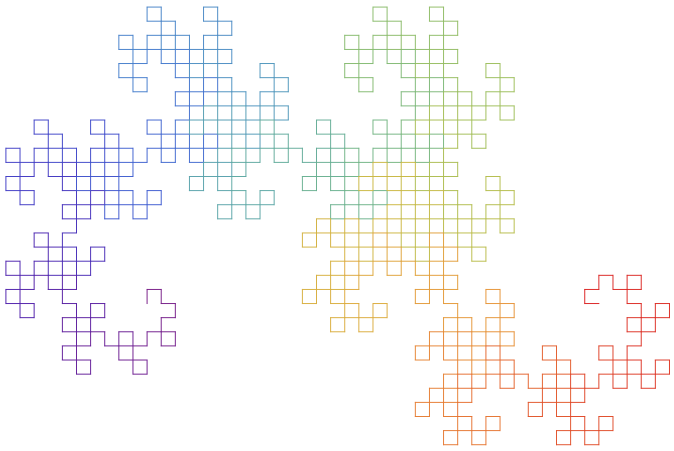
Dragon curve drawing , everything in this chapter works perfectly fine.

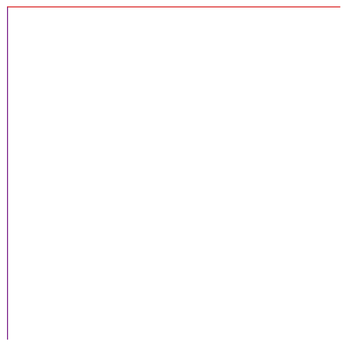
```
Mirror[list_] := list * -1 + 1;
Dragoncurve[0] = {};
Dragoncurve[n_] := Dragoncurve[n] =
   Join[Dragoncurve[n - 1], {1}, Reverse[Mirror[Dragoncurve[n - 1]]]];
(*Having +-1 instead of 0,1 in the sequence is easier for finding
 the new direction by the rotation matrix 'rotation'*)
PmDragoncurve[n_] := Dragoncurve[n] * (2) - 1;
(*There are two colour options to choose from,
the desired function has to be used below.*)
DragonColour1 [n_, k_] := (ColorData["Rainbow"][k/(2^n - 1)]);
DragonColour2[n_, k_] := (If[k < 2^(n-1), Black, Red]);
DragonDraw [n_] := Module[{
                            old = {0, 0},
                            new = {0, 1},
                            rotation = {{0, 1}, {-1, 0}}
                            },
                           line = {(ColorData["Rainbow"][0]), Line[{old, new}]};

                           For[k = 1, k ≤ 2^n - 1, k++,
                                dir = new - old;
                                newdir = (PmDragoncurve[n][[k]] * rotation).dir;
                                {old, new } = {new, new + newdir};

    newline = {DragonColour1[n, k], Line[{old, new}]};
                                line = line ~ Join ~ newline;

                            ];
                           Graphics[line]
                  ];
```

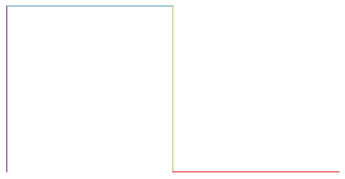
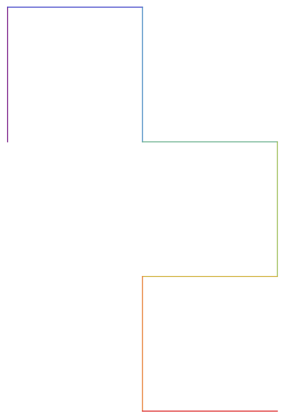
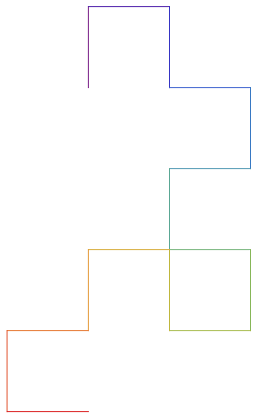
These are examples of what can be done with the dragon curve. Note that there are two colouring options to choose from in the definition above.

