```vb

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 inforation see: www.technobiology.ch or contact mzeder@technobiology.ch

    Public Function GetBloemVolume(ByVal Perimeter As Double, ByVal Area As Double) As Double
        ' Algorithm for the calculation of bacterial biovolume by approximating bacteria as rods or cocci (rod-model).
        ' J. Bloem "Fully Automatic Determination of Soil Bacterium Numbers, Cell Volumes, and Frequencies of Dividing
        ' Cells by Confocal Laser Scanning Microscopy and Image Analysis", Appl Environ Microbiol, 1995
        ' required input for volume calculation: fiberlength, fiberwidth, perimeter, area
        Dim Volume As Double
        Dim FiberLength, FiberWidth, EquDiameter, 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
            EquDiameter = Math.Sqrt(Area * 4 / Math.PI)
            FiberLength = EquDiameter : FiberWidth = EquDiameter
        End If
        Volume = Math.PI / 4 * FiberWidth ^ 2 * (FiberLength - FiberWidth / 3)
        Return Volume
    End Function

    Public Function GetBlackburnVolume(ByVal LongestChord As Double, ByVal Area As Double) As Double
        ' Algorithm for the calculation of bacterial biovolume by approximating bacteria as rods or cocci (rod-model).
        ' N. 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 GetFryVolume(ByVal Area As Double, ByVal Perimeter As Double) As Double
        ' Algorithm 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
        '
        ' J. C. Fry "Direct Methods and Biomass Estimation", Meth. Microbiol, 1990
        ' required input for volume calculation: length and width are calculated by the area and perimeter solely according to the rod-model

        ' instead of using length l and width d we are using the radius R and lenght X (= length of cylinder in the rod-model) whereas
        ' R = d / 2 --> see eq. (14) in Fry 1990
        ' X = l - d --> see eq. (15) in Fry 1990
        ' The volume is calculated according to eq. (11) in Fry 1990

        Dim Radicand, R, X, Volume 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
        Volume = 4 / 3 * R ^ 3 * Math.PI + R ^ 2 * Math.PI * X
        Return Volume
    End Function

    Public Function GetSierackiVolume(ByVal OrientedContour As List(Of PointF), ByVal NrBins As Integer, ByRef Segments As List(Of RectangleF)) As Double
        ' Algorithm for the calculation of bacterial biovolume by  approximating bacteria as integrated solids of revolution along a straight major axis
        ' M. Sieracki "Algorithm to Estimate Cell Biovolume Using Image Analyzed Microscopy", Cytometry, 1989
        ' 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(0) = OrientedContour(OrientedContour.Count - 1) Then
            OrientedContour.Add(OrientedContour(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
            If p.X > xMax Then xMax = p.X
        Next

```

```
        ' create bins (i.e. a list of the class clsBin)
        Dim BinSize As Double = (xMax - xMin) / NrBins
        Dim Bins As New List(Of Bin)
        For i = 1 To NrBins
            Bins.Add(New Bin(xMin + (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 ' p(i)
        Dim p1 As New PointF ' p(i+1)
        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
                    b.Y.Add(p0.Y + (p1.Y - p0.Y) * ((b.X - p0.X) / (p1.X - p0.X)))
                End If
            Next
        Next

        ' calculate volume
        Dim Volume As Double = 0
        For Each b As Bin In Bins
            ' sum up the volumes of each bin
            Volume += b.CalcVolume
            ' save the rectangle for later graphical depiction
            Segments.Add(New RectangleF(b.X - BinSize / 2, b.yMin, BinSize, b.yMax - b.yMin))
        Next

        Return Volume
    End Function
    Private Class Bin
        Public Y As New List(Of Double)
        Public yMax, yMin As Double
        Public X As Double
        Dim Size As Double
        Public Sub New(ByVal Xpos As Double, ByVal BinSize As Double)
            Size = BinSize
            X = Xpos
        End Sub
        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
        ' Algorithm for the calculation of bacterial biovolume by approximating bacteria as solids of revolution along a non-straight↙
     major axis.
