Dragon curve drawing (the original)

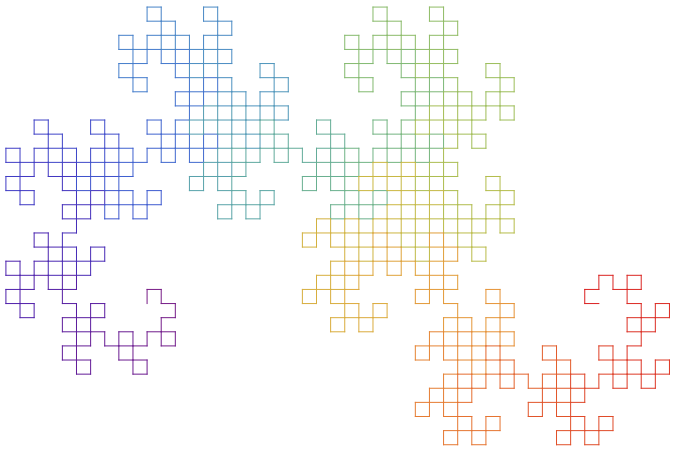
```
Mirror[list_] := list * -1 + 1;
Dragoncurve[0] = {};
Dragoncurve[n_] := Dragoncurve[n] =
   Join[Dragoncurve[n - 1], {1}, Reverse[Mirror[Dragoncurve[n - 1]]]];
(*Having +-1 instead of 0,1 in the sequence is easier for finding
 the new direction by the rotation matrix 'rotation'*)
PmDragoncurve[n_] := 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.:

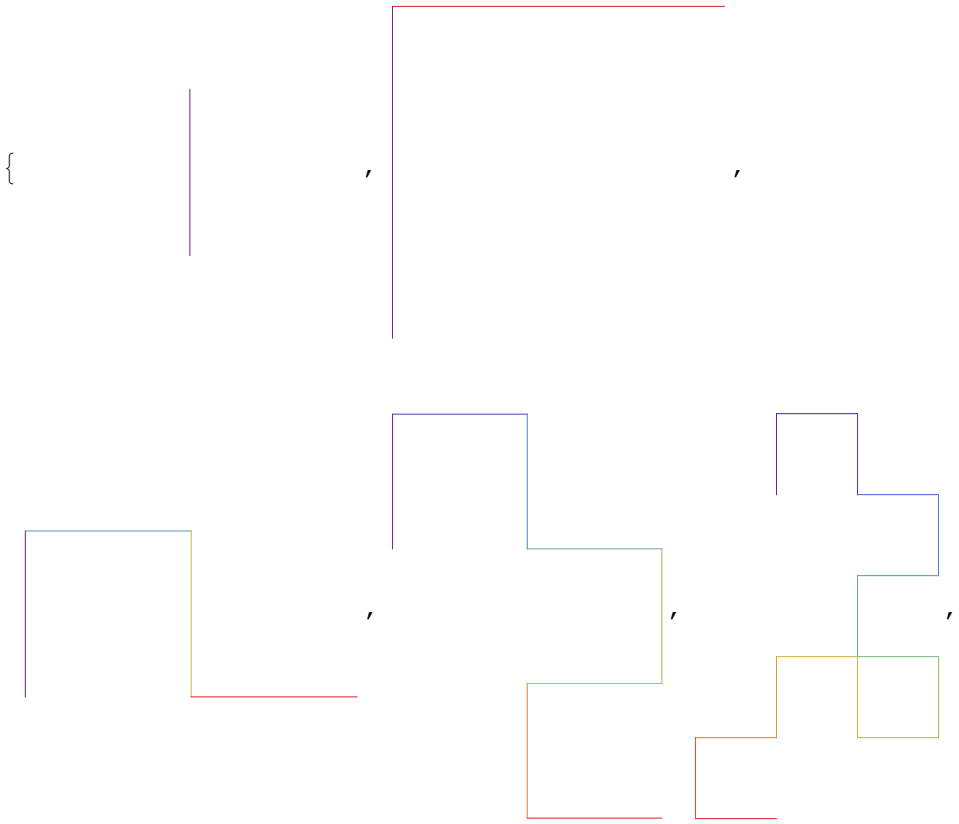
```
Dragoncurve[3]
```

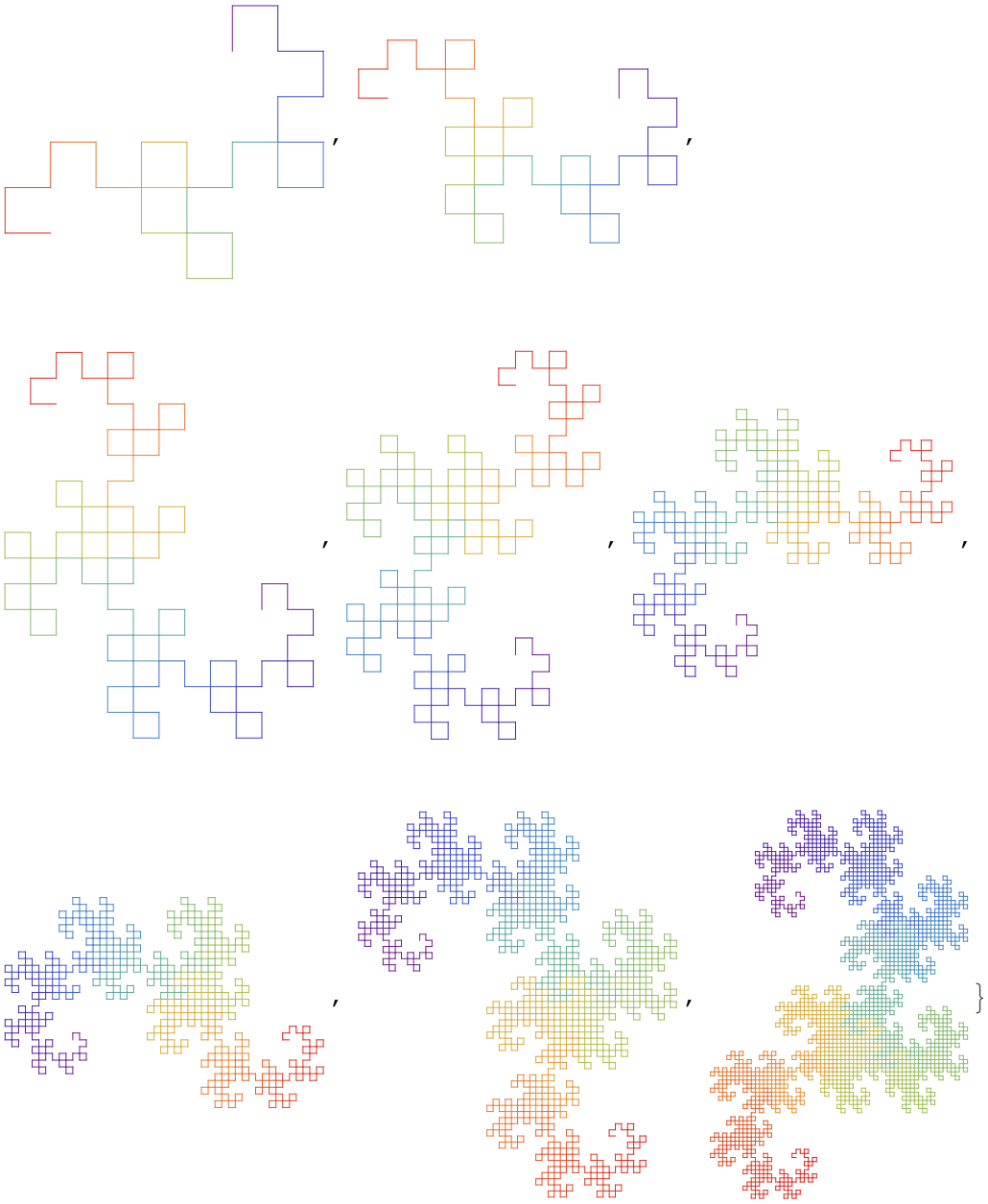
```
{1, 1, 0, 1, 1, 0, 0}
```