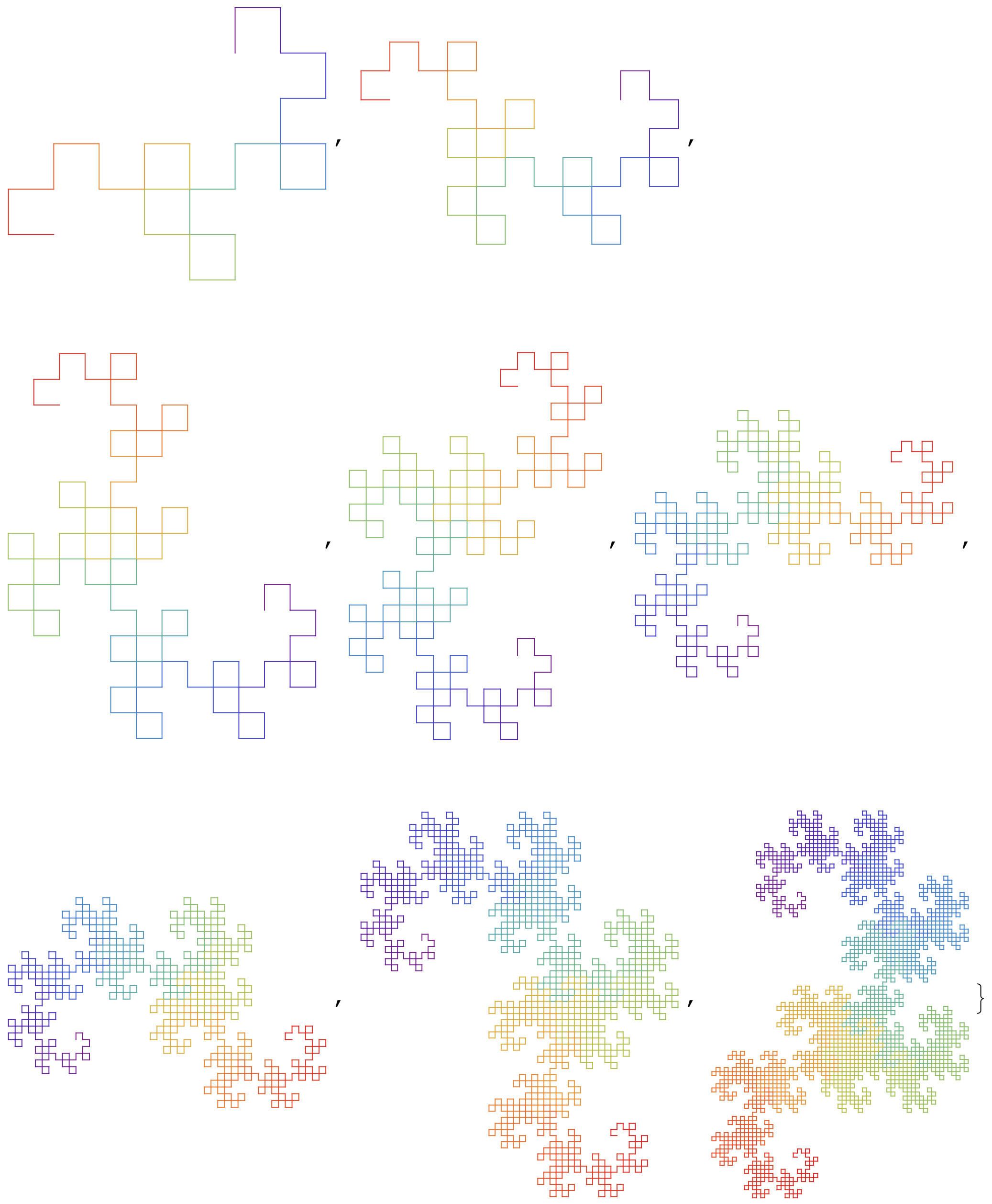
```
DragonDraw[10]
```



```
Table[DragonDraw[i], {i, 0, 12}]
```

