Tutorial 60: Facet generation for arbitrary contours in 3D space

2020-09-14: Tim C. Lueth, Professor at Technische Universität München, Germany (URL: http://www.SG-Lib.org) - Last Change: 2021-02-25

Contents

Complete List of all Tutorials with Publishable MATLAB Files of this Solid-Geoemtries Toolbox

The following topics are covered an explained in the specific tutorials:

- Tutorial 01: First Steps Using the VLFL-Toolbox for Solid Object Design
- Tutorial 02: Using the VLFL-Toolbox for STL-File Export and Import
- Tutorial 03: Closed 2D Contours and Boolean Operations in 2D
- Tutorial 04: 2½D Design Using Boolean Operators on Closed Polygon Lists (CPL)
- Tutorial 05: Creation, Relative Positioning and Merging of Solid Geometries (SG)
- Tutorial 06: Relative Positioning and Alignment of Solid Geometries (SG)
- Tutorial 07: Rotation of Closed Polygon Lists for Solid Geometry Design
- Tutorial 08: Slicing, Closing, Cutting and Separation of Solid Geometries
- Tutorial 09: Boolean Operations with Solid Geometries
- Tutorial 10: Packaging of Sets of Solid Geometries (SG)
- Tutorial 11: Attaching Coordinates Frames to Create Kinematik Models
- Tutorial 12: Define Robot Kinematics and Detect Collisions
- Tutorial 13: Mounting Faces and Conversion of Blocks into Leightweight-structures
- Tutorial 14: Manipulation Functions for Closed Polygons and Laser Cutting (SVG)
- Tutorial 15: Create a Solid by 2 Closed Polygons
- Tutorial 16: Create Tube-Style Solids by Succeeding Polygons
- Tutorial 17: Filling and Bending of Polygons and Solids
- Tutorial 18: Analyzing and modifying STL files from CSG modeler (Catia)
- Tutorial 19: Creating drawing templates and dimensioning from polygon lines
- Tutorial 20: Programmatically Interface to SimMechanics Multi-Body Toolbox
- Tutorial 21: Programmatically Convert Joints into Drives (SimMechanics)
- Tutorial 22: Adding Simulink Signals to Record Frame Movements
- Tutorial 23: Automatic Creation of a Missing Link and 3D Print of a Complete Model
- Tutorial 24: Automatic Creation of a Joint Limitations
- Tutorial 25: Automatic Creation of Video Titels, Endtitels and Textpages
- Tutorial 26: Create Mechanisms using Universal Planar Links
- Tutorial 27: Fourbar-Linkage: 2 Pose Syntheses and Linkage Export for 3D Printing
- Tutorial 28: Fourbar-Linkage: 3 Pose Syntheses and Linkage Export for 3D Printing
- Tutorial 29: Create a multi body simulation using several mass points
- Tutorial 30: Creating graphical drawings using point, lines, surfaces, frames etc.
- Tutorial 31: Importing 3D Medical DICOM Image Data and converting into 3D Solids
- Tutorial 32: Exchanging Data with a FileMaker Database
- Tutorial 33: Using a Round-Robin realtime multi-tasking system
- Tutorial 34: 2D Projection Images and Camera Coordinate System Reconstruction
- Tutorial 35: Creation of Kinematic Chains and Robot Structures
- Tutorial 36: Creating a Patient-Individual Arm-Skin Protector-Shell

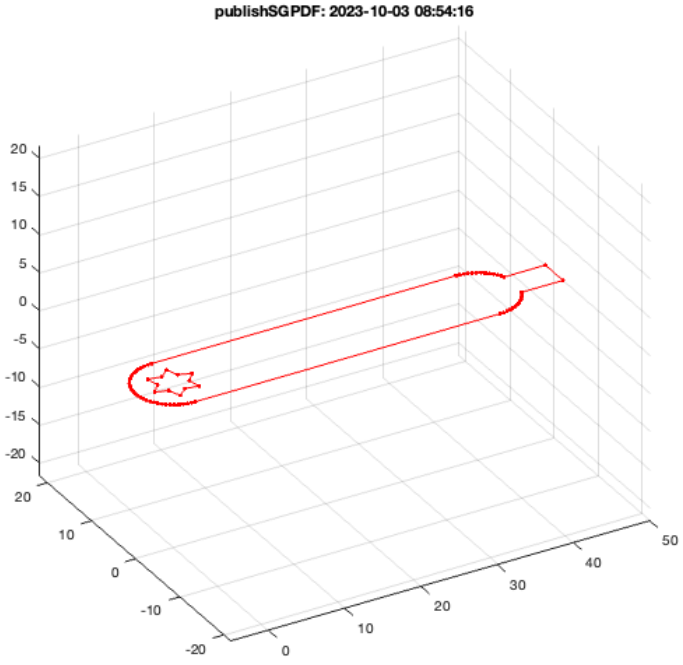
Motivation for this tutorial: (Originally SolidGeometry 5.0 required)

1 Motivation

1.1 Die Betrachtung einer 2D Umlaufkontur und das Füllen mit Dreiecken

Die Kontur besteht nur als 2D Koordaintenliste. Die zwei Konturen sind durch nan nan getrennt.

```
SGfigure(-30,30); CPL=CPLsample(37); CPLplot(CPL,'r.-');
```

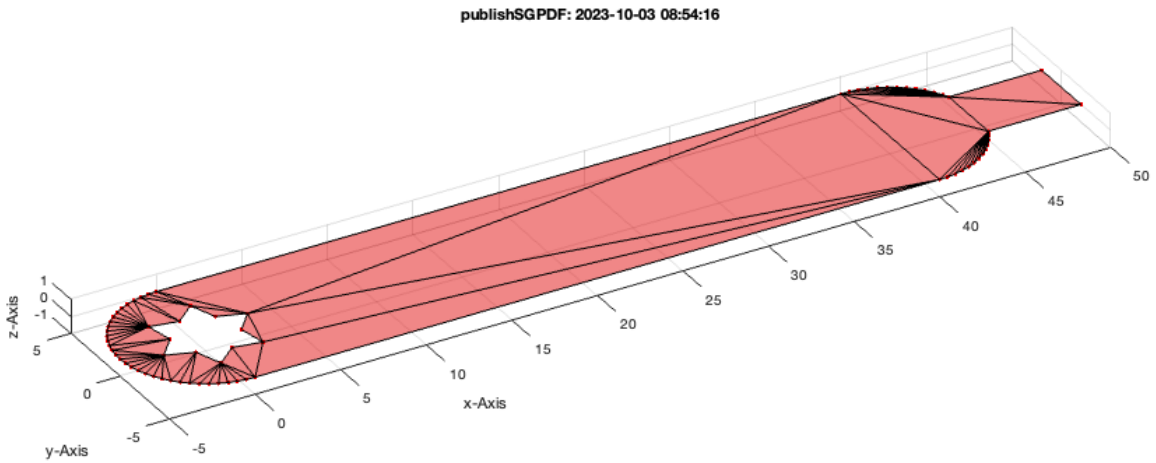


Die Tesselierung in Dreiecke basiert auf drei Schritten:

