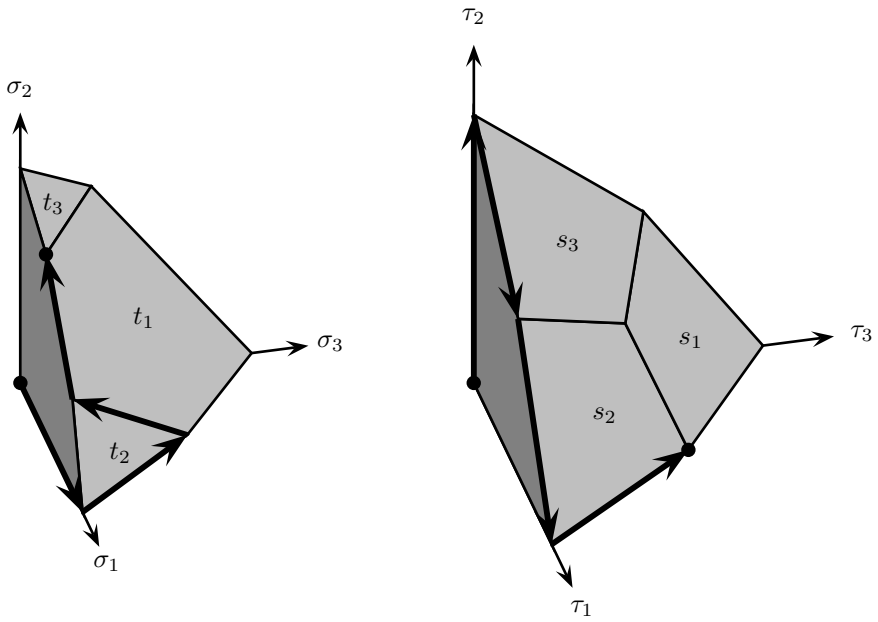
The `ThreeDTricks` Script*

Andrew McLennan
a.mclennan@economics.uq.edu.au

August 31, 2011

Abstract

`ThreeDTricks` is a perl script that allows the user to define basic three dimensional objects (dots, lines, polygons) using three dimensional coordinates, in a format that is a slight extension of pstricks code. The script passes from this to `PSTricks` code in which the three dimensional coordinates have been projected onto a two dimensional surface. This allows the user to achieve realistic perspective.



A Path of the Lemke-Howson Algorithm

*This is the documentation for version 1.0 of the package. This program is released under LPPL.

Contents

1 Introduction **2**

2 How to Use `ThreeDTricks` **3**
2.1 Running `ThreeDTricks` and Viewing the Output 3
2.2 What Happened? 5
2.3 Setting Parameters 6

3 Technical Stuff **8**

4 Bugs **8**

5 Gallery **8**

1 Introduction

`ThreeDTricks` is a tool that allows the user to efficiently use Timothy van Zandt's `PSTricks` package to create simple three dimensional illustrations with accurate perspective. It is, at this point, quite primitive, supporting only a very limited and basic range of objects, but since these are what appear in a large fraction of three dimensional scientific and mathematical illustrations, it should nonetheless prove quite useful. Its design has been guided by simplicity and ease of use. Those in search of elaborate professional tools should probably look elsewhere.

For simple scientific diagrams, "three dimensional graphics" really means the projection of three dimensional objects onto two dimensions. The main difficulty is that for the user, a three dimensional coordinate system is the most natural setting in which to describe the objects, but `PSTricks` only understands the two dimensional plane onto which they are projected. Since development of figures almost always involves extensive trial, error, and tweaking, it is essential that the computations be automated, and that there be a rapid edit-process-reassess cycle.

`ThreeDTricks` is a perl script that provides a calculational engine for computing the coordinates of the projected points. Briefly, the user edits a .tdi (Three D In) file, which looks like `pspicture` code "with features." Provided this file is formatted correctly, `ThreeDTricks` inputs this file, and outputs a .tdo (Three D Out) file, that is the code for a `pspicture`. In one scenario the user is working with a LaTeX file that includes the .tdo file, and can quickly run LaTeX, then use a previewer to look at the output.

