph
kV
V0

Number of geometric elements
Nv
Nv_pass
Ne
Nf
Nc

Geometric elements
v
v_pass
e
f
basal_edges
basal_vertices
basal_facets
apical_facets

Vertex attributes
v_F
v_normal_vector
v_height
v_type
v_partner
v_cell1
v_cell2
v_cell3
v_cell4
v_reserved
v_T1dir
v_vertT1
v_edgeT1

Passive vertex attributes
v_pass_type
v_pass_cell
v_pass_edge

Edge attributes
e_v_pass
e_lateral1
e_lateral2
e_lateral3
e_lateral4
e_cell1
e_cell2
e_reserved
e_length

Facet attributes
f_type
f_cell
f_edge

f_T

Cell attributes

c_cent_basal
c_cent_apical
c_V0
c_kV
c_grow

Miscellaneous

perioXYZ
h
Time
max_move
wA
wV
A_tot
v_free_id
e_free_id
v_pass_free_id
e_free_id
f_free_id
c_free_id
v_freeId
v_pass_freeId
e_freeId
f_freeId
c_freeId
nIter
array_max
Nx

HEADERS

functions.h
freeid.h
- FreeId
- FreeId::add()
- FreeId::get()

allocate.h
- read_code_arguments()
- allocate()
- deallocate()
- reset_arrays()

rndom.h
- rnd()
- rnd_int()
- rnd_H()

torus.h
- torus_dx_dy_dz()
- torus_vertex()
- torus_vertex_pass()
- expand_box()

distances.h
- edge_length()
- output_edge_lengths()
- dist()

dist_x()
dist_y()
dist_z()

vert_edg_fac.h

make_vertex()
make_vertex_pass()
make_edge()
make_facet()

basal_network.h

make_cell()
set_v_cell()
make_basal_vertices()

basal_side.h

basal_center()
set_e_cell()
make_basal_side()
make_basal_side_ALL()

copy_to_apical.h

copy_vertex_to_apical()
copy_vertex_to_apical_ALL()

apical_side.h

apical_center()
make_apical_side()
make_apical_side_ALL()

lateral_side.h

lateral_center()
make_lateral_side()
make_lateral_side_ALL()

initial_structure.h

set_initial_regHex()
set_initial_fromFile()

force_area.h

f_SurfaceTension_force()

force_volume.h

dV()
cell_volume()
f_VolCompressibility_force()
c_VolCompressibility_force()

constraints.h

void constraints_displacement()
void constraints_forces()

equation_of_motion.h

void forces()
fix_central_vertices()
eqOfMotion()

dissolve.h

dissolve_vertex()
dissolve_vertex_pass()
dissolve_edge()
dissolve_apical_basal_sides()
dissolve_cell()
delete_cell_from_basal_list()
dissolve_lateral()

list_manipulation.h

remove_edge_from_basal_edges()
insert_edge_to_basal_edges()

T1_transformation.h

shrink_edge()
merge_vertices()
valence_reduction()
T1()

cell_growth.h

cell_growth()

cell_division.h

new_vertex_at_edge_center()
divide()
cell_division()

cell_extrusion.h

T2()
cell_extrusion()

output.h

out_WolframM()
dump_structure()
out_edges_torusXY_apical()
out_edges_torusXY_basal()