- 1. Zerlegung der CPL in die durch [nan nan] Koordinaten getrennten Einzelkonturen

- 2. Erzeugung einer Kantenliste [1 2;2 3;3 4] usw. für die Konturpunkte
- 3. Aufruf von Matlab's denaunaytriangulation mit allen 2D-Punktkordinaten und der Kanten als Constraints
- VLFLofCPL führt diese drei Schritte durch

```
[VL,FL]=VLFLofCPL(CPL); VLFLplotalpha(VL,FL,'r',0.5,'k');
```



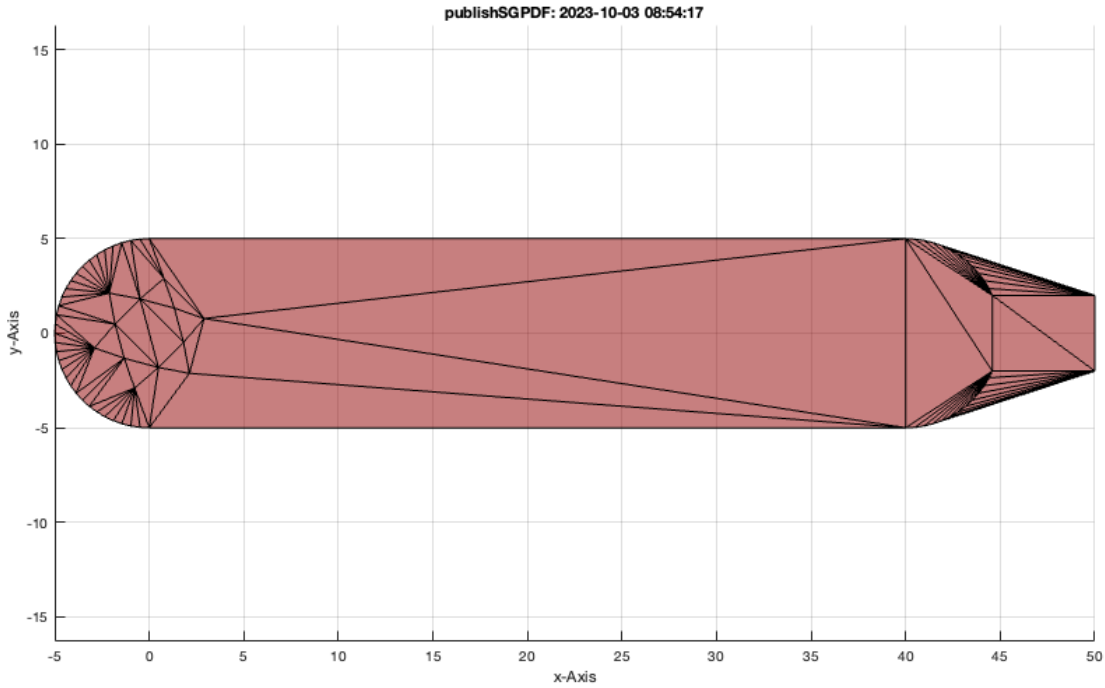
Ohne die Kanten als Constraints würde die Punkte als konvexe Hüllfläche betachtetn werden und es gäbe keine Löcher und keine konkaven Konturen. Die Points sind die Koordinaten, die Constraints sind die Konturkanten und die ConnectivityList ist die Zerlegung in Dreiecke

```
SGfigure
TR2=delaunayTriangulation(VL)
VLFLplotalpha(TR2.Points,TR2.ConnectivityList,'r',0.5,'k');
```

```
ans =
  Figure (1: AOI Matlab Solid Modeler app_2012_11_09) with properties:

      Number: 1
        Name: 'AOI Matlab Solid Modeler app_2012_11_09'
       Color: [1 1 0.9000]
    Position: [31 803 960 540]
       Units: 'pixels'

  Use GET to show all properties
TR2 =
  delaunayTriangulation with properties:
              Points: [73×2 double]
    ConnectivityList: [101×3 double]
         Constraints: []
```



Mit der Funktion freeboundary könnte man sich die Umlaufkontur als Kantenliste berechnen lassen, speziell dann wenn es keine Constraints gibt.

```
freeBoundary(TR2)
```

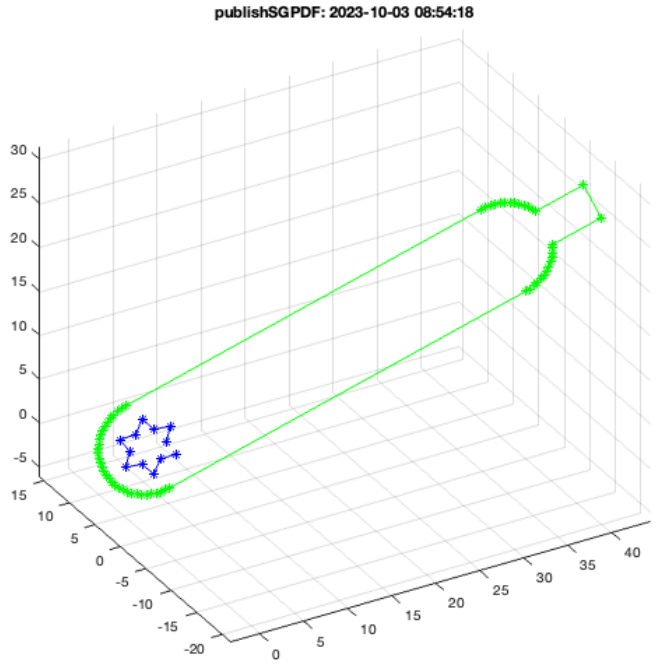
```
ans =
    13    73
    73    72
    72    71
    71    70
    70    69
    69    68
    68    67
    67    66
    66    65
    65    64
    64    63
    63    62
    62    61
    61    60
    60    59
    59    58
    58    57
    57    56
    56    55
    55    54
    54    53
    53    52
    52    51
    51    50
    50    49
    49    48
    48    47
    47    46
    46    45
    45    44
    44    43
    43    42
    42    41
    41    40
    40    39
    39    38
    38    28
    28    27
    27    17
    17    16
    16    15
    15    14
    14    13
```