2 How to Use ThreeDTricks

2.1 Running ThreeDTricks and Viewing the Output

Let's begin by describing how to work with **ThreeDTricks**. You will be editing a file with a .dti suffix, say `cubegrid.tdi`. It might look something like this:

```
\begin{pspicture}(-7,-5)(11,8)
  PRJT_set_viewpoint(22,16,12)
  PRJT_set_origin(0,0,0)
  PRJT_set_vertical(0,0,2)
  \psline[linewidth=2.2pt,arrowsize=3.0pt 3]{->}%<0,0,0>%%<7,0,0>%
  \psline[linewidth=2.2pt,arrowsize=3.0pt 3]{->}%<0,0,0>%%<0,7,0>%
  \psline[linewidth=2.2pt,arrowsize=3.0pt 3]{->}%<0,0,0>%%<0,0,7>%
  \psline[linewidth=1.0pt]%<0,2,0>%%<6,2,0>%
  \psline[linewidth=1.0pt]%<0,4,0>%%<6,4,0>%
  \psline[linewidth=1.0pt]%<0,6,0>%%<6,6,0>%
                          .
                          .
                          .
  \psline[linewidth=1.0pt]%<2,6,0>%%<2,6,6>%
  \psline[linewidth=1.0pt]%<4,6,0>%%<4,6,6>%
  \psline[linewidth=1.0pt]%<6,6,0>%%<6,6,6>%
\end{pspicture}
```

If you're happy with it, or just want to see how things stand, you run the command line

```
>> threedtricks.plx cubegrid
```

(Note that there is no suffix!) This generates a file `cubegrid.dto` which looks like this:

```
\begin{pspicture}(-7,-5)(11,8)
  \psline[linewidth=2.2pt,arrowsize=3.0pt 3]{->}(0.0000,0.0000)(-4.9858,-2.7669)
  \psline[linewidth=2.2pt,arrowsize=3.0pt 3]{->}(0.0000,0.0000)(6.4825,-1.9028)
  \psline[linewidth=2.2pt,arrowsize=3.0pt 3]{->}(0.0000,0.0000)(0.0000,7.0770)
  \psline[linewidth=1.0pt](1.6782,-0.4926)(-2.3470,-2.9875)
  \psline[linewidth=1.0pt](3.4874,-1.0237)(-0.3779,-3.7364)
  \psline[linewidth=1.0pt](5.4436,-1.5979)(1.7833,-4.5585)
                          .
                          .
                          .
  \psline[linewidth=1.0pt](4.3678,-2.4680)(4.8358,4.4890)
  \psline[linewidth=1.0pt](3.1568,-3.4476)(3.5187,3.8846)
  \psline[linewidth=1.0pt](1.7833,-4.5585)(2.0032,3.1891)
\end{pspicture}
```

Usually you will be working with a LaTeX document which includes a line like

```
\input{cubegrid.tdo}
```

Now you run LaTeX and update your previewer, seeing something like Figure 1 below.

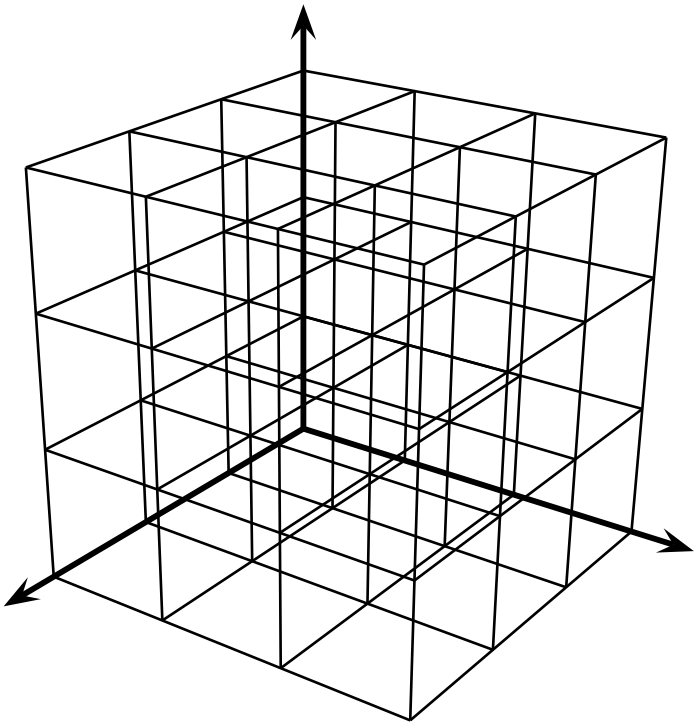


Figure 1

Since you'll be going through this cycle again and again, if you are using Unix you'll probably want to create an alias that combines all these steps into a single command. For example, after I had set up the file `cubegrid.tdi` and gone through the entire cycle once or twice by hand, I would type the following command lines

```
>> alias thr='threedtricks.plx cubegrid'
>> alias ltx='latex cubegrid.tex'
>> alias doall='thr; ltx'
```

After this, when I wanted to see how things were going I could just save `cubegrid.tdi` and run the command line

```
>> doall
```

2.2 What Happened?

In passing from `cubegrid.tdi` to `cubegrid.tdo` two things happened. First, certain parameters were reset by the commands beginning with `PRJT`, and these lines were removed. Second, each three dimensional point like `%<0,0,0>%` was converted to a two dimensional point like `(1.6782,-0.4926)`. The result is a file that `PSTricks` can process. You should be warned that the parsing in `ThreeDTricks` is very primitive, and is quite likely to give bad results (possibly without any warning to the user!) if you try to do anything out of the ordinary. In particular, it is assumed that each `PRJT` command has a line to itself, and that a point like `%<0,0,0>%` is not split between two lines. There is no guarantee of insensitivity to other sorts of white space. If you don't like these restrictions, or prefer delimiters other than `%<·,·,·>%`, you are certainly free to rewrite the perl script to your taste.

