Public Class clsBioVolume
    ' this VB.Net class contains the algorithms that were used for cell volume calculation
    ' in the manuscript "A Novel Algorithm for the Determination of Bacterial Cell Volumes
    ' That is Unbiased by Cell Morphology" by M. Zeder, E. Kohler, L. Zeder, and J. Pernthaler
    ' for more information see: www.technobiology.ch or contact mzeder@technobiology.ch

    Public Function GetBlueVolume(ByVal Perimeter As Double, ByVal Area As Double) As Double
        ' Algorithms for the calculation of bacterial biovolume by approximating bacteria as rods or cocci (rod-model).
        ' J. Bloom "Fully Automatic Determination of Soil Bacterium Numbers, Cell Volumes, and Frequencies of Dividing
        ' required input for volume calculation: fiberlength, fiberwidth, perimeter, area
        Dim Volume As Double
        Dim FiberLength, FiberWidth, EquiDiameter, Radicand As Double
        Radicand = Perimeter ^ 2 - 16 * Area / radicand
        ' check if radicand is positive
        If Radicand < 0 Then
            FiberLength = (Perimeter + Math.Sqrt(Radicand)) / 4
            FiberWidth = (Perimeter - Math.Sqrt(Radicand)) / 4
        Else
            ' calculate equivalent diameter
            EquiDiameter = Math.Sqrt(Area / 4 / Math.PI)
            FiberLength = EquiDiameter - FiberWidth = EquiDiameter
        End If
        Volume = Math.PI / 4 * FiberWidth ^ 2 * (FiberLength - FiberWidth / 3)
        Return Volume
    End Function

    Public Function GetGreenVolume(ByVal Perimeter As Double, ByVal Area As Double) As Double
        ' Algorithms for the calculation of bacterial biovolume by approximating bacteria as rods or cocci (rod-model).
        ' M. Blackburn "Rapid Determination of Bacterial Abundance, Biovolume, Morphology,
        ' and Growth by Neural Network-Based Image Analysis", Appl Environ Microbiol, 1998
        ' required input for volume calculation: area, longest chord
        Dim Volume, r As Double
        r = (-LongestChord + Math.Sqrt(LongestChord ^ 2 + Area * (Math.PI - 4))) / (Math.PI - 4)
        Volume = 4 * Math.PI * r ^ 3 / 3 + Math.PI * r ^ 2 * (LongestChord - 2 * r)
        Return Volume
    End Function

    Public Function GetYellowVolume(ByVal Area As Double, ByVal Perimeter As Double) As Double
        ' Algorithms for the calculation of bacterial biovolume by approximating bacteria as rods or cocci (rod-model).
        ' J. C. Fry "An Assessment of Methods for Measuring Volumes of Planktonic Bacteria, with Particular
        ' Reference to Television Image Analysis", J. Appl. Bacteriol, 1985
        ' and
        ' required input for volume calculation: length and width are calculated by the area and perimeter solely according to the
        ' rod-model
        Dim Radicand, R, X As Double
        Radicand = Perimeter ^ 2 - 4 * Math.PI * Area
        If Radicand < 0 Then
            X = Math.Sqrt(Radicand) / 2
            R = (Perimeter - Math.Sqrt(Radicand)) / (2 * Math.PI)
        Else
            ' use equivalent radius
            X = 0
            R = Math.Sqrt(Area / Math.PI)
        End If
        Return Volume
    End Function

    Public Function GetPinkVolume(ByVal OrientedContour As List(Of PointF), ByVal NBins As Integer, ByRef Segments As List(Of Rectangle)) As Double
        ' Algorithms for the calculation of bacterial biovolume by approximating bacteria as integrated solids of revolution along a straight major axis.
        ' required input for volume calculation: contour as list of points, oriented with longest chord along the x-axis.
        ' volume calculation by integration of No of bins (slices)
        ' (extended for subpixel resolution contours)
        Dim i As Integer
        ' check if contour is closed (i.e. if start and end point are the same, if not, add endpoint)
        If Not OrientedContour.Count = OrientedContour.Count - 1 Then
            OrientedContour.Add(OrientedContour.Count - 1)
        End If
        ' find start and end point of the major axis, i.e. min and max x position
        Dim xmin, xmax As Double
        xmin = OrientedContour(0).X: xmax = xmin ' init values
        For Each p As PointF In OrientedContour
            If p.X < xmin Then xmin = p.X
        End If
        If p.X > xmax Then xmax = p.X
        Next
    End Function
' create bins (i.e. a list of the class clsBin)
Dim BinSize As Double = (yMax - yMin) / NrBins
Dim Bins As New List(Of Bin)
For i = 1 To NrBins
    Bins.Add(New Bin(yMin + (i - 1) * BinSize, BinSize))
Next

