```
/*

Copyright 1996-2006 Roeland Merks

This file is part of Tissue Simulation Toolkit.

Tissue Simulation Toolkit is free software; you can
redistribute
it and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later
version.

Tissue Simulation Toolkit is distributed in the hope that it
will
be useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public
License
along with Tissue Simulation Toolkit; if not, write to the
Free
Software Foundation, Inc., 51 Franklin St, Fifth Floor,
Boston, MA
02110-1301 USA

*/
#include <stdio.h>
#ifndef __APPLE__
#include <malloc.h>
#endif
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <algorithm>
#include <fstream>
//#include <unistd.h>
#include <math.h>
#include "dish.h"
#include "random.h"
#include "cell.h"
#include "info.h"
#include "parameter.h"
#include "sqr.h"
#ifdef QTGRAPHICS
#include "qtgraph.h"
#else
#include "x11graph.h"
#endif


using namespace std;

INIT {
```

```
  try {

    // Define initial distribution of cells
    CPM->
GrowInCells(par.n_init_cells,par.size_init_cells,par.subfield)
;
    CPM->ConstructInitCells(*this);

    // If we have only one big cell and divide it a few times
    // we start with a nice initial clump of cells.
    //
    // The behavior can be changed in the parameter file.
    for (int i=0;i<par.divisions;i++) {
      CPM->DivideCells();
    }

  } catch(const char* error) {
    cerr << "Caught exception\n";
    std::cerr << error << "\n";
    exit(1);
  }

}

TIMESTEP {

  try {

    static int i=0;

    static Dish *dish=new Dish();
    static Info *info=new Info(*dish, *this);

    // slowly increase target length during the first time
steps
    // to prevent cells from breaking apart
    // static double targetlength=par.target_length;

    // secretion and chemotaxis only starts
    // after relaxation of initial condition
    if (i>=par.relaxation) {
      for (int r=0;r<par.pde_its;r++) {

     dish->PDEfield->Secrete(dish->CPM);
     dish->PDEfield->Diffuse(1);

      }
     dish->MeasureChemConcentrations();
     dish->CPM->GrowAndDivideCells(1);

    }
    dish->CPM->AmoebaeMove(dish->PDEfield);
    double res_compactness; double res_area; double
res_cell_area;
    ofstream myfile;
    myfile.open ("test.txt");
    myfile<<"Circumference: "<<dish->CPM->Circumference()
```

```
<<"\n";
    myfile<<"Compactness: "<<dish->CPM->
Compactness(&res_compactness, &res_area, &res_cell_area)
<<"\n";
    myfile<<"#Cells: "<<dish->CountCells()<<"\n";
    myfile.close();
    if (par.graphics && !(i%par.storage_stride)) {


      int tx,ty;

      BeginScene();


      dish->PDEfield->Plot(this,1);

      // You need to call "ClearImage" if no PDE field is
plotted,
      // because the CPM medium is considered transparant
      //ClearImage();
      dish->Plot(this);


      if (i>=par.relaxation)
      dish->PDEfield->ContourPlot(this,0,7);

      char title[400];
      snprintf(title,399,"CellularPotts: %.2f hr",dish->
PDEfield->TheTime()/3600);
      //snprintf(title,399,"CellularPotts: %d MCS",i);
      //ChangeTitle(title);
      EndScene();
      info->Menu();


    }
    if (par.store && !(i%par.storage_stride)) {
      char fname[200];
      sprintf(fname,"%s/extend%05d.png",par.datadir,i);

      BeginScene();

      dish->PDEfield->Plot(this,1);
      //ClearImage();
      dish->Plot(this);
      if (i>=par.relaxation)
      dish->PDEfield->ContourPlot(this,0,7);

      EndScene();

      Write(fname);
    }


    i++;
  } catch(const char* error) {
    cerr << "Caught exception\n";
```

```
    std::cerr << error << "\n";
    exit(1);
  }
}


void PDE::Secrete(CellularPotts *cpm) {

  const double dt=par.dt;


  double K = 0.1;
  double qs_max = par.secr_rate[1]; //0.25
  double Yxs = 0.2;
