Supplementary materials

Excluded Range

The excluded range for a pair of two circular chambers can be calculated as follows: Let us denote ω as a direction of line connecting centers of overlapping chambers (see Fig. 15). The excluded range is defined as (ω − α,ω + α). From Figure 15 we have the following relations:

$$\cos\alpha = \frac{a}{r_2} \tag{5a}$$

$$a + b = L \tag{5b}$$

$$b^2 + x^2 = r_1^2 \tag{5c}$$

$$a^2 + x^2 = r_2^2. \tag{5d}$$

Let us eliminate b from equation (5c) by using equation (5b):

$$(L - a)^2 + x^2 = r_1^2 \tag{5e}$$

$$L^2 - 2aL + a^2 + x^2 = r_1^2. \tag{5f}$$

Let us do the same with a^2 by using equation (5d):

$$L^2 - 2aL + r_2^2 - x_2^2 + x^2 = r_1^2 \tag{5g}$$

$$L^2 - 2aL + r_2^2 = r_1^2 \tag{5h}$$

$$a = \frac{r_2^2 - r_1^2 + L^2}{2L}. \tag{5i}$$

Angle α is calculated this way:

$$\cos\alpha = \frac{a}{r_2} = \frac{r_2^2 - r_1^2 + L^2}{2r_2L} \tag{5j}$$

$$\alpha = \arccos\left(\frac{r_2^2 - r_1^2 + L^2}{2r_2L}\right). \tag{5k}$$

The Program Code

In this section we present a core code of the program for generating foram shells. The program has been implemented in C++, one of the most popular object-oriented languages. Compared with other well-known programming languages such as C and Pascal, which are called structural languages, C++ provides higher abstraction level allowing to define new types of data, more closely related to real objects. Thus, we have defined types such as **`Vector`**, **`Cell`**, and **`Foraminifera`**, corresponding to fundamental terms of our model.

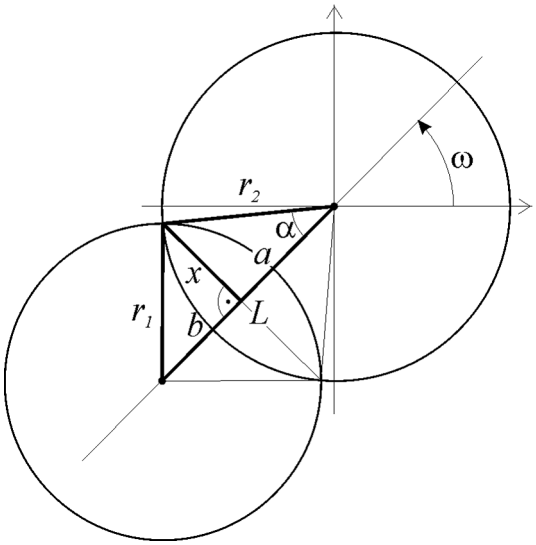

FIGURE 15. Excluded range for the pair of chambers.

We can also define basic functionality in a form of subroutines. **`Vector`** data type, widely used in this code, provides rich functionality for manipulating 2-D vectors as translations, rotations and scaling. In this way we can also define new vectors in various ways, e.g., by using polar or euclidean coordinates. The **`addNewCell`** subroutine implements the main model algorithm as it is described in the section ''Algorithm.'' We do not present the full code of the program because of space limitation; however, detailed comments should supply the necessary information. We do not include definitions of the subroutines used in this code, but their internal functionality is obvious or can be easily deducted from the model description. The **`Cell`** type represents a single circular chamber and it stores coordinates of a center, a radius of the chamber, and an aperture. The **`Foraminifera`** type is a simple container for **`Cell`** types. The series of **`Cell`** objects is stored as a one-directional chain: variable **`lastCell`** indicates the previously added chamber.

```
//add new cell in each iteration

int Foraminifera::addNewCell()
{
  float newCenterX, newCenterY;

  //r—indicates aperture location
  //r.radius—size of new chamber
  Vector r;

  //size of new cell
  //''scale'' function may implement
  //any method of (isometric) scalling
  r.radius = scale(lastCell->radius);

  //v—growth vector
  //the same direction as reference growth axis
  Vector v(lastCell->centerX, lastCell->centerY,
    lastCell->aperturX, lastCell->aperturY);

  //scale by TF factor
  scaleGrowthVector(v, r.radius);

  //modifies growth vector by deviating
  //it from the reference growth axis
  deviateGrowthVector(v);

  //center of new cell
  newCenterX = lastCell->apertureX + v.x;
  newCenterY = lastCell->apertureY + v.y;

  //new aperture location
  //two candidates:
  float tmpGamma1 = minimize(v,r.radius)
  float tmpGamma2 = tmpGamma1 + PI;

  Vector tmpR1(r.radius, tmpGamma1);
  Vector tmpR2(r.radius, tmpGamma2);

  Vector u1 = v + tmpR1;
  Vector u2 = v + tmpR2;
  if(u1.length() < u2.length)
   r = tmpR1;
  else
   r = tmpR2;

//What if new aperture lies inside any other cell
  //calculate excluded range
  Cell* itCell = firstCell;
  float cLimitL =−10000.0f, cLimitR = 10000.0f;

  //iterate through all the previous chambers
```

```
for(; itCell ; itCell = itCell->nextCell)
{
 //count distance between centers
 Vector L(newCenterX, newCenterY,
   itCell->centerX, itCell->centerY);

 float distance = L.length();
 if(distance > (itCell->radius + r.radius))
   continue; //this cell does not overlap new cell
 //else
 //count range of overlapping
 float alpha = acos((r.radius * r.radius-
    itCell->radiusR * itCell->radiusR +
    distance*distance) /
    (2 * distance * r.radius);

 float limitL, limitR;
 limitL = L.getPhi()-alpha;
 limitR = L.getPhi() + alpha;

 //do the corrections (0 =< angle =< 2PI)
 if(limitL < 0.0f)
   limitL = 2 * PI + limitL;
 if(limitR > 2 * PI)
   limitR = limitR- 2 * PI;

 //update global excluded range
 if(cLimitL > limitL)
   cLimitL = limitL;
 if(cLimitR < limitR)
   cLimitR = limitR;
}

//if new foramina lies between cLimitL and cLimitR
if(r.gamma > cLimitL && r.gamma < cLimitR)
{
  Vector candidate_1(r.radius, cLimitL);
  Vector candidate_2(r.radius, cLimitR);

  Vector r1 = v + candidate_1;
  Vector r2 = v + candidate_2;
  if(r1.length() < r2.length())
   r = r1;
  else
   r = r2;
 }
 lastCell->nextCell = new Cell(newCenterX,
    newCenterY, r.radius,
    newCenterX+r.x, newCenterY+r.y);

 lastCell = lastCell->nextCell;
 return 0;
 }
```