Exercise 05: Creation, Relative Positioning and Merging of Solid Geometries (SG)

2014-11-22: Tim C. Lueth, MIMED - Technische Universität München, Germany (URL: http://www.mimed.de)

Contents

- 1. Summary of the already explained aspects of the VLFL-Toolbox
- 2. Creating, plotting, writing of the struct *Solid Geometry* (SG)
- 3. Spatial transformations of solid geometries and *sets of solid geometries*
- 4. Merging of solid geometries (SG) and sets of solid geometries
- 5. Non-manifold points, edges, and facets of solid geometries (SG)
- 6. Additive Design: Separate or penetrate solid geometries (SG)
- Final remarks on toolbox version and execution date

1. Summary of the already explained aspects of the VLFL-Toolbox

In examples 1-4 we've learned about:

- Point lists (PL), vertex lists (VL), facet lists (VL) and edge lists (EL)
- Functions to import, check, and export of STL-Data
- Functions for generating text strings as solid objects
- Closed polygon lists (CPL) for boolesche operations of even embedded contours
- 2½D design by extrusion of closed polyogon lists
- Spatial transformation of objects in 2D and 3D

2. Creating, plotting, writing of the struct *Solid Geometry* (SG)

Even if it is useful to know that a vertex list (VL), and a facet list (FL) is required for 3D modeling, it is more convenient to use matlab structs for solid geometries (SG). Instead of writing VL or FL, we use SG.VL, SG.FL as variables. The advantage is, that each solid object is described by one struct that contains vertex list and facet list, but can contain other defined information (such as the underlying CPL, PL or EL) or it is open for your own defined information.

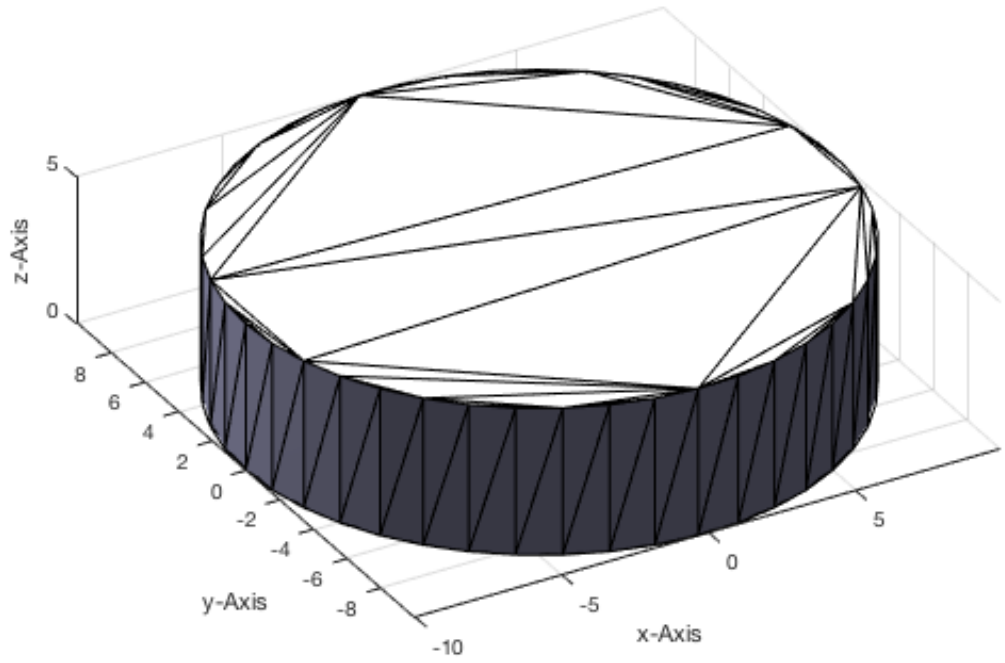
- **SGofCPLz** for extruding a solid object from a CPL similar to VLFLofCPLz.
- **SGplot** for plotting one or more solid objects similar VLFLplots.
- **SGchecker** for checking a solid object similar to VLFLchecker.
- **SGwriteSTL** for writing a solid object similar to VLFLwriteSTLb.
- **SGsize** for generating the bounding box of a solid geometry (SG).

```
close all;
nSG=SGofCPLz(PLcircle(10),5)
SGplot(nSG,'r',1); VLFLplotlight(1); view (-30,30);
SGchecker(nSG);
SGwriteSTL (nSG,'EXP05-1');
```

```
nSG =
```