```
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, i.e. after each path segment the new segment is rotatet be the axis specified in the string of x’s, y’s and z’s.

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

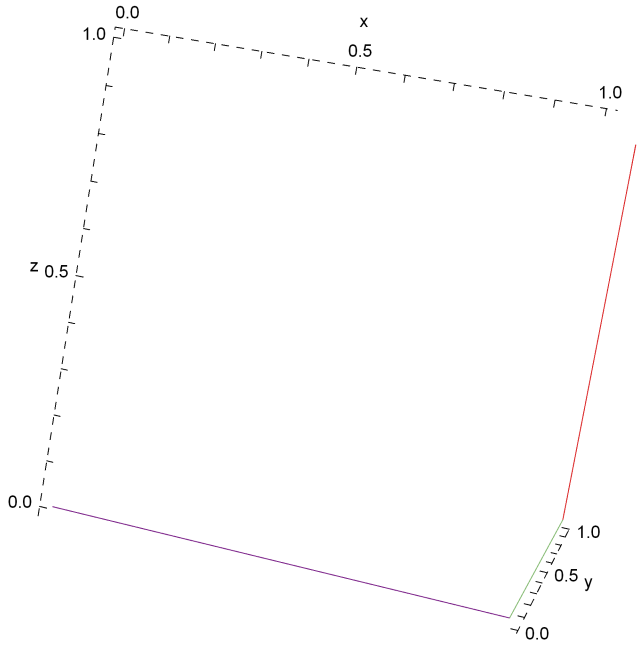
operations into rotation operations of the static global frame.The issue I had to bypass was that every rotation along an axis $\lambda \in \{x,y,z,-x,-y,-z\}$ changes the axis of rotation for any following operation. For example, if I want to go up and then left (="ul"), I would first rotate along z, 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:

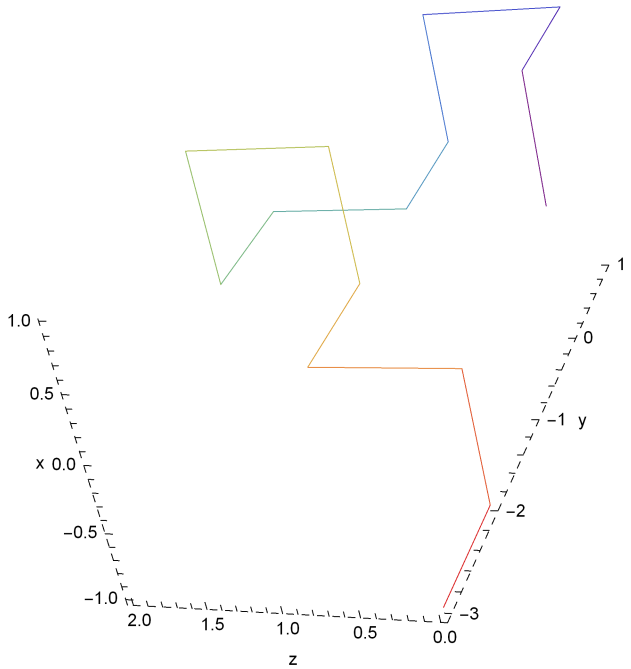
T_x : (x,y,z) -> (x,z,-y), T_y : (x,y,z) -> (-z,y,x), T_z : (x,y,z) -> (y,-x,z) (Note that the minus signs appears whenever you 'shift' the left neighbour in the string xyz and not to the right, it's the 'same' minus as in the cross product.

```
(*Don't rotate along the axis of the current direction,
 in particular never start with a rotation along x in those cases
  the string will only be plottet up to the inconsistency. When
  the end of the snake is red, everything is okay ;)*)
VisualizeXYZ[input_] := 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},
                                  list
                                 },
                                Rainbow[n_, k_] := (ColorData["Rainbow"][k/n]);

   list = StringReplace[input, {"ix" → "X", "iy" → "Y", "iz" → "Z"}];
                                 (*Simplest way
    to define the rotation matrices*)
                                mat["z"] = Take[m, {1, 3}, {1, 3}];
                                mat["y"] = Take[m, {2, 4}, {2, 4}];
                                mat["x"] = Take[m, {3, 5}, {3, 5}];
                                mat["Z"] = -Take[m, {1, 3}, {1, 3}];
                                mat["Y"] = -Take[m, {2, 4}, {2, 4}];
                                mat["X"] = -Take[m, {3, 5}, {3, 5}];

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

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

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

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

```
VisualizeXYZ["zx"]
```




```
VisualizeXYZ["zxiyzixxzyyzxyz"]
(*This is an example this function working as desired, play around with this.*)
```



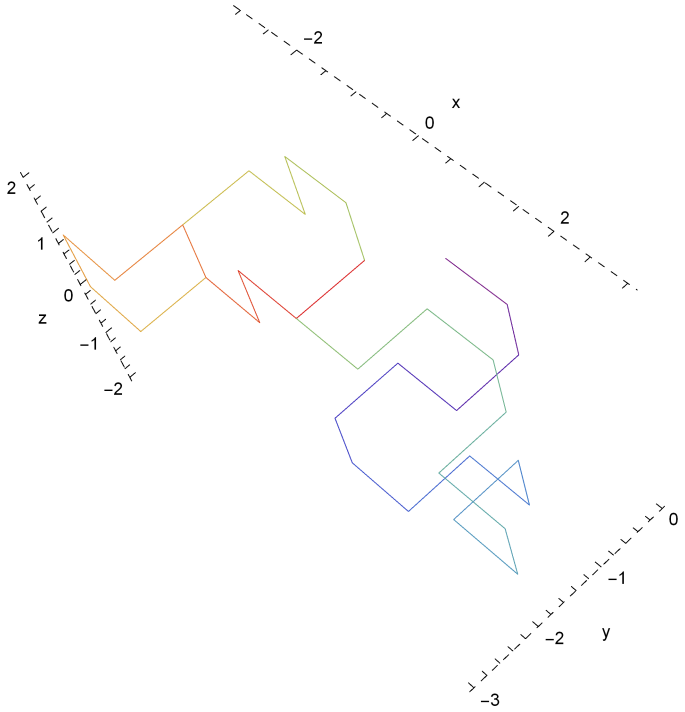
Next I define a function to transform from the local notion of “right, left, up, down” to the tranformation around one of the three axes.

Note that one of the biggest advantages of this local picture is that any sequence ulluLLuuUUl... is valid where ‘zxz’ is not valid as described above. So with this we are reducing ourselfs to the valid set of operations.