{ ,  ,

 ,  ,  ,



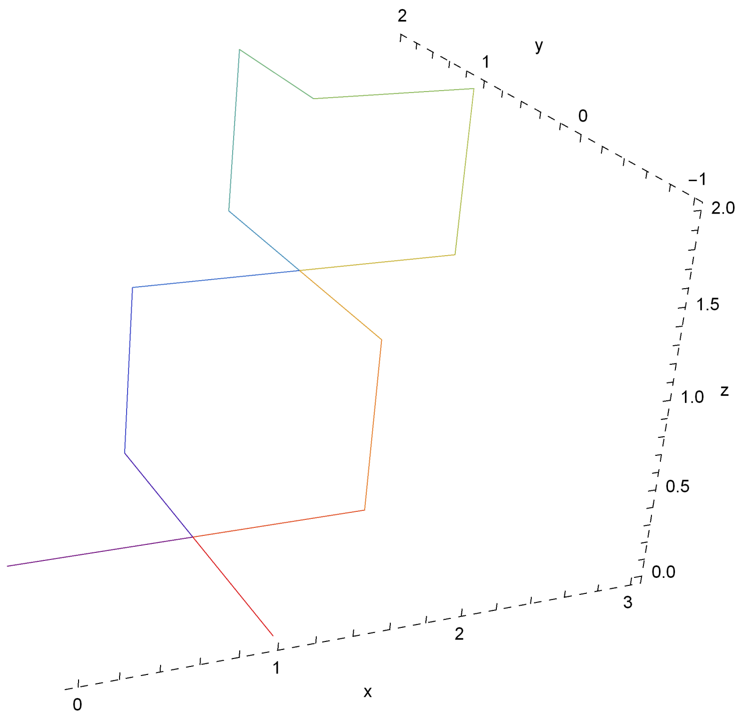
Approach to a 3D analogous of the Dragon curve

The function VisualizeXYZ function takes a list of rotation operations, for example {z,x,y}, and plots a ‘snake’ with exactly these rotations.
The next function FromUtoX takes a list of operations such as u=’up’, -u=’down’, l=’left’ and -l=’right’ and converts them into a list of rotations along x,y,z. One can imagine up, down, left and right to be a valid operation in the local frame of the “traveller along the curve” and my goal is to translate these operations into rotation operations of the static global frame.The issue I had to bypass was that every

rotation along an axis $\lambda \in$ {x,y,z,-x,-y,-z} changes the axis of rotation for any following operation. For example, if I want to go up and then left (={u,l}), I would first rotate along z, but then turning left is not the same as turning left without going up initially.
The general rule I found is: For a rotation λ, every following rotation ϕ has to be transformed into $T_\lambda(\phi)$ as follows:
T_x : (x,y,z) -> (x,z,-y), T_y : (x,y,z) -> (-z,y,x), T_z : (x,y,z) -> (y,-x,z)
Both, the VisualizeXYZ function and the transformation FromUtoX work very well separately, but not together. It appears to me that the order of executing functions in this notebook after quitting the local kernel has an impact on the result, which should definitely not happen!! My guess is that the definition of x,y,z as the 3D rotation matrices could be the cause.

```
(*Don't rotate along the axis of the current direction,
in particular never start with a rotation along x*)
VisualizeXYZ [list_] := Module[{

    m = {{0, -1, 0, 0, 0}, {1, 0, 0, 1, 0}, {0, 0, 0, 0, 0},
                                           {0, -1, 0, 0, -1}, {0, 0, 0, 1, 0}},
                                     old = {0, 0, 0},
                                     new = {1, 0, 0}, x, y, z
                                     },
                              Rainbow[n_, k_] := (ColorData["Rainbow"][k/n]);

   (*Simplest way to define the rotation matrices*)
                              z = Take[m, {1, 3}, {1, 3}];
                              y = Take[m, {2, 4}, {2, 4}];
                              x = Take[m, {3, 5}, {3, 5}];

   line = {(ColorData["Rainbow"][0]), Line[{old, new}]};
                              For[k = 1, k ≤ Length[list], k++,
                                     dir = new - old;
                                     newdir = list[[k]].dir;
    (*This is the product of rotation matrix

     with the vector of the current direction*)
                                     {old, new } = {new, new + newdir};

    newline = {Rainbow[Length[list], k], Line[{old, new}]};
                                     line = line ~ Join ~ newline;
                               ];

   Graphics3D[line, Boxed → False, Axes → True, AxesStyle → Dashed,
                              AxesLabel → {"x", "y", "z"}]
                          ];
```

```
VisualizeXYZ[{z, x, y, z, x, x, z, y, y, z, x, y, z}]
(*This is an example this function working as desired, play around with this.*)
```



```
FromUtoX[list_] := Module[{l1},
                    Shift[list1_, c_] := Switch[c, x, (list1 /. {y ⧴ z, z ⧴ -y}),
      y, (list1 /. {x ⧴ -z, z ⧴ x}), z, (list1 /. {x ⧴ y, y ⧴ -x}),
                                                        -x, (list1 /. {y ⧴ -z, z ⧴ y}),
      -y, (list1 /. {x ⧴ z, z ⧴ -x}), -z, (list1 /. {x ⧴ -y, y ⧴ x})];
                    Transform[list2_] := list2 /. {u ⧴ z, l ⧴ y};
                    Rec[{e_}] := {e};
                    Rec[list3_] := If[Length[list3] ≥ 2,

      {list3[[1]]} ~ Join ~ FromUtoX[Shift[list3[[2 ;;]], list3[[1]]]],
                                   Print["Bullshit, the list is too short"]
                              ];
                    l1 = Transform[list];
                    Rec[l1]
                  ];
```