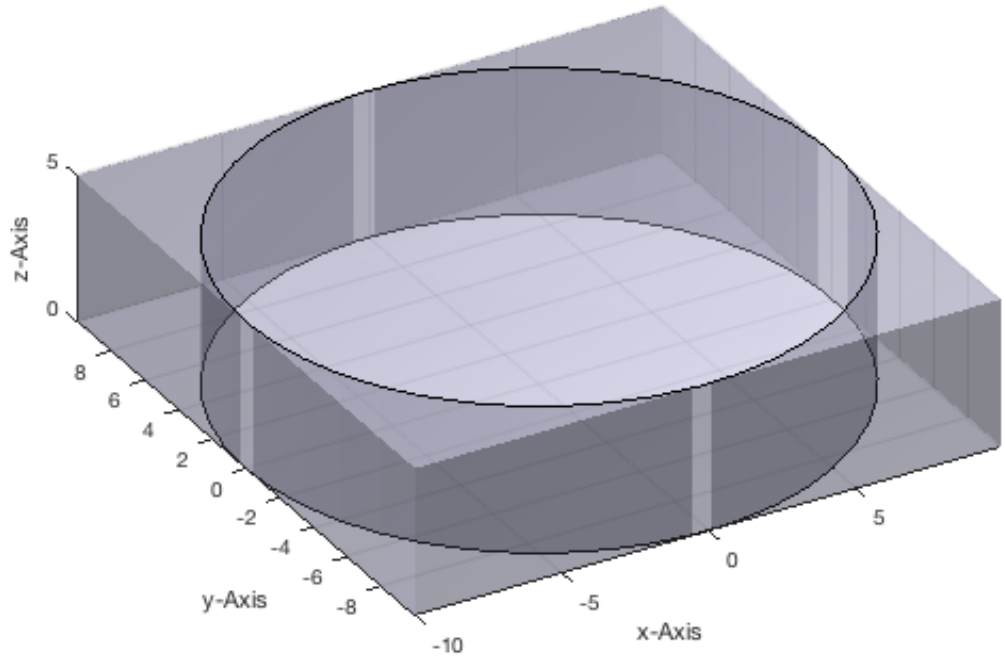
```
struct with fields:

  CPL: [45×2 double]
   VL: [90×3 double]
   FL: [176×3 double]
   PL: [45×2 double]
   EL: [45×2 double]
```



Often it is useful to know the size of the bounding box of an object and to plot it.

```
bs=SGsize(nSG); [BB.VL,BB.FL]=VLFLofBB(bs); SGplot (BB,'w'); VLFLplotlight(1,0.4);
```