1.2 Die Betrachtung einer 2D Umlaufkontur im 3D-Raum und das Füllen mit Dreiecken

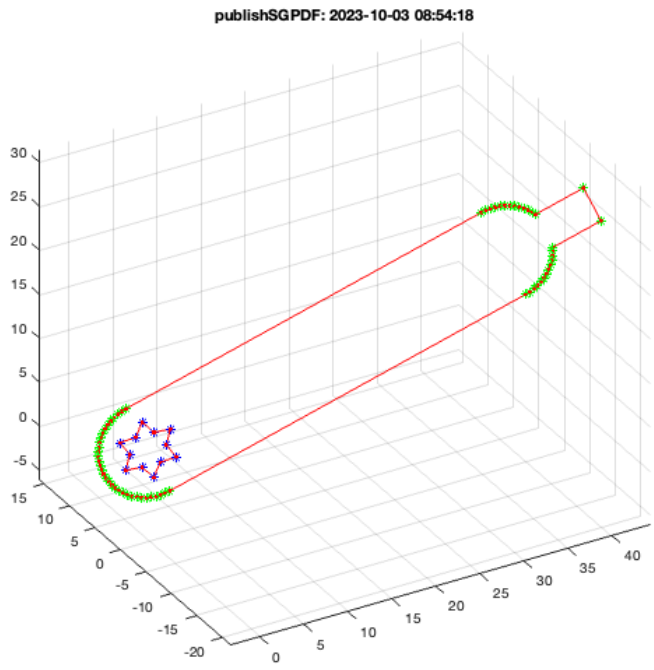
Wir ergänzen nun zu der 2D-Kontur eine z=0 Koordinate und drehen die Kontur im Raum

```
SGfigure(-30,30);
Texp=TofR([pi/6,-pi/6,0],[1 2 3])                    % A Matrix to move to [1 2 3] and rotate
CVL=VLtransT(VLaddz(CPL,0),Texp);                    % Move to [1 2 3] and rotate by using Texp
CVLplot(CVL);                                        % Plot vertices as succeding points as open contour
```

```
Texp =
    0.8660         0   -0.5000    1.0000
   -0.2500    0.8660   -0.4330    2.0000
    0.4330    0.5000    0.7500    3.0000
         0         0         0    1.0000
```



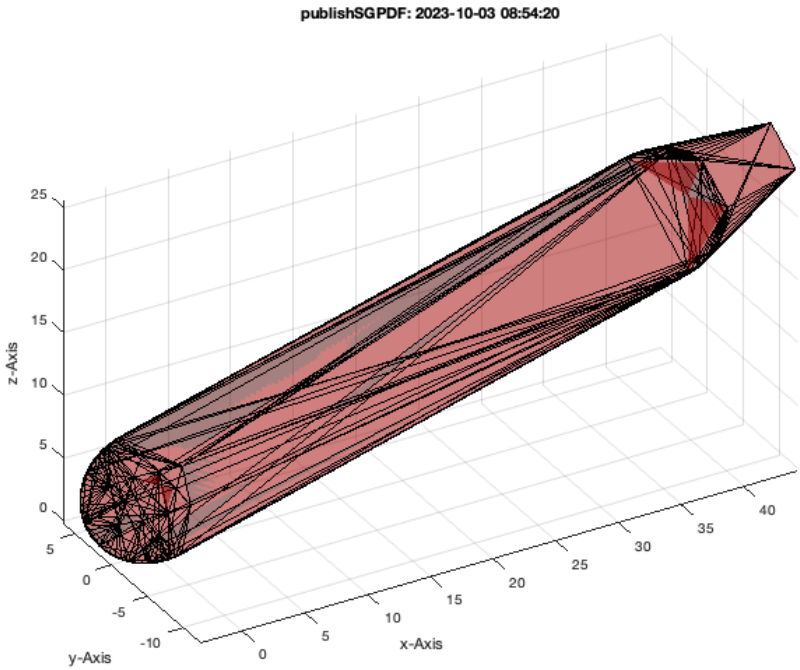

```
CVLplots(CVL,'r.-');                                 % Plot vertices as closed contour
```



1.3 Die Grenzen der delaunayTriangulation in 3D

So wie in 2D die delaunayTriangulation aus den Konturkanten als Constraint eine Zerlegung in Dreicke durchführt wünscht man sich in 3D aus den Oberflächendreiecken als Constraints eine Zerlegung in Tetraeder. Dies ist aber leider aktuell in Matlab nicht möglich. In 3D kann man bei der delaunayTriangulation keine Oberflächendreiecke als Constraints angeben. Es wird immer ein konvexer Hüllkörper aus Tetraedern als Ergebnis geliefert. Die Oberflächendreiecke lassen sich aber durch freeboundary zurückliefern.

```
SGfigure(-30,30);
VL=CVL(~isnan(CVL(:,1)),:);                                    % Verlust der Konturen durch das Entfernen der [nan nan] Separatoren
TR3=delaunayTriangulation(VL);                                 % Tesselierung in Dreiecke
VLFLplotalpha(TR3.Points,TR3.ConnectivityList,'r',0.5,'k');
```



Bei genauer Betrachtung erkennen wir dass es eine Dreieckszerlegung auf der Oberseite und auf der Unterseite gibt. Es ist definitiv nicht das Ergebnis, das wir gesucht haben oder verwenden könnten.

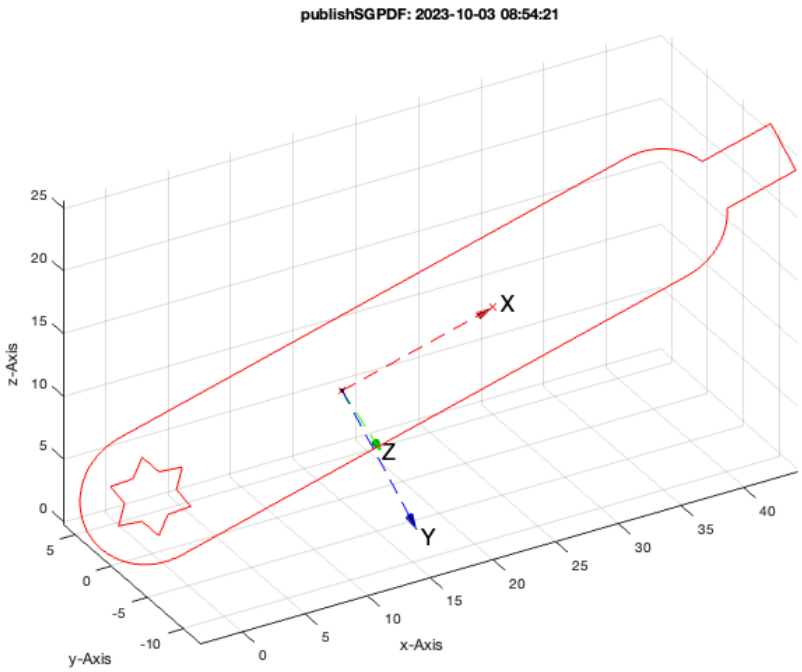
2. Die Transformation planarer Oberflächen aus dem 3D-Raum in den 2D-Raum