```
FromUtoX[word_] :=
  Module[{}, (*This is the ugly and brute force, but it works without errors*)
                  Shift[w_, c_] := Switch[c,
                                                  "x",
     StringReplace[w, {"y" → "z", "Y" → "Z", "z" → "Y", "Z" → "y"}],
                                                  "y",
     StringReplace[w, {"x" → "Z", "z" → "x", "X" → "z", "Z" → "X"}],
                                                  "z",
     StringReplace[w, {"x" → "y", "y" → "X", "X" → "Y", "Y" → "x"}],
                                                  "X",
     StringReplace[w, {"y" → "Z", "Y" → "z", "z" → "y", "Z" → "Y"}],
                                                  "Y",
     StringReplace[w, {"x" → "z", "z" → "X", "X" → "Z", "Z" → "x"}],
                                                  "Z",
     StringReplace[w, {"x" → "Y", "y" → "x", "X" → "y", "Y" → "X"}]
                                              ];

   Transform[l_] := StringReplace[l, {"u" → "z", "l" → "y", "U" → "Z", "L" → "Y"}];
                  Rec[list3_] := If[StringLength[list3] == 1,
                                        list3,

     StringTake[list3, 1] <> Rec[Shift[StringDrop[list3, 1], StringTake[list3, 1]]]
                                    ];
                  Rec[Transform[word]]
               ];
StringPlot[word_] := VisualizeXYZ[FromUtoX[word]];

FromUtoX["ulu"]
```

```
zXy
```

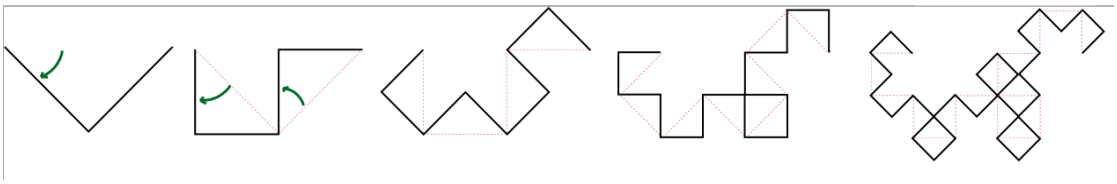
```
StringPlot["lUlLuLuULUluulULUuulUUulUuLuuLullLU"](*Just a random example*)
```



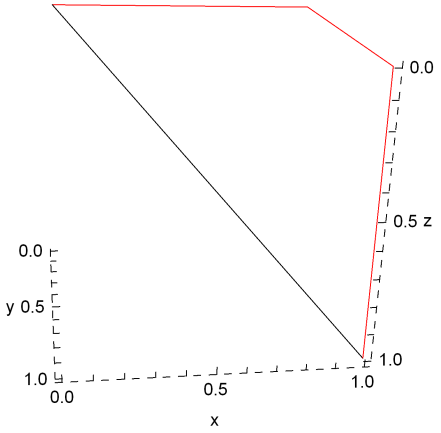
Further steps and ideas & some fractal strings

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 the picture,
using the WikiData function results a picture with a horrible resolution.*)
Import[
 "/Users/lennart/Dropbox/Math/Mathematica/Source/2.1/Döppenschmitt_Lennart_160122
   _Dragoncurve/Dragon_curve_wiki.png"]
```



```
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"}]
```



Now we want to create some fractals in string form, some useful functions for this are the following. This all can be potentially use to create strings of u's and l's which have an interesting visualization.

```
StringReverse["ulul"]
```

```
lulu
```

```
StringRotateLeft["uuuuullllll", 3]
```

```
uullllluuu
```

```
StringReplace["luLuLLu", {"l" → "u", "u" → "L", "L" → "U", "U" → "l"}]
```

```
uLULUUL
```

```
"uluu" <> "lull"
```

```
uluulull
```

```
"uluulull"
```

```
Mirror0[w_] := StringReverse[StringReplace[w, {"1" → "0" , "0" → "1"}]];
Mirror1[w_] := StringReplace[w, {"l" → "u", "u" → "l", "L" → "U", "U" → "L"}];
```

```
Mirror2[w_] :=
  StringReverse[ StringReplace[w, {"l" → "L", "u" → "U", "L" → "l", "U" → "u"}]];
ShiftMirror[w_] := StringReverse[
   StringReplace[w, {"l" → "u", "u" → "L", "L" → "U", "U" → "l"}]];
Curve1[0] = "";
Curve1[n_] := Curve1[n - 1] <> "uL" <> StringReverse[Mirror1[Curve1[n - 1]]];
Curve2[0] = "";
Curve2[n_] := Curve2[n - 1] <> "uL" <> StringReverse[ShiftMirror[Curve2[n - 1]]];
Curve3A[0] = "";
Curve3A[n_] := Curve3[n - 1] <> "U" <> StringReverse[ShiftMirror[Curve3[n - 1]]];
Curve3[0] = "";
Curve3[n_] :=
  Curve3A[n - 1] <> "l" <> StringReverse[ShiftMirror[Curve3A[n - 1]]];
Curve[0] = "";
```