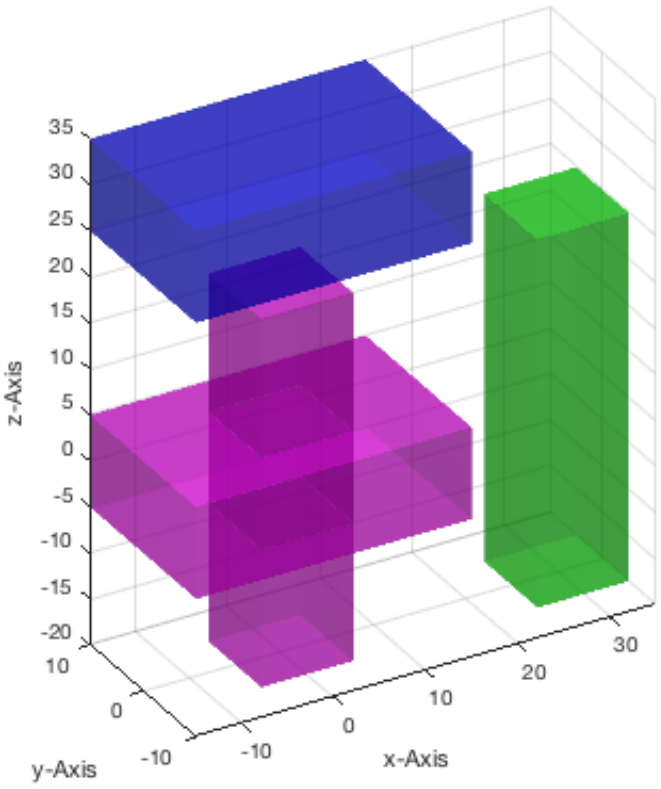


3. Spatial transformations of solid geometries and *sets of solid geometries*

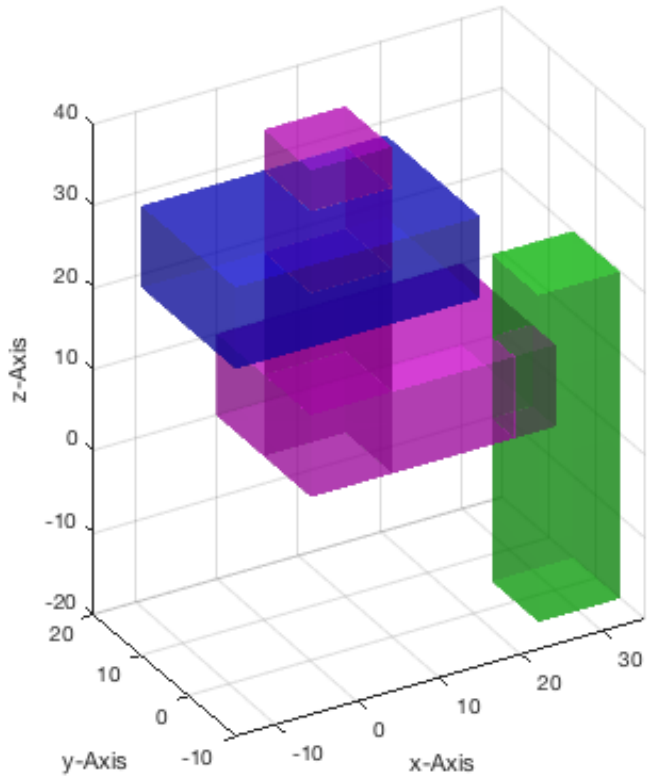
In contrast to manipulate an individual solid geometry as struct, it is often useful to manipulate or handle a set of solid geometries. For this purpose, we use the cell concept of Matlab. A=SGbox([30,20,10]); {A, A, A} is a set of three solid geometries that can be given as arguments of a function and can also be the output argument of a function.

- **SGbox** creates a simple box at the origin indimensions [x y z].
- **SGtransP** moves a solid geometry (SG) or a set of SG by a translation vector.
- **SGtransR** rotate a solid geometry (SG) or a set of SG by a rotation matrix .
- **SGtransT** transform a solid geometry (SG) or a set by a homogenous transformation matrix.
- **SGtrans0** moves a solid geometry (SG) or a set of SG into the coordinate systems origin.
- **SGtrans1** moves a solid geometry (SG) or a set of SG into quadrant 1.

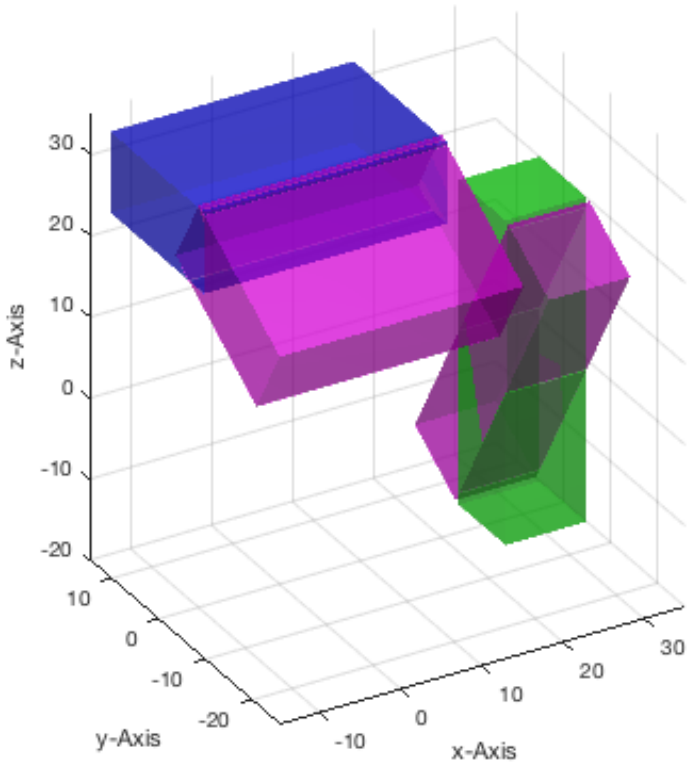
```
close all;
A=SGtransP(SGbox([30,20,10]),[0;0;30]); SGplot (A,'b'); show;
B=SGtransP(SGbox([10,10,40]),[30;0;0]); SGplot (B,'g'); show
view (-30,30);
SGplot(SGtrans0({A,B}),'m'); VLFLplotlight (1,0.5)
```