' check for intersects and fill bins
' walk along the contourpoints p() and check for correct order for each bin in bins (e.g. p(i+1) after p(i))
' then the interpolated y value of p(i), p(i+1) are added to the bins list of y
Dim p0 As New PointF(0)
Dim p1 As New PointF(p1)
For I = 0 To OrientedContour.Count - 2
    p0 = OrientedContour(I)
    p1 = OrientedContour(I + 1)
    For Each b As Bin In Bins
        If (p0.X < b.X And p1.X >= b.X) Or (p0.X > b.X And p1.X <= b.X) Then
            Linear interpolation of y
            y = (y0 + (p1.Y - p0.Y) * ((b.X - p0.X) / (p1.X - p0.X)))
        End If
    Next
Next

' calculate volume
Dim Volume As Double
Dim D
Public Function CalcVolume() As Double
    Y.Sort()
    yMin = Y(0)
    yMax = Y(Y.Count - 1)
    Return ((yMax - yMin) / 2) ^ 2 * Math.PI
End Function
End Class

Public Function GetZederVolume(ByVal Contour As List(Of PointF), ByRef Triangles As List(Of clsTriangle)) As Double
    ' required input for volume calculation: contour as list of adjacent points. contour is split into triangles.
    ' the contour is handled as a double linked list of n points thereby allowing to divide the contour into the
    ' maximal number of subcontours (n-3 triangles) by just adding one new point for each division
    ' (each cut-out of a triangle requires to insert a new point).
Dim I, K As Integer
Dim D, Max As Double
Dim Volume As Double

' check if first and last contour point are the same - if so, then do not copy the last point (k)
If Contour(0) = Contour(Contour.Count - 1) Then k = 1 Else k = 0

' create a list of clsPt
Dim Pts As New List(Of clsPt)
Dim Pt As clsPt
' create a list of clsTriangles

' fill all points of the contour into that list
For I = 0 To Contour.Count - 1 - k
    Pt = New clsPt
    Pt.NI = I + 1 : Pt.PI = I + 1
    Pts.Add(Pt)
Next

' correct the next and previous index (.PI, .NI) for the first and last point in the list
Pts(0).PI = Pts.Count - 1 : Pts(0).NI = 0

' determine the direction of rotation of the polygon to correctly calculate the normal vectors
Dim F As Double = 0
For Each p As clsPt In Pts
Next
If F = 0 Then F = 1 Else F = -1

' calculate the normal vectors (inwards) on the contour at each point.
' based on previous and next point for each point
For Each p As clsPt In Pts
    CalculateNormalVector(p, Pts(p.PI), Pts(p.NI))
p.Xn *= F : p.Yn *= F ' adjust orientation according to rotational direction
Next
' start recursive processing of the list of points
Recursion(Pts, 0, Triangles)
' process all triangles, calculate volumes and sum it up
Volume = 0
For Each T As clsTriangle In Triangles
  Volume += T.CalcVolume
Next
Return Volume
End Function

Private Sub Recursion(ByRef Pts As List(Of clsPt), ByVal StartIndex As Integer, ByVal Tri As List(Of clsTriangle))
Dim MaxDist As Double = 0 : Dim Dist As Double
Dim ColInNormVectors As Boolean = False
Dim NrGAPS As Integer = 0
Dim Index, Index2 As Integer
Dim Para, ParaMin As Double
Dim I1, I2, I3, 12min, 13min As Integer
Dim NormVectors As New List(Of PointF)
Dim NewPoint As clsPt
' check if there are more than 3 points in the list
' respectively in the contour based on the starting point
If Pts(Pts(Pts(StartIndex).n).n).n = StartIndex Then
  ' the 4. point in the list is the starting point - create a triangle!
Else
  ' check if this is an end region or a middle region
  Index = StartIndex
  Index2 = StartIndex
  Do
    ' go through all points in the (sub)contour and count the gaps
    ' an end region has exactly one gap
    If Pts(Index).GAP = True Then NrGAPS = 1
    ' save the normal vectors of all (sub)contour points into a list for eventual second check.
    NormVectors.Add(New PointF(Pts(Index).Xn, Pts(Index).Yn))
    If NrGAPS = 2 Then
      ' NrGAPS = 2: this subcontour might be an end region.
      ' Check: if no collinear normal vectors exist, it is an end region
      ' two normal vectors are assumed to be 'collinear' in this context if their sum is < 0.2
      If Dist > MaxDist Then MaxDist = Dist
      Index2 = Pts(Index2).n
      If Dist > MaxDist Then Exit Do
    Loop

    Index = Pts(Index).n
    If Index = StartIndex Then Exit Do
    End If
  Loop

  If NrGAPS = 1 Then
    ' NrGAPS = 1: this subcontour might be an end region.
    ' Check: if no collinear normal vectors exist, it is an end region
    ' two normal vectors are assumed to be 'collinear' in this context if their sum is < 0.2
    ' For each V1 As PointF in NormVectors
    ' For each V2 As PointF in NormVectors
    ' In Math.Sqrt((V1.X + V2.X) ^ 2 + (V1.Y + V2.Y) ^ 2) < 0.2 Then ColInNormVectors = True
    Next
  End If