Wenn wir wissen, dass wir in diesen Fall eine 2D-Kontur betrachten, die nur in den 3D-Raum transformiert wurde dann kann man die Dreickszerlegung dennoch lösen. Die Tesselierung von 2D Konturen im 3D-Raum basiert auf sechs Schritten:

- 1. Bestimmung des Koordinatensystems T in dem die 3D-Kontur als 2D-Kontur betrachtet werden kann
- 2. Transformation der Konturkoordinaten in die Ebene (über die Inverse von T: inv(T))
- 3. Ignorieren der Z-koordinaten und Betrachtung der Kontur als 2D-Kontur
- 4. Zerlegung in Flächendreiecke
- 5. Transformation der 2D Punkte mit der Matrix T in den 3D-Raum
- 6. Nutzung der in 2D brechneten Flächendreiecke für die 3D-Koordinaten
- VLFLofCPL führt diese drei Schritte durch

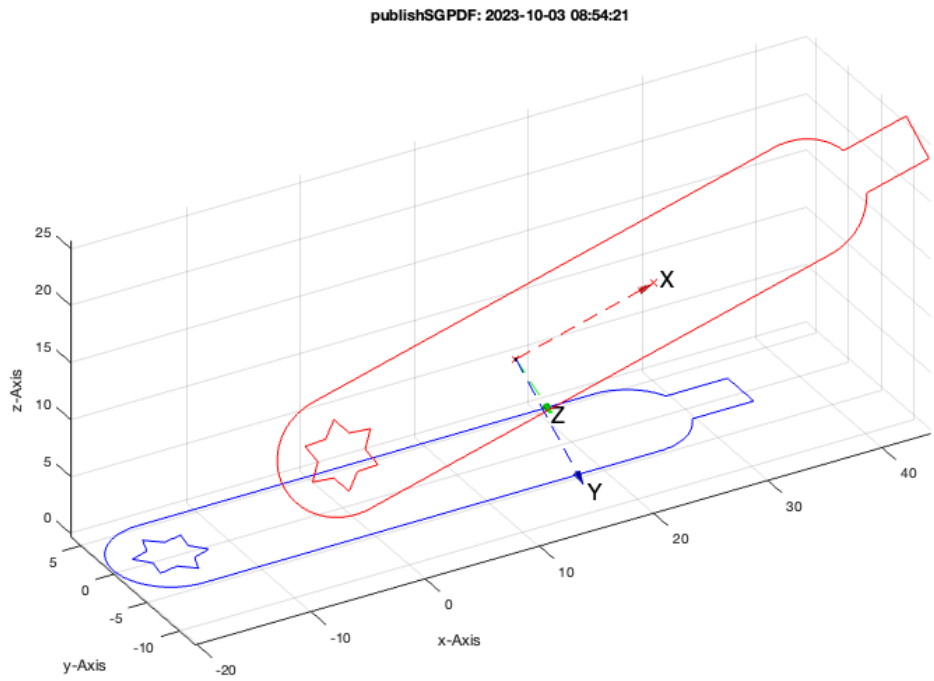
2.1 Transformation von 2D in 3D über die Eigenvektoren der Punktmenge

```
SGfigure (-30,30);
T=TofVL(CVL);           % TofVL benuzt die Eigenvektoren (Körperdimensionen) im 3D zur Berechnung der Transformation
CVLplots(CVL,'r-');     % ursprüngliche planare Konturen im 3D-Raum
tplot(T,CVL);
```

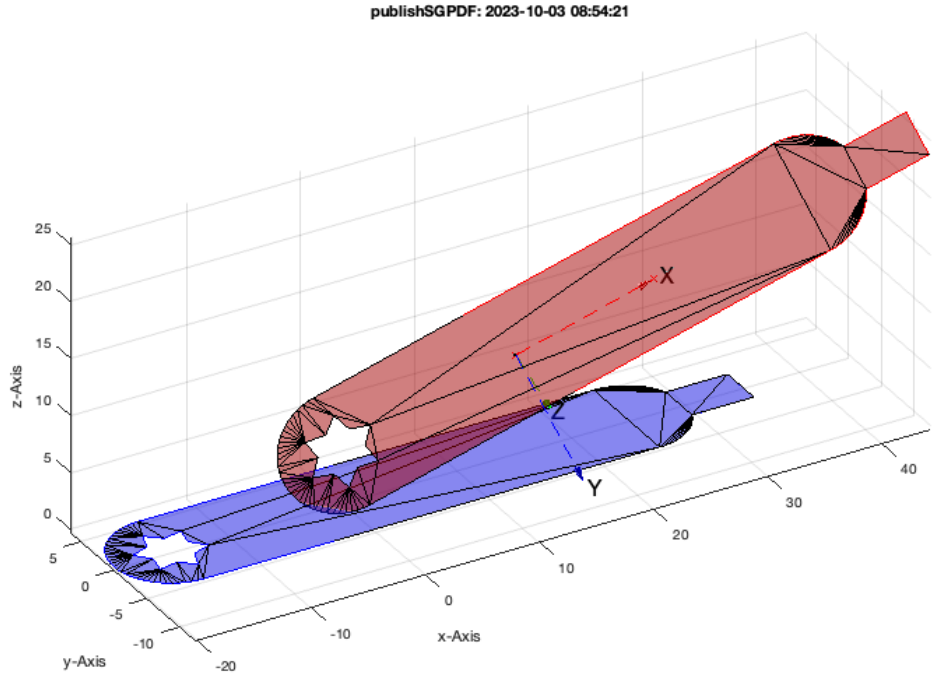



```
CVL2=VLtransT(CVL,inv(T));  % Transformation der Koordinaten
CVL2(1,3)                   % Interessierte werden erkennen, dass hier bereits trigonometische Rechenfehler in der Größenordnung 1e-4 auftreten!
CVLplots(CVL,'r-');     % ursprüngliche planare Konturen im 3D-Raum
CVLplots(CVL2,'b-');    % die in 2D transformierte Kontur
```

```
ans =
   5.7301e-05
```




```
[PL,FL]=VLFLofCPL(CVL2(:,1:2));      % Nur [x y] Kordinaten verwenden; nur Flächenliste verwenden
VL=VLtransT(VLaddz(PL),T);           % Transformation der 2D-Punktliste in den 3D-Raum zurück
VLFLplotalpha(VL,FL,'r',0.5,'k');    % Rot ist die 3D Kontur Flächenfüllung
VLFLplotalpha(PL,FL,'b',0.5,'k');    % Blau ist die 2D Kontur Flächenfüllung
```