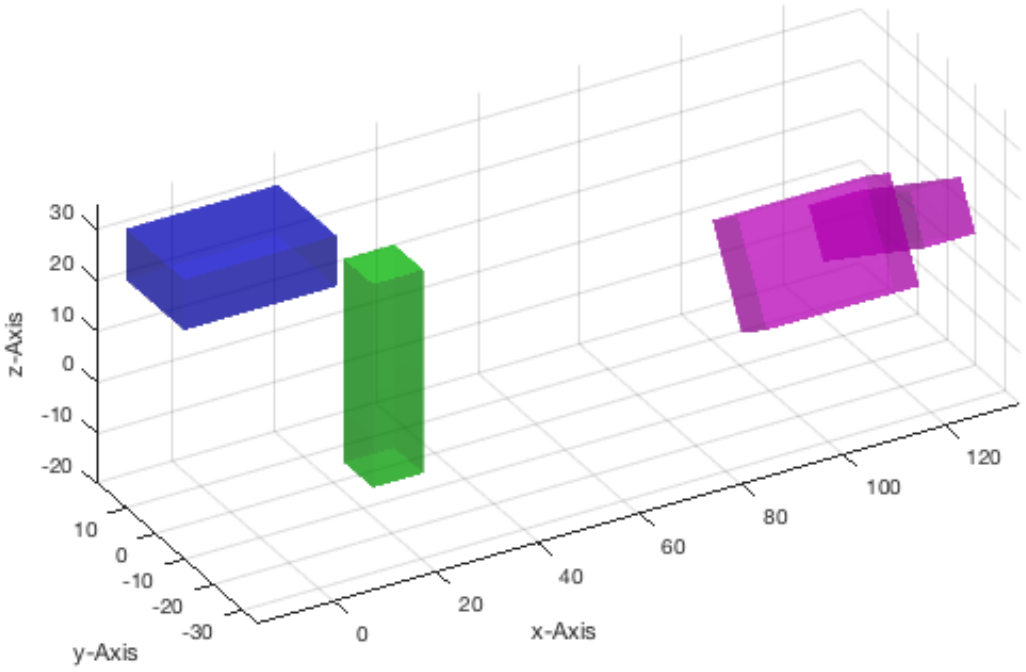

```
close all;
A=SGtransP(SGbox([30,20,10]),[0;0;30]); SGplot (A,'b'); show;
B=SGtransP(SGbox([10,10,40]),[30;0;0]); SGplot (B,'g'); show
view (-30,30);
SGplot(SGtrans1({A,B}),'m'); VLFLplotlight (1,0.5)
```




```
close all;
A=SGtransP(SGbox([30,20,10]),[0;0;30]); SGplot (A,'b'); show;
B=SGtransP(SGbox([10,10,40]),[30;0;0]); SGplot (B,'g'); show
view (-30,30);
SGplot(SGtransR({A,B},rot(pi/6,0,0)),'m'); VLFLplotlight (1,0.5)
```




```
close all;
A=SGtransP(SGbox([30,20,10]),[0;0;30]); SGplot (A,'b'); show;
B=SGtransP(SGbox([10,10,40]),[30;0;0]); SGplot (B,'g'); show
view (-30,30);
SGplot(SGtransT({A,B},[rot(pi/3,0,0),[100;0;0]]),'m'); VLFLplotlight (1,0.5)
```