        ' required input for volume calculation: contour as list of adjacent points. contour is split into triangles.
        ' the contour is handeled 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, Dmax 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.I = i : Pt.X = Contour(i).X : Pt.Y = Contour(i).Y
            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(Pts.Count - 1).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
            F += p.X + Pts(p.nI).Y - p.Y * Pts(p.nI).X
        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
            CalculateNormVector(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, ByRef 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, I1min, I2min, I3min 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).nI).nI).nI = StartIndex Then
            ' the 4. point in the list is the starting point - create a triangle!
            Tri.Add(New clsTriangle(New PointF(Pts(StartIndex).X, Pts(StartIndex).Y), New PointF(Pts(Pts(StartIndex).nI).X, Pts(Pts(StartIndex).nI).Y), New PointF(Pts(Pts(Pts(StartIndex).nI).nI).X, Pts(Pts(Pts(StartIndex).nI).nI).Y), False))

        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))

                Do
                    ' calculate all distances within the existing subcontour and find the longest one
                    Dist = Math.Sqrt((Pts(Index).X - Pts(Index2).X) ^ 2 + (Pts(Index).Y - Pts(Index2).Y) ^ 2)
                    If Dist > MaxDist Then MaxDist = Dist
                    Index2 = Pts(Index2).nI
                    If Index2 = StartIndex Then Exit Do
                Loop

                Index = Pts(Index).nI
                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
                        If Math.Sqrt((V1.X + V2.X) ^ 2 + (V1.Y + V2.Y) ^ 2) < 0.2 Then CoLinNormVectors = True
                    Next
                Next
            End If
            If NrGAPS = 1 And CoLinNormVectors = False Then
                ' if exactly one gap exists and no collinear normal vectors are found, then this is an end region.
                ' END-REGION
                ProcessEnd(Pts, StartIndex, Tri)

            Else
                ' MIDDLE-REGION
                ' find optimal triangle - calculate the optimization parameter for all possible triangles
                I1 = StartIndex
                I2 = Pts(I1).nI
                I3 = Pts(I2).nI
                ParaMin = CalcOptimizationParameter(Pts(I1), Pts(I2), Pts(I3), MaxDist)
                Do
                    Para = CalcOptimizationParameter(Pts(I1), Pts(I2), Pts(I3), MaxDist)
                    If Para <= ParaMin Then
                        ParaMin = Para
                        I1min = I1 : I2min = I2 : I3min = I3
                    End If
                    I3 = Pts(I3).nI
                    If I3 = I1 Then
                        I1 = I2
                        I2 = Pts(I2).nI
                        I3 = Pts(I2).nI
                        If I2 = StartIndex Then Exit Do
                    End If
                Loop
                ' create a triangle from I1min, I2min, I3min
                Tri.Add(New clsTriangle(New PointF(Pts(I1min).X, Pts(I1min).Y), New PointF(Pts(I2min).X, Pts(I2min).Y), New PointF(Pts(I3min).X, Pts(I3min).Y), False))

```

```
                ' adjust the polygon
                ' modify Pts(I1min) and create another point

                NewPoint = New clsPt ' create a new point at location I3min
                With NewPoint ' copy values from actual I3min
                    .X = Pts(I3min).X : .Y = Pts(I3min).Y
                    .Xn = Pts(I3min).Xn : .Yn = Pts(I3min).Yn
                    .nI = I2min : .pI = Pts(I3min).pI ' adjust next index of new point at I3min
                    .GAP = True : .I = Pts.Count ' change GAP-property to true
                End With
                Pts.Add(NewPoint)
                Pts(Pts(I3min).pI).nI = NewPoint.I  ' write the new index of the new point into the previous point of I3min