  double Yeps = 0.2;
  double a = par.secr_rate[2]; //0.08
  double b = 1/a;
  double Ki = 0.1;

for (int x=0;x<sizex;x++) {
   for (int y=0;y<sizey;y++) {
    double qs = 0;
    double growth = 0;
    double eps_prod = 0;
    sigma[2][x][y] = 0;
    // if borders;
    // secrete chemical at the borders, to the inside

    if (((x<=sizex)&&(y==0))||((x<=sizex)&&(y==sizey-1))) {

    sigma[0][x][y]+=dt*(par.secr_rate[0]);


    } else if (((x==0) &&
(y<=sizey))||((x==sizex-1)&&(y<=sizey))) {

    sigma[0][x][y]+=dt*(par.secr_rate[0]);


    } else {
        if (cpm->Sigma(x,y)) {  //inside the cells
            qs = dt*qs_max*( sigma[0][x][y] / (sigma[0][x][y]
+ K));
            growth = a*Yxs*qs;
            eps_prod = b*Yeps*qs;
              sigma[0][x][y]-= qs;
            sigma[2][x][y]+= growth;
            //sigma[1][x][y]-=dt*sigma[1][x]
[y]*par.secr_rate[1];
            if (cpm->Sigma(x,y)==cpm->Sigma(x+1,y)) {      //
if location on right side of point is does not have the same
index (other cell or medium ) than secrete
                sigma[1][x][y] += eps_prod;
            } else if (cpm->Sigma(x,y)==cpm->Sigma(x-1,y))
{ // if location on left side of point is does not have the
same index (other cell or medium ) than secrete
                //sigma[1][x][y] +=dt*(par.secr_rate[1]);
```

```
                sigma[1][x][y] +=eps_prod;
            } else if (cpm->Sigma(x,y)==cpm->Sigma(x,y+1))
{ // if location on top side of point is does not have the
same index (other cell or medium ) than secrete
                //sigma[1][x][y] +=dt*(par.secr_rate[1]);
                sigma[1][x][y] +=eps_prod;
            } else if (cpm->Sigma(x,y)==cpm->Sigma(x,y-1))
{ // if location on bottom side of point is does not have the
same index (other cell or medium ) than secrete
                //sigma[1][x][y] +=dt*(par.secr_rate[1]);
                sigma[1][x][y] +=eps_prod;
            } else if ((cpm->Sigma(x,y)==cpm->Sigma(x+1,y+
1))||(cpm->Sigma(x,y)==cpm->Sigma(x-1,y-1))) { // if location
on right top or left bottom of point is does not have the same
index (other cell or medium ) than secrete
                //sigma[1][x][y] +=dt*(par.secr_rate[1]);
                sigma[1][x][y] +=eps_prod;
            } else if ((cpm->Sigma(x,y)==cpm->Sigma(x+
1,y-1))||(cpm->Sigma(x,y)==cpm->Sigma(x-1,y+1))) { // if
location on right bot or left top of point is does not have
the same index (other cell or medium ) than secrete
                //sigma[1][x][y] +=dt*(par.secr_rate[1]);
                sigma[1][x][y] +=eps_prod;
            } else {
                sigma[1][x][y] -= dt*sigma[1][x]
[y]*par.decay_rate[1];
                }
          } else {                // outside the cells
            //c_substrate[x][y] = 0;
            sigma[1][x][y] -= dt*sigma[1][x]
[y]*par.decay_rate[2]; // can also be
value*dt*par.decay_rate[1] or
value*sigma[1]*dt*par.decay_rate[1];
            }
           }
        }
    }
  }
int PDE::MapColour(double val) {

  return (((int)((val/((val)+1.))*100))%100)+155;
}

int main(int argc, char *argv[]) {


  try {

#ifdef QTGRAPHICS
    QApplication a(argc, argv);
#endif
    // Read parameters
    par.Read(argv[1]);

    Seed(par.rseed);

    //QMainWindow mainwindow w;
```

```
#ifdef QTGRAPHICS
    QtGraphics g(par.sizex*2,par.sizey*2);
    a.setMainWidget( &g );
    a.connect(&g, SIGNAL(SimulationDone(void)),
SLOT(quit(void)) );

    if (par.graphics)
      g.show();

    a.exec();
#else
    X11Graphics g(par.sizex*2,par.sizey*2);
    int t;

    for (t=0;t<par.mcs;t++) {

      g.TimeStep();

    }
#endif

  } catch(const char* error) {
    std::cerr << error << "\n";
    exit(1);
  }
  catch(...) {
    std::cerr << "An unknown exception was caught\n";
  }
  return 0;
}
```