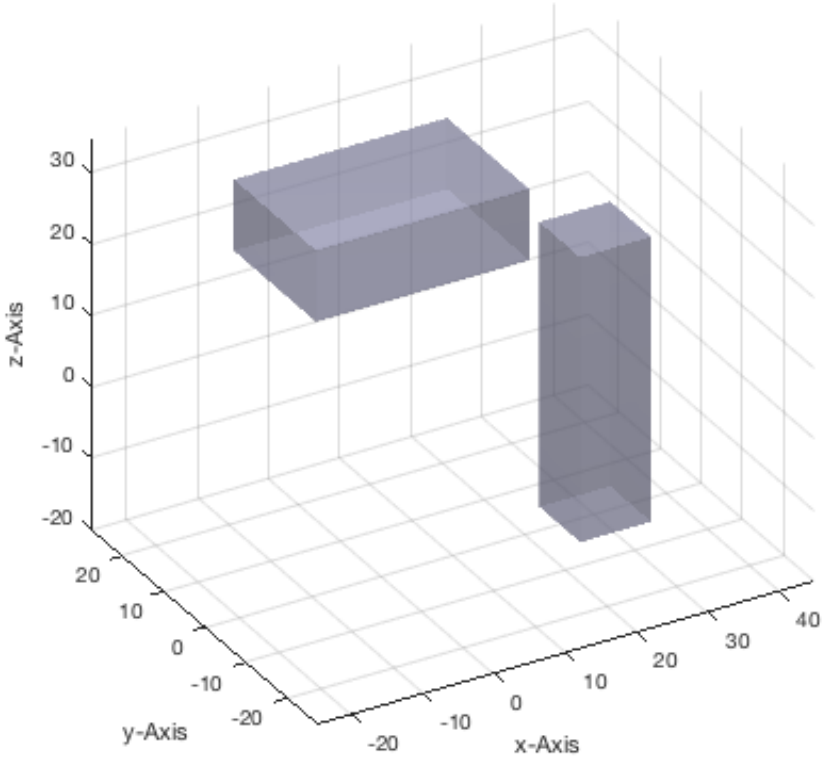


4. Merging of solid geometries (SG) and sets of solid geometries

In the previous example we often created and manipulated two solids. As explained, it is possible to handle several objects at the same time by using sets of elements. Nevertheless, in mosts cases, after some operations we want to merge several solid geometries into one single object. SGcat concatenates the vertex list and the facet list of a set og given solids into one list. Furthermore like in VLFLcat, doubled vertices are detected and removed. It is not possible anymore to separate the objects in general afterwards.

- **VLFLcat** merges two VL/FL into one VL/FL.
- **VLFLcat2** simply concatenates two VL/FL into one VL/FL.
- **SGcat** merges single solids or a set of solids into one solid object.

```
close all;
nSG=SGcat ({A,B}); SGplot (nSG,'w'); view (-30,30); VLFLplotlight (1,0.5)
SGwriteSTL (nSG,'EXP05-2');
```