In general I can create an arbitrary curve of this type by specifying some initial string, how ; often it should itearate, what kind of mirror it should use and whether it should reverse the strings as it patches them to the end or not.

```
Curve[initial_, n_, mirror_, Reverse_] :=
  Curve[n - 1] <> initial <> If[Reverse == "Y" || Reverse == "yes",
    StringReverse[mirror[Curve[n - 1]]], mirror[Curve[n - 1]]];
```

This however doesn't completely satisfies me, since depending on the complexity of the mirror, the set of possible strings is very limited by its initial string, which acts a generator.
The extra dimension gives us another generator, so I'd like to have another perspective on the sequence of the dragon curve. It can also be viewed as an inital string where one inserts a version of the current iteration result, which is mirrored in every other case, between every character of the current iteration result. View the sequence as follows:

110 – 100 – 110 – 100 – 110 – 100 – 110 – 100 – 110 – 100 – 110 – 100 – 110 – 100 – 110 – 100
1 1 0 1 0 0 1 1 0 1 0 0
1 1 0
...
So this is what 'Curve4' implements.
The main advantage is that I don't have to jump between different recursive definitions as I did in 'Curve3' and 'Curve3A' to change the string that is inserted in the middle. The downside is that the string length grows not as 2n+1 as before, but as n(n+1), so the capacity of Mathematica is reached farely quickly.

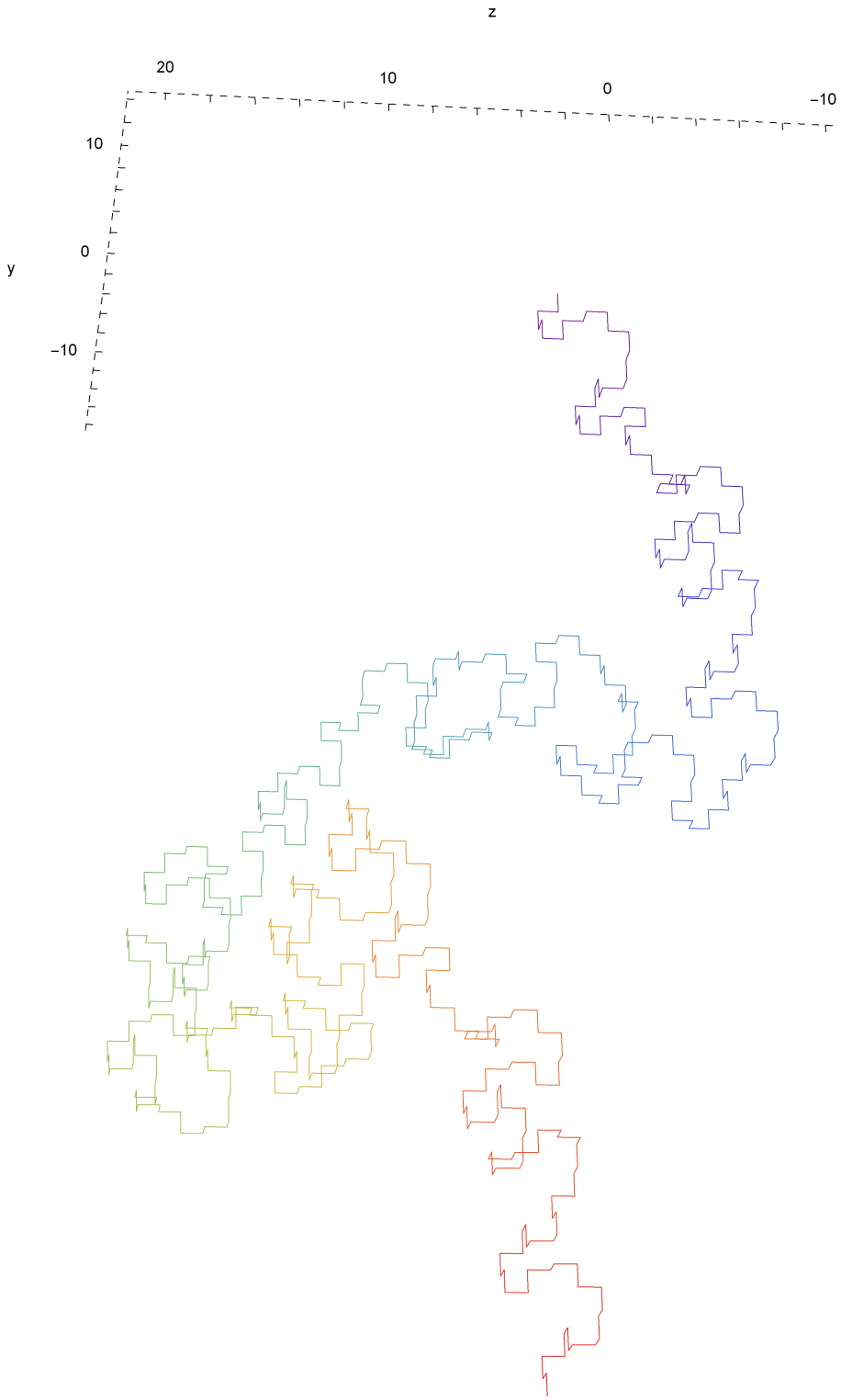
```
 ;

Curve4[initial_, n_, SpecificMirror_] :=
  Module[{old = initial, cur = initial, mirror = SpecificMirror},
```

```
                    For[i = 1, i ≤ n, i++,
                     For[j = 1, j ≤ StringLength[old] + 1, j++,
                          cur = StringInsert[cur,
          Nest[mirror, old, j + 1], (j - 1) * (StringLength[old] + 1) + 1];
                      ];
                     old = cur;
                     ];
                    cur
                ];
```