These operations always work

```
FromUtoX[{u, l, u}](*Play around with this function,
keep in mind that you maybe have to quit the local kernel and only define this
 function. A good idea to understand how this recursion works is to insert
 a print line in the definition that prints the partially translated list.*)
```

```
{{{0, -1, 0}, {1, 0, 0}, {0, 0, 0}},
 {{0, 0, 0}, {0, 0, 1}, {0, -1, 0}}, {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}}
```

```
FromUtoX[{u, l, u, u, -l, u}]
```

```
{{{0, -1, 0}, {1, 0, 0}, {0, 0, 0}}, {{0, 0, 0}, {0, 0, 1}, {0, -1, 0}},
 {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}, {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}},
 {{0, 0, -1}, {0, 0, 0}, {1, 0, 0}}, {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}}
```

```
FromUtoX[{l, u, l, u, l}]
```

```
{{{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}},
 {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}}, {{0, -1, 0}, {1, 0, 0}, {0, 0, 0}},
 {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}, {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}}}
```

These operatioins (usually) don’t work when the function VisualizeXYZ has been ‘compiled’ = ‘defined’ before.

```
FromUtoX[{l, u, l, u, l, -l, -u}]
```

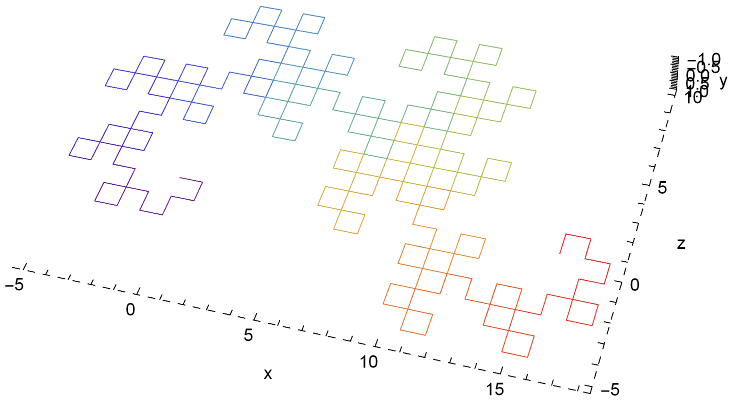
```
{{{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}},
 {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}}, {{0, -1, 0}, {1, 0, 0}, {0, 0, 0}},
 {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}, {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}},
 {{0, 0, -1}, {0, 0, 0}, {1, 0, 0}}, {{0, 1, 0}, {-1, 0, 0}, {0, 0, 0}}}
```

```
FromUtoX[{l, u, l, u, l, -l, -u}]
```

```
{{{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}},
 {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}}, {{0, -1, 0}, {1, 0, 0}, {0, 0, 0}},
 {{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}, {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}},
 {{0, 0, -1}, {0, 0, 0}, {1, 0, 0}}, {{0, 1, 0}, {-1, 0, 0}, {0, 0, 0}}}
```