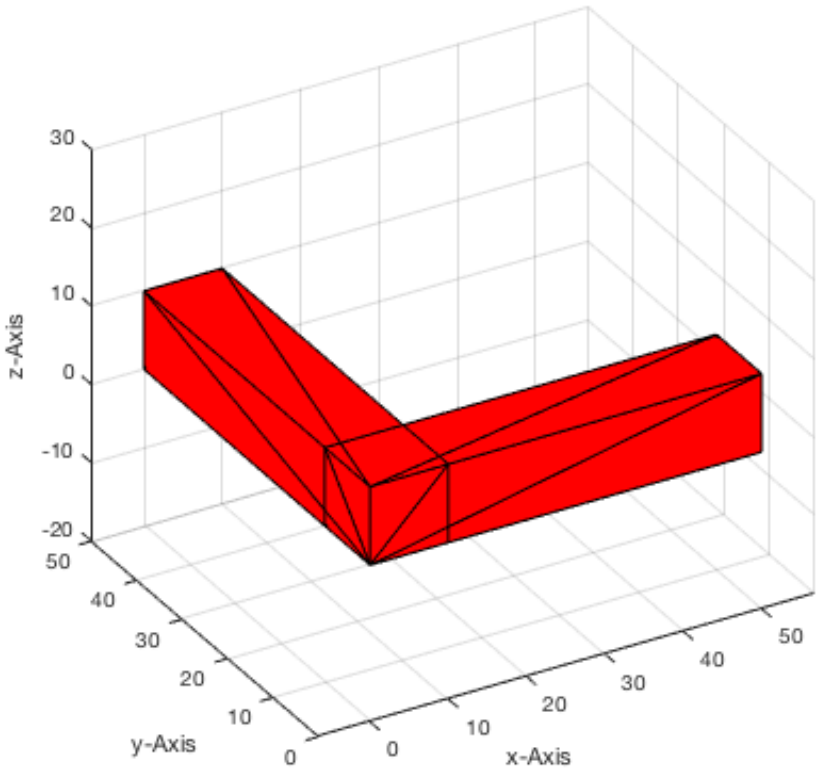


5. Non-manifold points, edges, and facets of solid geometries (SG)

By using functions such as SGcat or VLFLcat/VLcat2, it is very easy and efficient to create solid models and STL-files by simply attaching or penetrating individual solid objects. It is some kind of *additive design of solid objects* in 3D. For a 3D printing process, those additive designed objects are not a real problem, i.e. several independent parts are simply attached or penetrate each other. 3D contour printing of penetrating objects is automatically handled by the slicer, a piece of software that we get to know later.

Nevertheless, as soon as a vertex is used by two independent solids, an edge is used by two independent solids. In this case a slicer software will not be able to solve the manifold problem.

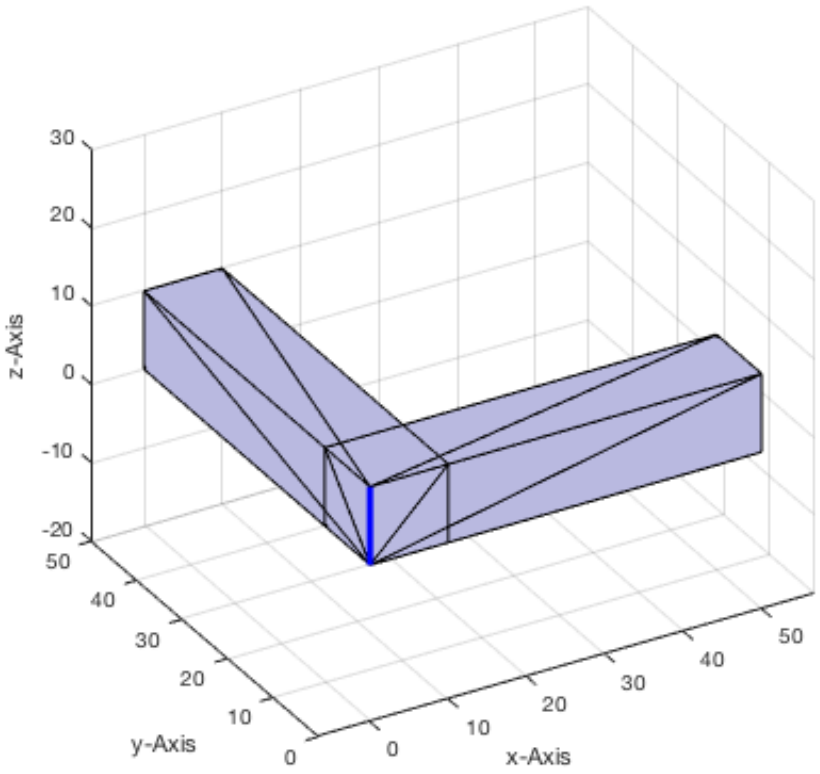
```
close all;
A=SGtrans1(SGbox([10,50,10])); B=SGtrans1(SGbox([50,10,10]));
nSG=SGcat({A,B});  SGplot (nSG); view (-30,30);
[VL,EL,PEL]=SGchecker (nSG);
if ~isempty(VL); VLELplots (VL(:,2:4),PEL,'m*-',4); VLFLplotlight (1,0.7); end
```



If we call SGchecker with a second argument ('plot'), we get a figure showing the non manifold objects that generate the conflict

```
close all; SGchecker (nSG,'plot');
```

```
2 edges [blue] are doubled, not removed
```