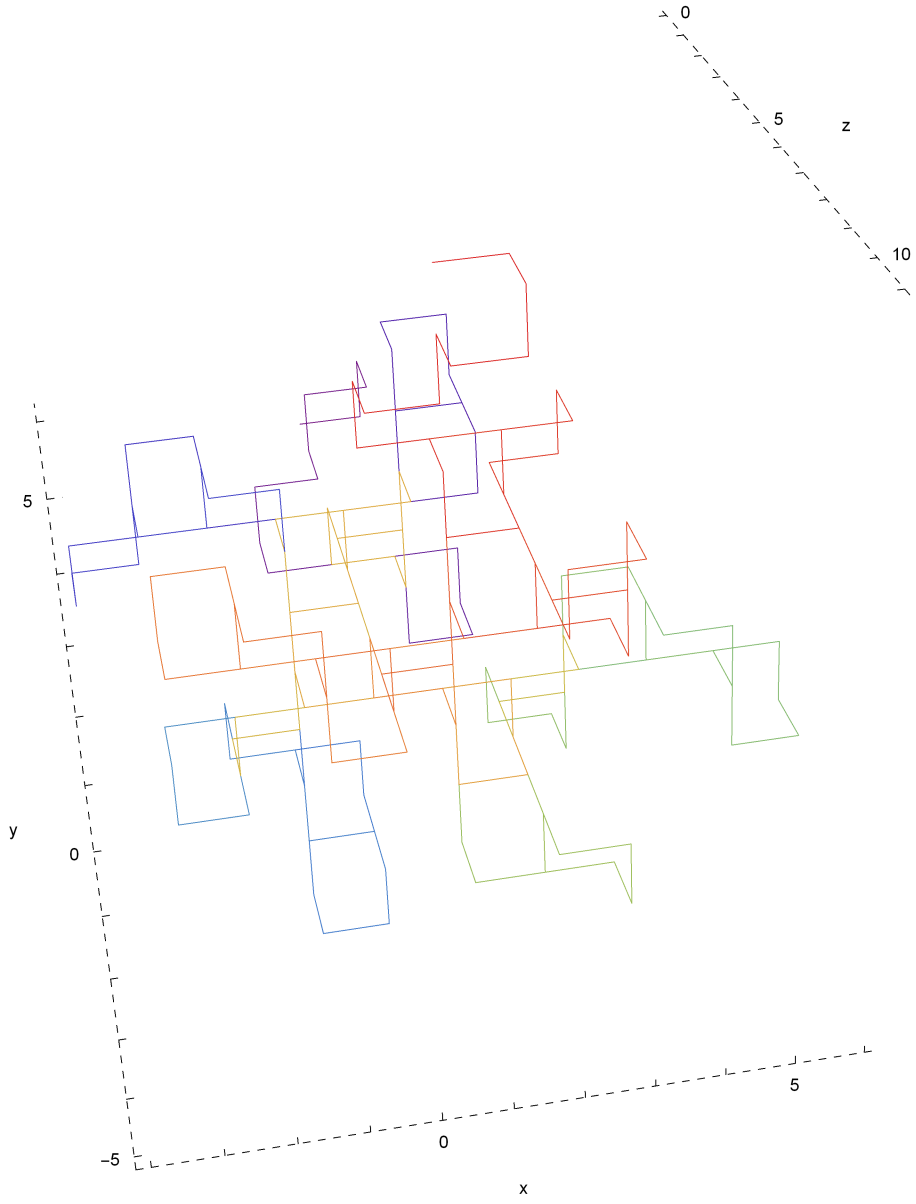
You can now either take a look at the examples below or play around and try out different initial string and mirror combinations and try different curves.

```
 ;
```

```
StringPlot [Curve4["LluU", 3 , ShiftMirror]]
```