2.2 Vorsicht beim 2D in 3D über die Eigenvektoren der Punktmenge

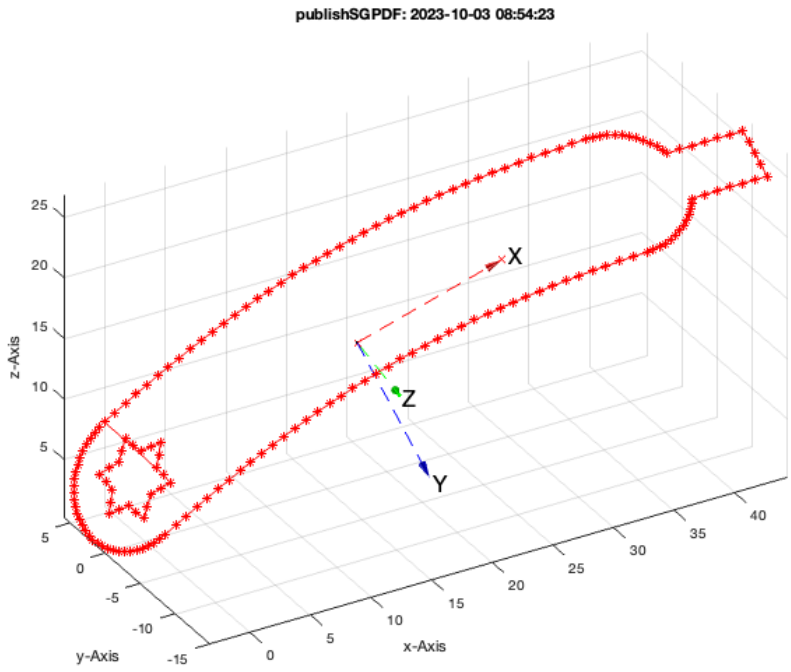
Es gibt ein Problem mit schmaler gekrümmten oder gewinkelten Konturen

Die gekrümmten "planare" Kontur ist in ihrer z-Ausdehnung kleiner als in X-Y

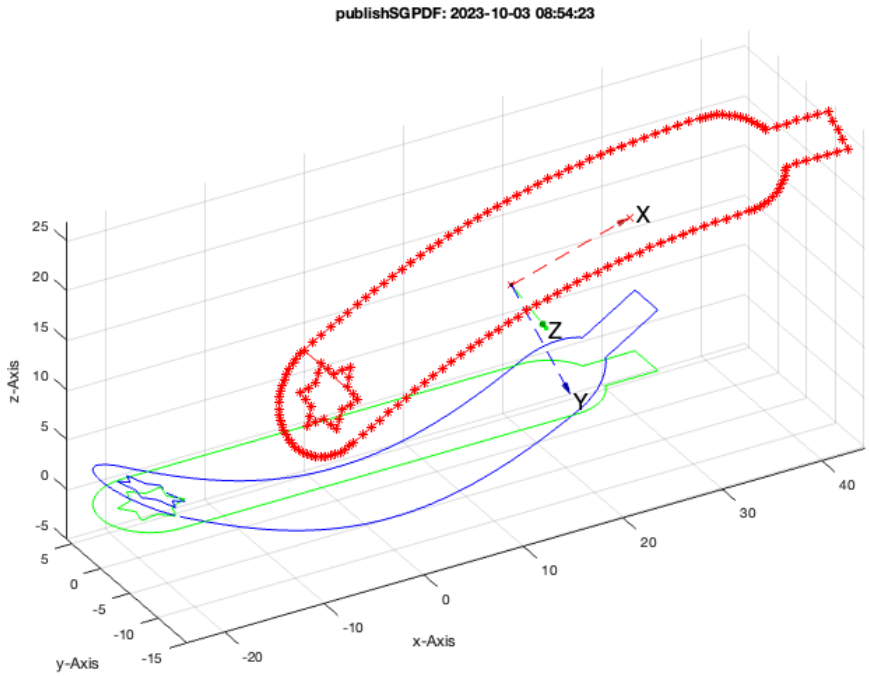
ez wird korrekt ausgerechnet und die Tesselierung in Dreiecke wird erfolgreich sein

```
CVLA=CVLofCPLbendsinus(CPLaddauxpoints(CPLsample(37),1),'x',0.5,10);
CVLA=VLtransT(CVLA,Texp);
```

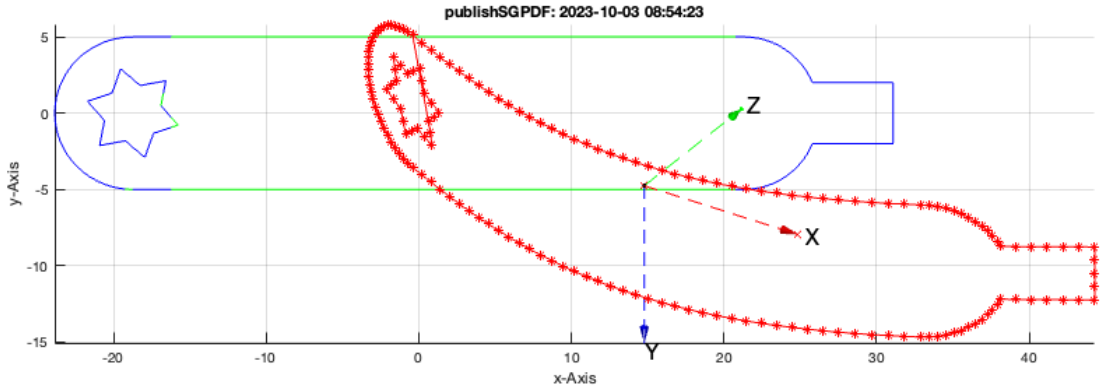
```
TofVL(CVLA);T=ans;                                  % EZ=Vector
```




```
PLA=VLtransT(CVLA,inv(T));
CVLplot(PLA,'b-');                  % Kontur liegt waagrecht auf der x-y Ebene
CPLplot(PLA(:,1:2),'g-');           % Punktlinie ist eine 2D-Projektion der gestauchten Kontur
```