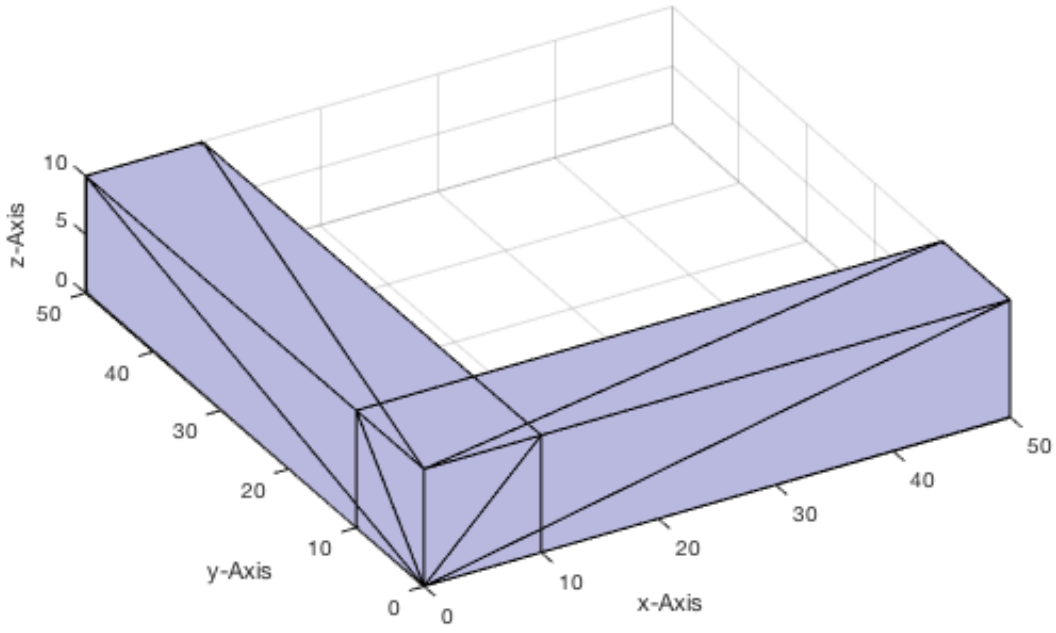
6. Additive Design: Separate or penetrate solid geometries (SG)

During additive solid geometry design, we will always get problems with non-manifold points or edges, as long as we try always to align objects point-to-point, edge-to-edge or face-to-face. As a basic rule, it is better to shorten or increase the length of a object slighly by a micrometer and do not align it with another face, edge, point. Even if the number of points of a solid geometry is increased by this strategy, the number of facets of this solid is decreased. Additive solid geoemtry design is therefore, not an inefficient but an efficient design methodology.

```
close all;
slot=1e-3;
A=SGtrans1(SGbox([10,50,10]));
B=SGtrans1(SGbox([50,10,10])); B=SGtransP(B,[slot;0;0]);
nSG=SGcat({A,B});
SGchecker (nSG,'plot');
```

```
==> SGchecker: 2 Surfaces to check.

Surface 1:
Surface 2:
```



The value for shifting the object about 1 micrometer is much lower than the manufacturing accuracy of the 3D printer. Anyway, if this would not be the case, then simply change it to 1 nanometer (1e-6) or one picometer (1e-9) if we consider a millimeter as default integer unit. It is also clear that we can automate the correction by simply splitting the objects and adding a random submicrometer value to the coordinates.

Another possibility is separating the objects instead of penetrating them. This will lead to the same solution to avoid non manifold edges. Nevertheless, some manufacturing preprocessors analyze STL-Files and detect objetcs that are separated and do not penetrate each other. These objects are then separated and repositioned in the 3D printing working volume to optimize the use of the print job's working volumen and material use.

The later presented function for relative spatial alignment of solid geometries will support a parameter for a gap between objects. Negative gap sizes correspond to a slightly penetration of the solid geometries.

Final remarks on toolbox version and execution date

```
VLFLlicense
```

```
This VLFL-Lib, Rel. 3.6 (2017-Mar-23), is for limited non commercial educational use only!
Licensee: 3.6 Tim Lueth for academic and education 05-Mar-2017 !
Please contact Tim Lueth, Professor at TU Munich, Germany!
WARNING: This VLFL-Lib (Rel. 3.6) license will exceed at 18-Mar-2018!
Executed 22-Mar-2017 22:58:30 by 'lueth' using Matlab 9.1.0.441655 (R2016b) on a MACI64
======================================== Used Matlab products: ============================
==========
map_toolbox
matlab
simmechanics
```

```
simscape
simulink
================================================================================
==========
```

- *Tim Lueth, tested and compiled on OSX 10.7.5 with Matlab 2014b on 2014-11-23*
- *Tim Lueth, executed and published on 64 Bit PC using Windows with Matlab 2014b on YYYY-MMM-DD*

Published with MATLAB® R2016b