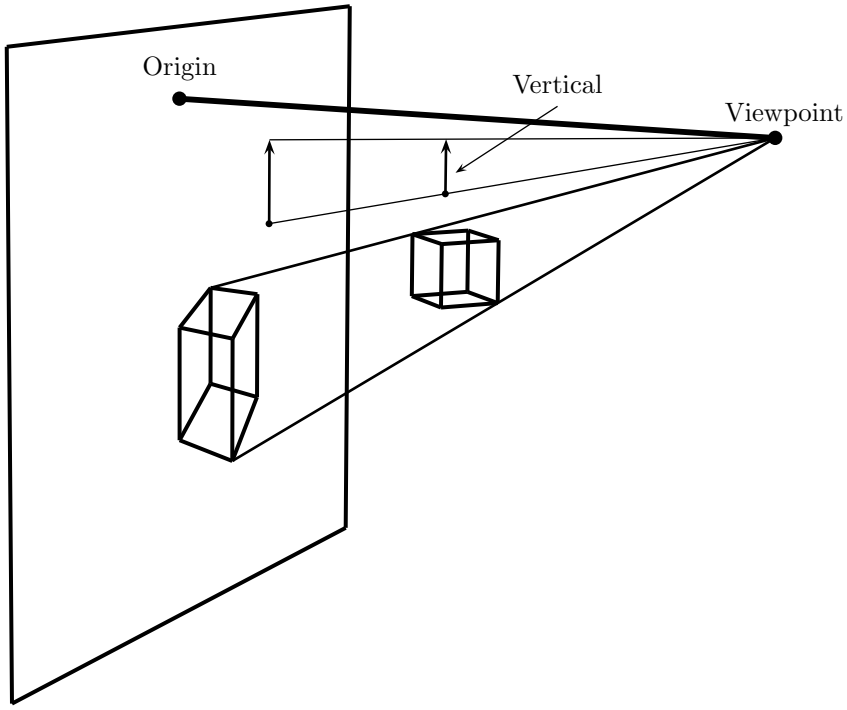


Figure 2

The heart of the process is the conversion of a three dimensional point `%<x,y,z>%` to a two dimensional point `(s,t)`. The parameters include two points in 3-space called the Viewpoint v and the Origin o. The *projection plane*

P is the plane containing o that is perpendicular to the vector $v - o$. The point (x, y, z) is first projected onto the point $f(x, y, z) = (x', y', z')$ where the ray emanating from vp and passing through (x, y, z) intersects P.

The next step is to convert the point (x', y', z') to a point in the two dimensional coordinate system that `PSTricks` understands. One of the parameters is a vector $\nu = (\nu_1, \nu_2, \nu_3)$ that indicates the vertical direction. We set

$$V = \frac{f(\nu) - f(0,0,0)}{\|f(\nu) - f(0,0,0)\|} \quad \text{and} \quad H = \frac{V \times (v - o)}{\|V \times (v - o)\|}.$$

(Here $\times$ is the cross product.) Clearly V and H are orthogonal unit vectors that are parallel to P. There are two more parameters, a scalar m called the *multiplication* and a two dimensional vector τ called the translation. We convert a point $(x', y', z') \in P$ to a point (s, t) in the plane by the formula

$$(s, t) = m \cdot \big(\langle (x', y', z'), H\rangle, \langle (x', y', z'), V\rangle\big) + \tau.$$

In sum, the passage from a three dimensional point to a point in the `PSTricks` coordinate system is

$$(x, y, z) \mapsto m \cdot \big(\langle f(x, y, z), H\rangle, \langle f(x, y, z), V\rangle\big) + \tau.$$

Figure 2 is worth a small remark. The coordinates of the two dimensional image of the cube were computed by applying `ThreeDTricks` to a file with the cube's three dimensional coordinates, setting other parameters as shown in the figure. The projected coordinates were then converted to three dimensional points by adding the appropriate third coordinate, yielding the points that appeared in the .tdi file used to prepare Figure 2.

2.3 Setting Parameters

The commands for setting parameters are quite straightforward. We enumerate them for the sake of easy reference.