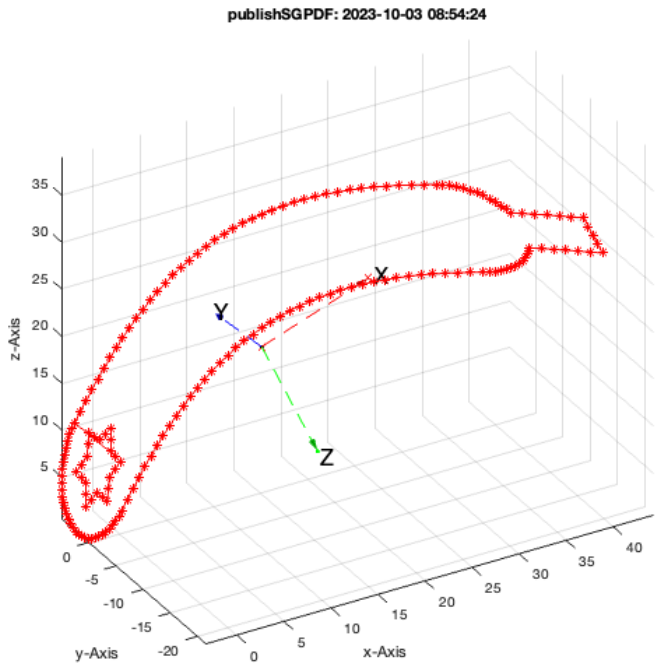

```
view(0,90)
```



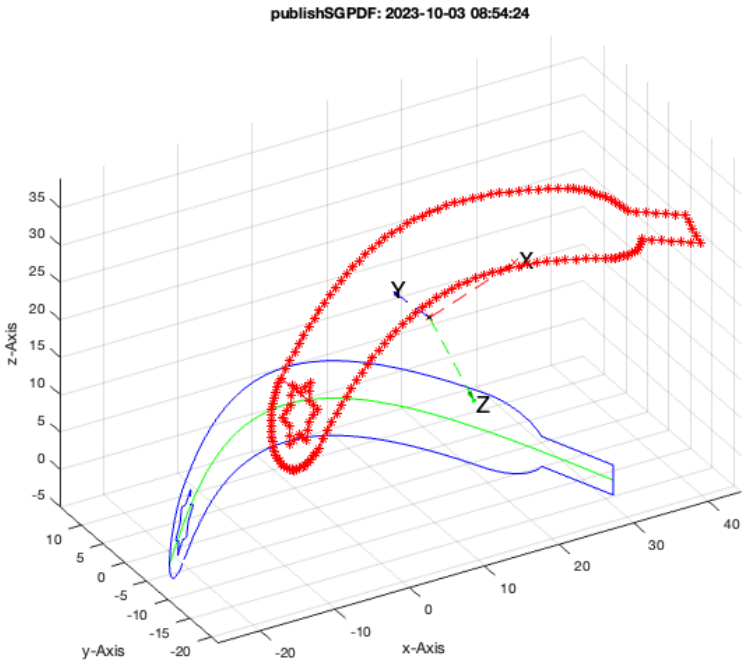
Die gekrümmten "planare" Kontur ist in ihrer z-Ausdehnung größer als in Y

ez wird falsch ausgerechnet und die Tesselierung in Dreiecke nur eine Projektionslinie detektieren

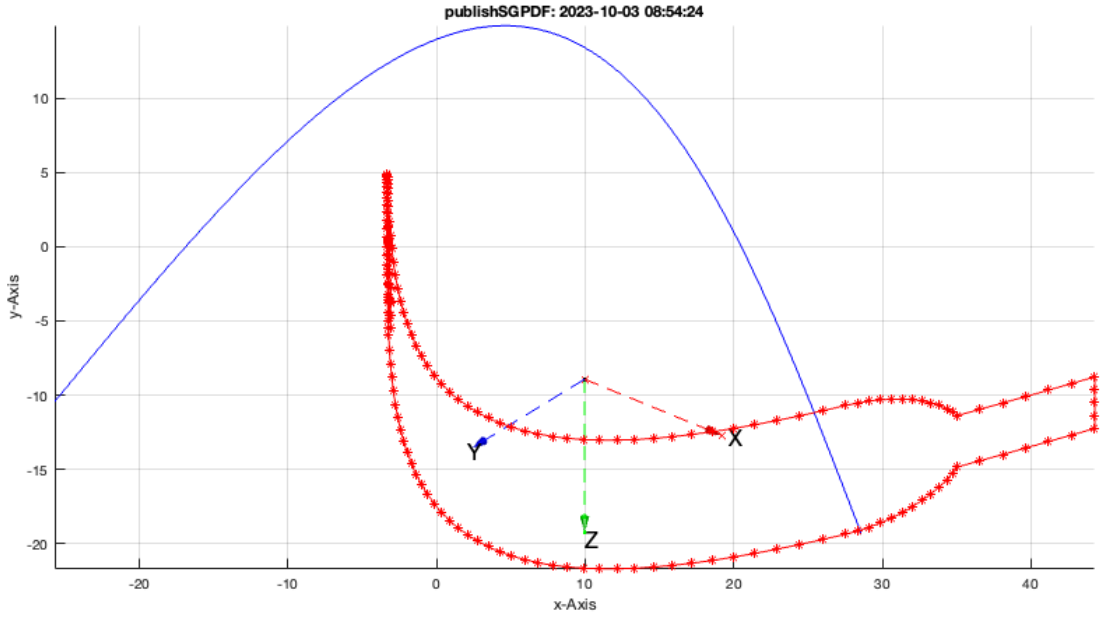
```
CVLB=CVLofCPLbendsinus(CPLaddauxpoints(CPLsample(37),1),'x',0.5,30);
CVLB=VLtransT(CVLB,Texp);
TofVL(CVLB);T=ans;
```




```
PLB=VLtransT(CVLB,inv(T));
CVLplot(PLB,'b-');              % Kontur steht senkrecht auf der x-y Ebene
CPLplot(PLB(:,1:2),'g-');       % Punktlinie ist nur eine Linie
```

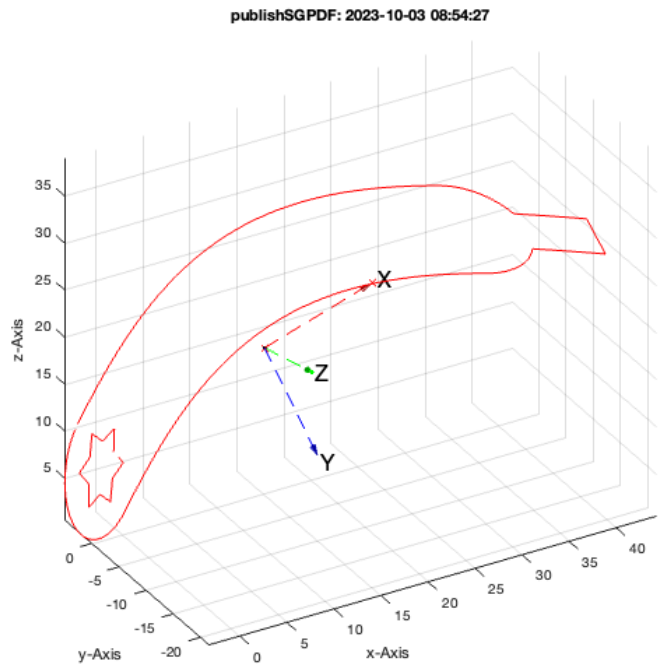



```
view(0,90)                    % direkter Blick von oben
```