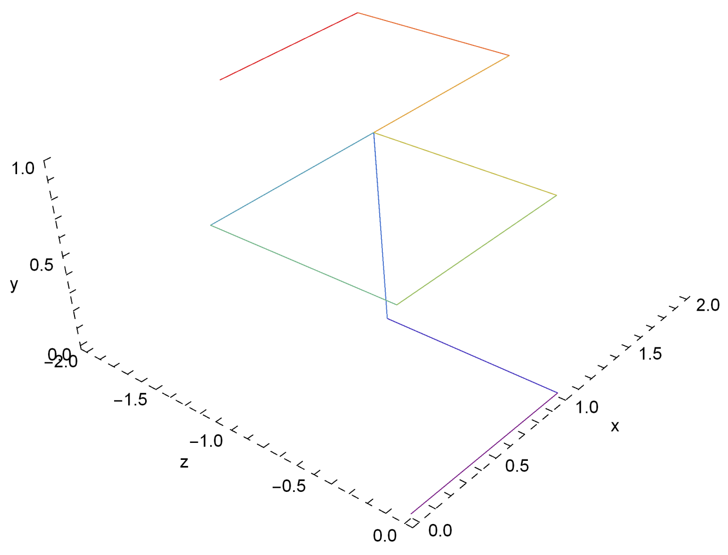
Using rotations only around the orthogonal axis apparently works. Iterations over 9 is however to much for my computer

```
VisualizeXYZ[FromUtoX[Dragoncurve[8] /. {1 :> 1, 0 :> -1}]]
(*The conventional Dragon curve visualized in three dimensions.*)
```



This function obviously only works if both functions ‘VisualizeXYZ’ and ‘FromUtoX’ are defined, so the prblems mentioned above might arise.
I could observe that sometimes not all operations in the list are performed, this could indicate that the list includes a rotation along the axis of the current direction, this however shouldn’t happen according to the algorithm used in the function ‘FromUtoX’.
Another very common error that I don’t understand is “Switch::argct : Switch caled with 12 arguments. Switch must be called with an odd number of arguments.”

```
VisualizeXYZ[FromUtoX[{l, u, l, u, u, u, -l, u, u}]]
```



Further steps and ideas

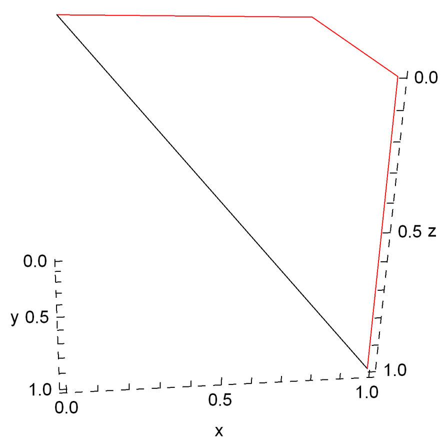
If everything works, my next step will be to plot several fraktal-like patterns involving not {0,1}, but {+-l, +-u} and see if something interesting happens.
The wikipedia arctilce on the Dragon curve has the following image that shows how in every iteration any line (dashed) is replaced by two new lines, one line along each of the 2 axes in the plane.
So another promising idea is to use an iteration process in which every line in the 3D curve is replaced by 3 new lines, one along each of the 3 axes in the 3-space.

```
(*This path has to be changed to reload picture*)
Import[
 "/Users/lennart/Dropbox/Math/Mathematica/Source/2.1/Döppenschmitt_Lennart_160122
   _Dragoncurve/Dragon_curve_wiki.png"]
```

```
Import::nffil : File not found during Import. >>
```

```
$Failed
```