z
20
10
0
−10
y
10
0
−10

```
StringPlot[Curve1[8]]
```

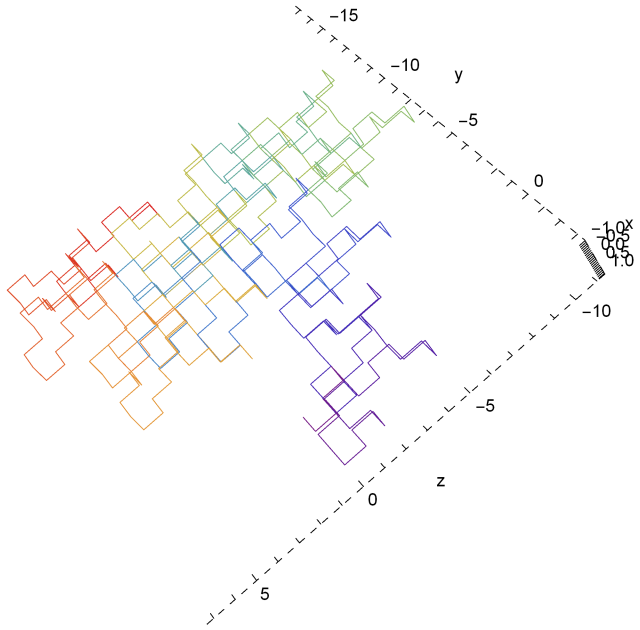


```
mystring = Curve2[10]
```

```
uLuLULuLUlULULuLUlUlulULUlULULuLUlUlulUluLululULUlUlulULUlULULuLUlUlulUluLululUlu↘
   LuLULuluLululULUlUlulUluLululULUlUlulULUlULULuLUlUlulUluLululUluLuLULuluLululU↘
   luLuLULuLUlULULuluLuLULuluLululULUlUlulUluLululUluLuLULuluLululULUlUlulUluLulu↘
   lULUlUlulULUlULULuLUlUlulUluLululUluLuLULuluLululUluLuLULuLUlULULuluLuLULuluLu↘
   lulUluLuLULuLUlULULuLUlUlulULUlULULuluLuLULuLUlULULuluLuLULuluLululULUlUlulUlu↘
   LululUluLuLULuluLululUluLuLULuLUlULULuluLuLULuluLululULUlUlulUluLululUluLuLULu↘
   luLululULUlUlulUluLululULUlUlulULUlULULuLLULUlULUlulUlULUluluLulUlulUlULUluluL↘
   uluLULuLulUluluLulUlulUlULUluluLuluLULuLuluLULUlULuLULuLulUluluLuluLULuLulUlul↘
   uLulUlulUlULUluluLuluLULuLuluLULUlULuLULuLuluLULUlULUlulUlULuLULUlULuLulUl↘
   uluLuluLULuLuluLULUlULuLULuLulUluluLuluLULuLulUlulUlULUluluLuluLULuLul↘
   uLULUlULuLULuLuluLULUlULUlulUlULuLULUlULuLuluLULUlULUlulUlULUluluLulUlulUl↘
   ULuLULUlULUlulUlULuLULUlULuLULuLulUluluLuluLULuLuluLULUlULuLULuLuluLULUlULUlul↘
   UlULuLULUlULuLULuLulUluluLuluLULuLuluLULUlULuLULuLulUluluLuluLULuLulUluluLulUl↘
   ulUlUuLLULUlULUlulUlULUluluLulUlulUlULUluluLuluLULuLulUluluLulUlulUlULUluluLul↘
   uLULuLuluLULUlULuLULuLulUluluLuluLULuLulUlulUlULUluluLuluLULuLuluLULUl↘
   ULuLULuLuluLULUlULUlulUlULuLULUlULuLuluLULuLulUlulUlULUluluLuluLULuLuluLULUlul↘
   uLuluLULuLulUluluLulUlulUlULUluluLuluLULuLuluLULUlULuLULuLuluLULUlULUlulUlULuL↘
   ULUlULuLULuLuluLULUlULUlulUlULUluluLulUlulUlULuLULUlULUlulUlULuLULUlULuLulUl↘
   UluluLuluLULuLuluLULUlULuLULuLuluLULUlULUlulUlULuLULUlULuLULuLulUluluLuluLULuL↘
   uluLULUlULuLULuLulUluluLuluLULuLulUlulUlULUUlUlulUluLululUluLuLULuluLu↘
   lulUluLuLULuLUlULULuluLuLULuluLululUluLuLULuLUlULULuLUlUlulULUlULULuluLuLULuLU↘
   lULULuluLuLULuluLululUluLuLULuLUlULULuLUlUlulULUlULULuLUlUlulUluLululULUlUlulU↘
   LUlULULuluLuLULuLUlULULuLUlUlulULUlULULuluLuLULuLUlULULuluLuLULuluLululUluLuLU↘
   LuLUlULULuLUlUlulULUlULULuLUlUlulUluLululULUlUlulULUlULULuLUlUlulUluLululUluLu↘
   LULuluLululULUlUlulUluLululULUlUlulULUlULULuluLuLULuLUlULULuLUlUlulULUlULULuLU↘
   lUlulUluLululULUlUlulULUlULULuluLuLULuLUlULULuLUlUlulULUlULULuluLuLULuLUlULULu↘
   luLuLULuluLulul
```