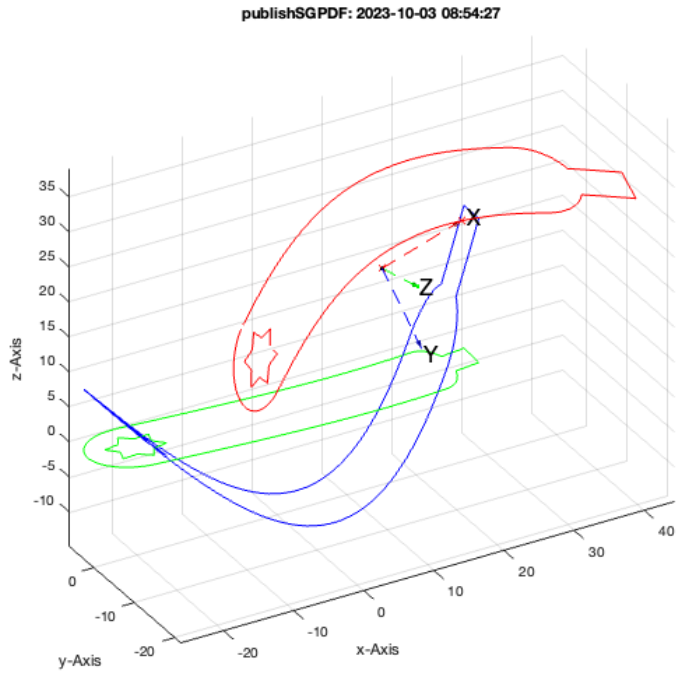


2.3 Erkennen der Normalenvektoren einer Kontour und Nutzung von TofCVL

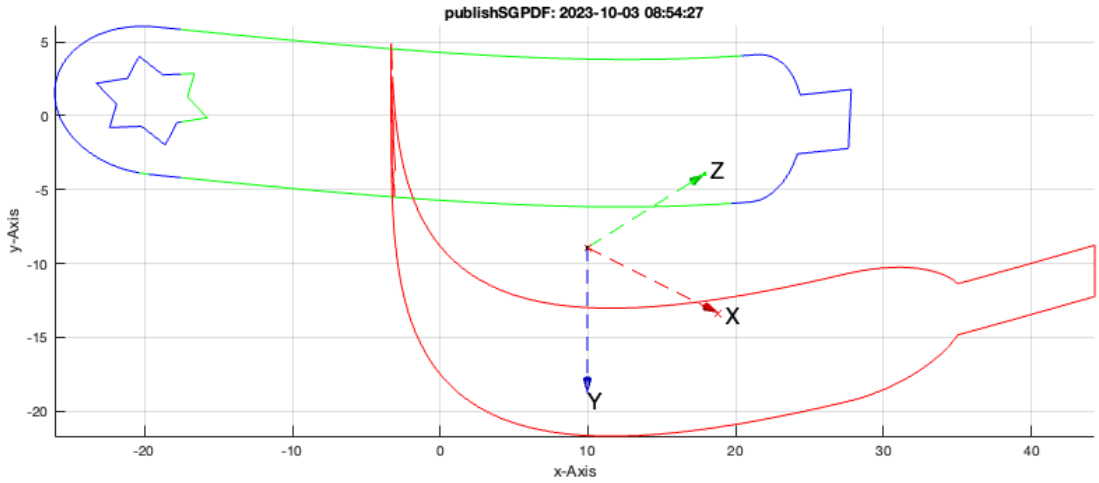
```
CVLedgeNormal(CVLA);   % Important function from summer 2017, three years ago
TofCVL(CVLB); T=ans;
```




```
PLB=VLtransT(CVLB,inv(T));
CVLplot(PLB,'b-');              % Kontur steht senkrecht auf der x-y Ebene
CPLplot(PLB(:,1:2),'g-');       % Punktlinie ist nur eine Linie
```

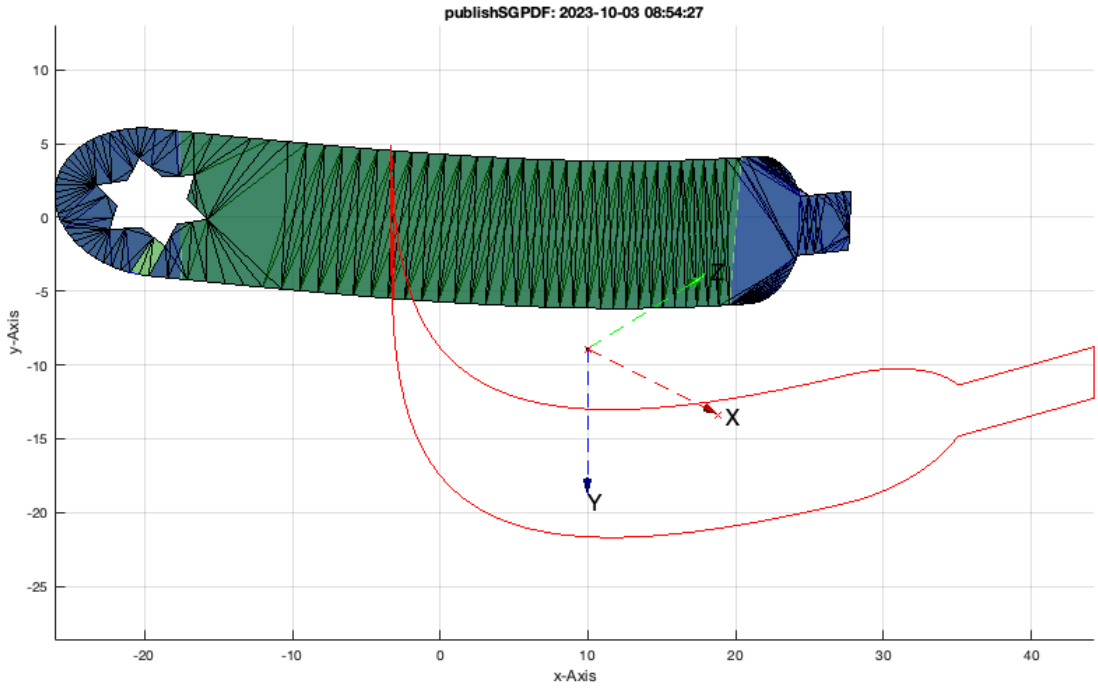



```
view(0,90)                      % direkter Blick von oben
```