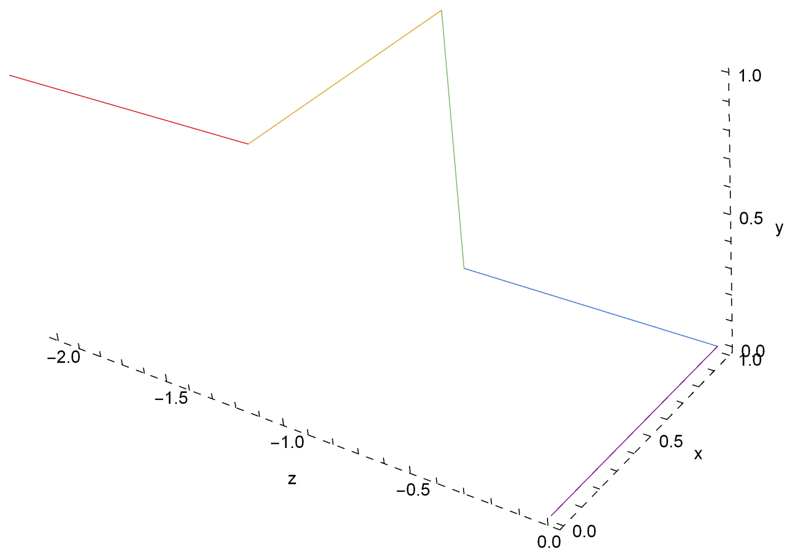
```
Graphics3D[{Red, Line[{{0, 0, 0}, {1, 0, 0}}],
                  Line[{{1, 0, 0}, {1, 1, 0}}],
                  Line[{{1, 1, 0}, {1, 1, 1}}],
                  Black, Line[{{0, 0, 0}, {1, 1, 1}}]},
                         Boxed → False,
 Axes → True, AxesStyle → Dashed, AxesLabel → {"x", "y", "z"}]
```



Collected Error Analysis

This sequence should not be visualized this way, because the last turn the plotted curve makes (from yellow to red) is a down and not right, as specified in the sequence.

```
VisualizeXYZ[FromUtoX[{l, u, l, -l}]]
```



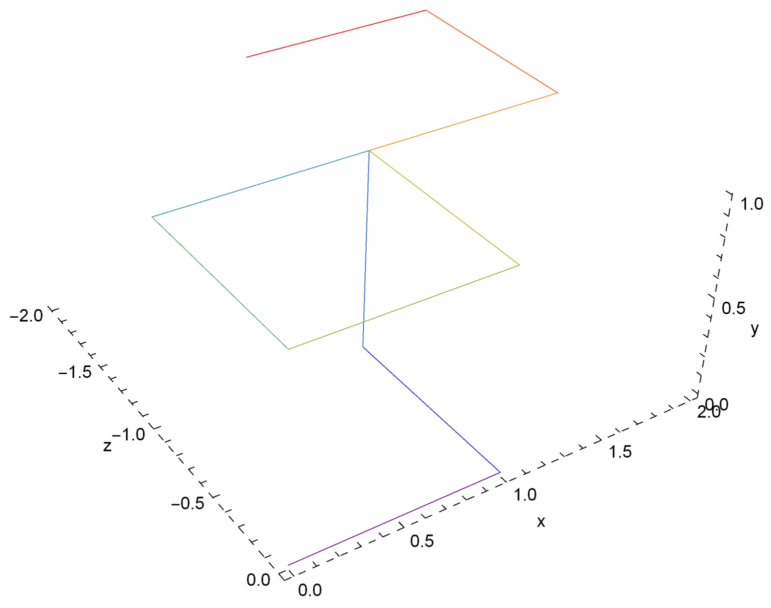
I quit the kernel and defined ‘FromXtoU’ again:

```
FromUtoX[{l, u, l, -l}]
{{{0, 0, 1}, {0, 0, 0}, {-1, 0, 0}}, {{0, 0, 0}, {0, 0, -1}, {0, 1, 0}},
 {{0, -1, 0}, {1, 0, 0}, {0, 0, 0}}, {{0, 0, -1}, {0, 0, 0}, {1, 0, 0}}}
```

This is exactly what it should look like, the graphic above shows definitely that the last rotation is not along the z axis, but aong the y axis!
My guess is that the ‘VisualizeXYZ’ and ‘FromUtoX’ interfere, this shouldn’t happen, since both are modules with local variables.

The same “misinterpretation” of the sequence happens in the following case, because the 3rd last rotation shouldnot be along the same axis as the two last rotations.

```
VisualizeXYZ[FromUtoX[{l, u, l, u, u, u, -l, u, u}]]
```


-2.0
-1.5
-1.0
z
-0.5
0.0
0.0
0.5
1.0
x
1.5
2.0
0.0
0.5
y
1.0