                Pts(I1min).nI = I3min ' change next index from I1min to I3min
                Pts(I3min).pI = I1min  ' change previous index from I3min to I1min
                Pts(I1min).GAP = True ' change GAP-property to true
                Pts(I2min).pI = NewPoint.I

                ' do recursion if more than 2 points are left in each remaining subcontour
                If Pts(I1min).nI <> I1min And Pts(Pts(I1min).nI).nI <> I1min Then Recursion(Pts, I3min, Tri)
                If Pts(I2min).nI <> I2min And Pts(Pts(I2min).nI).nI <> I2min Then Recursion(Pts, I2min, Tri)
            End If
        End If
    End Sub
    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, l As Integer
        i = StartIndex : j = Pts(i).nI : k = Pts(i).pI : l = Pts(k).pI
        S1 = Math.Sqrt((Pts(i).X - Pts(l).X) ^ 2 + (Pts(i).Y - Pts(l).Y) ^ 2)
        S2 = Math.Sqrt((Pts(k).X - Pts(j).X) ^ 2 + (Pts(k).Y - Pts(j).Y) ^ 2)
        If S1 < S2 Then
            ' create triangle i, k, l
            Tri.Add(New clsTriangle(New PointF(Pts(i).X, Pts(i).Y), New PointF(Pts(k).X, Pts(k).Y), New PointF(Pts(l).X, Pts(l).Y), True))
            k = l ' move k and l towards the end
            l = Pts(k).pI
        End If
        Do
            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
            Tri.Add(New clsTriangle(New PointF(Pts(i).X, Pts(i).Y), New PointF(Pts(j).X, Pts(j).Y), New PointF(Pts(k).X, Pts(k).Y), True))
            If l = j Or j = k Or l = k Then Exit Do ' exit condition
            ' second triangle is k, j, l
            Tri.Add(New clsTriangle(New PointF(Pts(k).X, Pts(k).Y), New PointF(Pts(j).X, Pts(j).Y), New PointF(Pts(l).X, Pts(l).Y), True))
            ' move i, j, k, l towards end
            i = Pts(i).nI : j = Pts(j).nI : k = Pts(k).pI : l = Pts(l).pI
        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 CalcVolume() 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
                    If D2 > D3 Then
                        Mid = D2 : Min = D3
                    Else
                        Mid = D3 : Min = D2
                    End If
                Else
                    Max = D3
                    Mid = D1
```

```
                    Min = D2
                End If
            Else
                If D1 < D3 Then
                    Min = D1
                    If D2 < D3 Then
                        Mid = D2
                        Max = D3
                    Else
                        Mid = D3
                        Max = D2
                    End If
                Else
                    Mid = D1
                    Min = D3
                    Max = D2
                End If
            End If
            If EndRegion Then
                m = (Max * Max - Mid * Mid + Min * Min) / (2 * Max)
                h = Math.Sqrt(Min ^ 2 - m ^ 2)
                result = ((Max - m) / 2) ^ 2 * Math.PI * h / 2
            Else
                result = (Mid / 2) ^ 2 * Math.PI * Min / 2
            End If
            Return result
        End Function
    End Class

    Private Sub CalculateNormVector(ByRef P As clsPt, ByVal pP As clsPt, ByVal nP As clsPt)
        ' define vectors
        Dim pP_P, P_nP, Norm As PointF
        Dim LNorm As Double
        pP_P = New PointF(P.X - pP.X, P.Y - pP.Y)
        P_nP = New PointF(nP.X - P.X, nP.Y - P.Y)
        Norm = New PointF(pP_P.X + P_nP.X, pP_P.Y + P_nP.Y)
        LNorm = Math.Sqrt(Norm.X ^ 2 + Norm.Y ^ 2)
        Norm.X = Norm.X / LNorm : Norm.Y = Norm.Y / LNorm
        P.Xn = -Norm.Y : P.Yn = Norm.X
    End Sub

    Private Function CalcOptimizationParameter(ByVal P1 As clsPt, ByVal P2 As clsPt, ByVal P3 As clsPt, ByVal MaxDist As Double) As ↙
    Double
        Dim D_AC, r, s As Double
        Dim V_ACn As PointF
        D_AC = Math.Sqrt((P1.X - P3.X) ^ 2 + (P1.Y - P3.Y) ^ 2)
        V_ACn = New PointF((P3.X - P1.X) / D_AC, (P3.Y - P1.Y) / D_AC)
        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
```