2.4 Flächenerstellung mit TofCVL statt TofVL

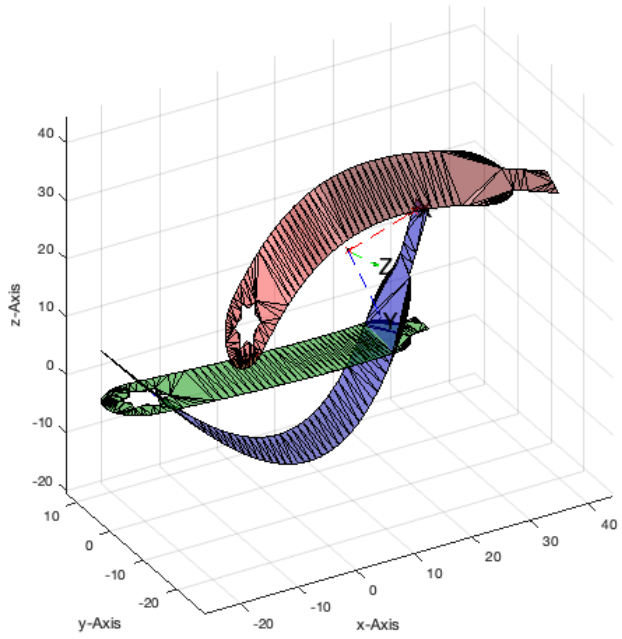
```
[PL,FL]=VLFLofCPL(PLB(:,1:2));      % Nur [x y] Kordinaten verwenden; nur Flächenliste verwenden
VL=VLtransT(VLaddz(PL),T);           % Transformation der 2D-Punktliste in den 3D-Raum zurück
VLFLplotalpha(PLB,FL,'b',0.5,'k');    % Rot ist die 3D Kontur Flächenfüllung
VLFLplotalpha(PL,FL,'g',0.5,'k');    % Blau ist die 2D Kontur Flächenfüllung
```



Die Koordinaten der delaunaytriangulation sind in der gleichen Reihenfolge wie die Konturliste ohne nan nan wenn keine Punkte inegfügt oder entfernt wurden

```
VLFLplotalpha(woNaN(CVLB),FL,'r',0.5,'k');    view(-30,30);
```

publishSGPDF: 2023-10-03 08:54:27



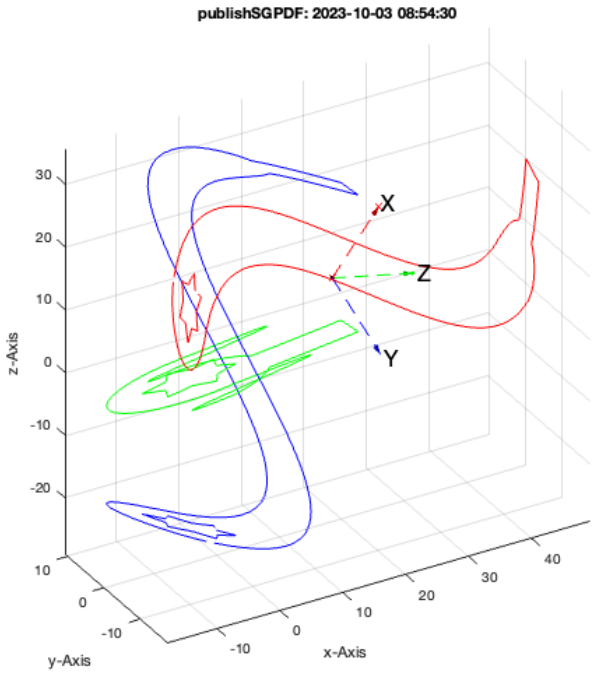
2.5 Krümmung von mehr als 180 Grad - Überlappung der Projektionen - ZERLEGUNG ist notwendig

Die gekrümmten "planare" Kontur ist in ihrer z-Ausdehnung größer als in Y

```
SGfigure(-30,30);
CVLB=CVLofCPLbendsinus(CPLaddauxpoints(CPLsample(37),1),'x',1,30);
CVLB=VLtransT(CVLB,Texp);
CVLplot(CVLB,'r-');
TofCVL(CVLB); T=ans;

PLB=VLtransT(CVLB,inv(T));
CVLplot(PLB,'b-');              % Kontur steht senkrecht auf der x-y Ebene
CPLplot(PLB(:,1:2),'g-');       % Kontur überlappt sich als Projektion
```

publishSGPDF: 2023-10-03 08:54:30

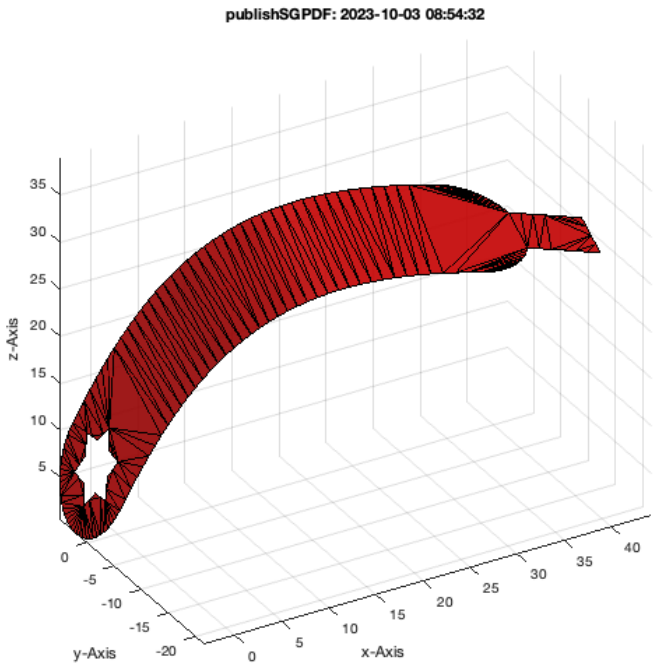


Hier gibt es noch Bedarf nach einer Zerlegung der Konturen in Bereich mit Normalenvektordifferenzen kleiner 180 Grad

3. 3D Konturen

```
SGfigure(-30,30);
CVLB=CVLofCPLbendsinus(CPLaddauxpoints(CPLsample(37),1),'x',0.5,30);   % klappt bis 180 grad
CVLB=VLtransT(CVLB,Texp);
CVLplot(CVLB,'r-');
TofCVL(CVLB); T=ans;

PLB=VLtransT(CVLB,inv(T));
CVLplot(PLB,'b-');              % Kontur steht senkrecht auf der x-y Ebene
CPLplot(PLB(:,1:2),'g-');       % Kontur überlappt sich als Projektion
SGfigure(-30,30);
[VL,FL]=VLFLofCVLdelaunay2D(CVLB);
VLFLplotalpha(VL,FL,'r',0.9,'k');
```



VLFLofCVLdelaunay3D(CVLB)

Final Remarks

```
close all
VLFLlicense
```

```
This VLFL-Lib, Rel.  (2023-Oct-03), is for limited non commercial educational use only!
Licensee: Tim Lueth (Development Version)!
Please contact Tim Lueth, Professor at TU Munich, Germany!
WARNING: This VLFL-Lib (Rel. ) license will exceed at 06-Jul-2078 08:54:33!
Executed 03-Oct-2023 08:54:35 by 'timlueth' on a MACI64 using Mac OSX 13.6 | R2023a Update 5 | SG-Lib 5.4
======================================== Used Matlab products: ========================================
database_toolbox
distrib_computing_toolbox
fixed_point_toolbox
image_toolbox
map_toolbox
matlab
optimization_toolbox
pde_toolbox
simmechanics
simscape
simulink
================================================================================================
```

Published with MATLAB® R2023a