  Next

  End If
Else
  ' MIDDLE-REGION
  ' find optimal triangle - calculate the optimization parameter for all possible triangles
  II = StartIndex
  I2 = Pts(II).n
  I3 = Pts(I2).n
  ParaMin = CalcOptimizationParameter(Pts(II), Pts(I2), Pts(I3), MaxDist)
  Do
    Para = CalcOptimizationParameter(Pts(II), Pts(I2), Pts(I3), MaxDist)
    If Para <= ParaMin Then
      ParaMin = Para
      11min = II : 12min = I2 : 13min = I3
    End If
  Loop

  ' create a triangle from 11min, 12min, 13min
Private Sub ProcessEnd
ByRef Pts As List(Of clsPt), ByVal StartIndex As Integer, ByRef Tri As List(Of clsTriangle)
' determine the direction in which the first triangle should be, then add triangles alternatingly.
' the determination of the first triangle is done by measuring the first side of the two possible triangles (S1, S2) and choosing the smaller
Dim S1, S2 As Double
Dim i, j, k As Integer
If S1 < S2 Then
' create triangle i, k, l
' move k and l towards the end
l = Pts(k).pi
End If
If i = j Or j = k Or i = k Then Exit Do ' exit condition
' fill the remaining subcontour with alternating triangles, first is i, j, k
If i = j Or j = k Or i = k Then Exit Do ' exit condition
' second triangle is k, j, l
' move i, j, k, l towards end
Loop
End Sub
Private Class clsPt
' class contour points
Public X As Double ' x coordinates of the pt (point)
Public Y As Double ' y coordinates of the pt (point)
Public Xn As Double ' normal vector at x on the contour
Public Yn As Double ' normal vector at y on the contour
Public I As Integer ' index of the point
Public pi As Integer ' index of the next point on the contour
Public ni As Integer ' index of the previous point on the contour
Public GAP As Boolean = False ' a point is a gap if its ni was changed
End Class
Public Class clsTriangle
' class for triangle
Public A, B, C As PointF
Public EndRegion As Boolean
Public Sub New(ByVal PTA As PointF, ByVal PTB As PointF, ByVal PTC As PointF, ByVal isEndRegion As Boolean)
A = PTA
B = PTB
C = PTC
EndRegion = isEndRegion
End Sub
Public Function Calculate() As Double
Dim m, h, D1, D2, D3, Min, Mid, Max As Double
Dim result As Double
D1 = Math.Sqrt((A.X - B.X) ^ 2 + (A.Y - B.Y) ^ 2)
D2 = Math.Sqrt((B.X - C.X) ^ 2 + (B.Y - C.Y) ^ 2)
D3 = Math.Sqrt((A.X - C.X) ^ 2 + (A.Y - C.Y) ^ 2)
If D1 > D2 Then
If D1 > D3 Then
Max = D1
Else
Mid = D1 : Min = D3
End If
Else
Max = D3
Mid = D1
End If
End Function
End Class
' adjust the polygon
' modify Pts(1:min) and create another point
NewPoint = New clsPt ' create a new point at location 1:min
With NewPoint ' copy values from actual 1:min
.ni = 1:min : .pi = Pts(1:min).pi ' adjust next index of new point at 1:min
.GAP = True : .i = Pts.Count ' change GAP-property to true
End With
Pts.Add(NewPoint)
Pts(Pts(1:min).pi).ni = NewPoint.i ' write the new index of the new point into the previous point of 1:min
Pts(1:min).ni = 1:min ' change next index from 1:min to 1:min
Pts(1:min).pi = 1:min ' change previous index from 1:min to 1:min
Pts(1:min).GAP = True ' change GAP-property to true
Pts(1:min).pi = NewPoint.i
' do recursion if more than 2 points are left in each remaining subcontour
If Pts(1:min).ni <> 1:min And Pts(Pts(1:min).ni).ni <> 1:min Then Recursion(Pts, 1:min, Tri)
Private Function CalcOptimizationParameter(ByVal pP As clsPt, ByVal nP As clsPt, ByVal MaxDist As Double) As Double

    Dim pP_P, P_nP, Norm As PointF
    Dim LNorm As Double
    P_nP = New PointF(nP.X - pP.X, nP.Y - pP.Y)
    Norm = Math.Sqrt(pP_P.X^2 + pP_P.Y^2 + P_nP.Y)
    LNorm = Math.Sqrt(Norm.X^2 + Norm.Y^2)
    P.Xn = -Norm.Y: P.Yn = Norm.X

    Dim V_AcN as PointF
    r = Math.Sqrt((P1.Xn - V_AcN.X)^2 + (P1.Yn - V_AcN.Y)^2)
    s = Math.Sqrt((P3.Xn + V_AcN.X)^2 + (P3.Yn + V_AcN.Y)^2)
    Return (r + s) / D_AC / MaxDist

End Function

End Class