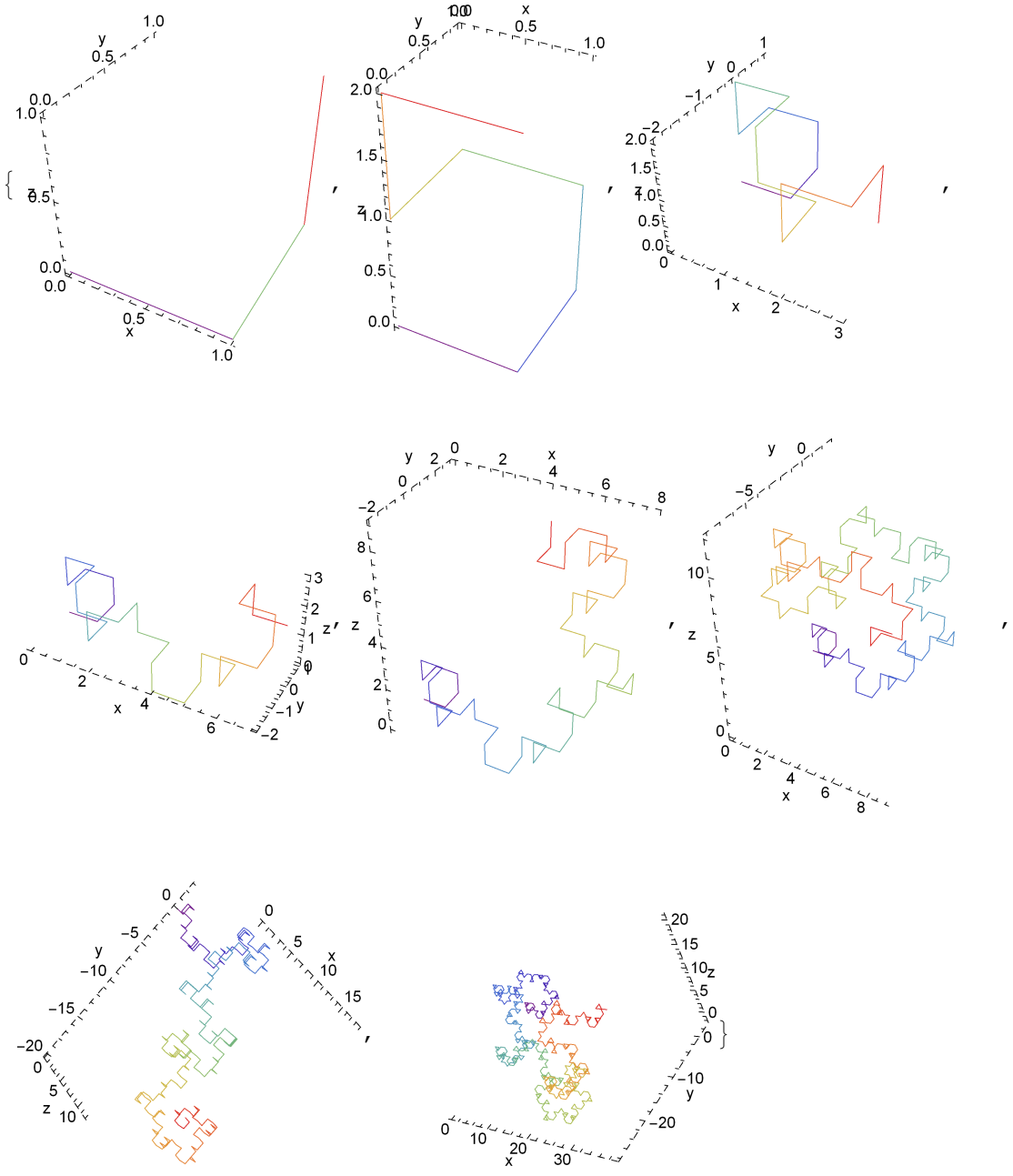
This is an interesting example, because the plot seems to be bounded in one dimension.

```
StringPlot[Curve3[9]](*More than 8 is not possible,
I guess due to the specification of Mathematica in recursion depth*)
```



This one looks at each iteration step like a spiral that is itselfed curled ...

```
Table[StringPlot[Curve2[i]], {i, 1, 8}]
```



As you can see, it is impossible to try out every possible combination, I have played around with it and plotted above what I think are good examples to take a look at. Apparently there is not one curve that deserves to bear the title ‘dragon curve in 3D’, so it’s your turn to try out several variations of fractal strings.