- `PRJT_set_viewpoint(`v_1`,`v_2`,`v_3`)` sets the viewpoint to the indicated point. There is a builtin default, but only because I didn't want to bother with programming error messages chastising users who do not set this parameter. You won't like the builtin, so a warning would be superfluous.
- `PRJT_set_origin(`o_1`,`o_2`,`o_3`)` sets the origin to the indicated point. The default is $(0, 0, 0)$.
- `PRJT_set_vertical(`ν_1`,`ν_2`,`ν_3`)` sets the sample vertical vector to the indicated point. The default is $(0, 0, 1)$.
- `PRJT_set_magnification(`m`)` sets the magnification to the indicated number. The default is 1.

- `PRJT_set_translation(`τ_1`,`τ_2`)` sets the translation to the indicated point. The default is $(0, 0)$. It is important to realize that the translation is a vector in the `PSTricks` coordinate system.

- `PRJT_set_stepback(`s`)` sets the "stepback" to the indicated number. The default is 1. The idea is to move the viewpoint closer to or further from the origin, thereby increasing or decreasing the "intensity" of the sense of perspective. Mathematically this amounts to replacing the viewpoint with $o + s(v - o)$.

All of these parameters can change during the process of constructing a figure. This allows for relatively easy handling of repeated elements. Gravity being as pervasive and unfluctuating as it is, for "scientific" purposes there will few occasions when changing the vertical direction is useful, but at least we can have a little fun while illustrating how easy it is to create new effects by replicating existing elements.

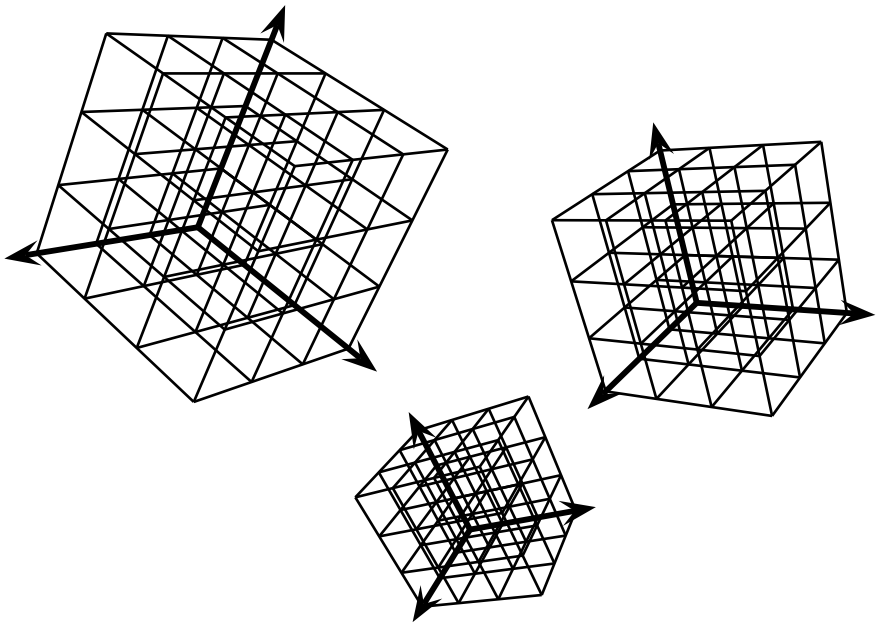


Figure 3

3 Technical Stuff

There is not too much to say here. The underlying perl script is, I hope, fairly easy to understand, at least if you know a little about perl. (I learned most of what I know about the language largely by looking things up as I wrote this, so it is certainly not sophisticated.) Basically it does three things: a) parse; b) compute; c) print the output file. The parsing is crude, and I have put no effort into providing a systematic collection of error messages. The computation is a simple piece of vector geometry. Printing is straightforward, modulo the vagaries of perl's print commands, which are largely adopted from C.

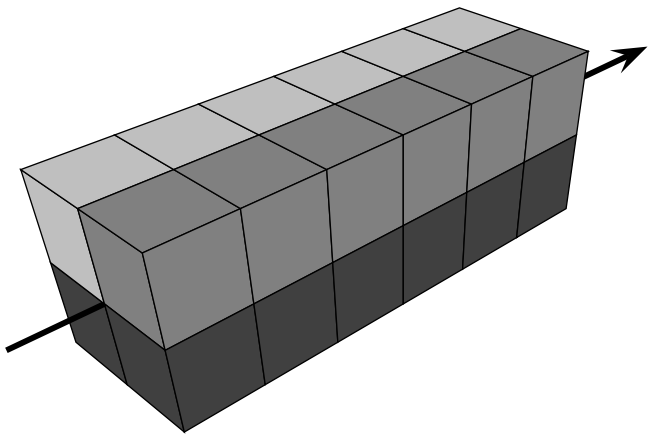
4 Bugs

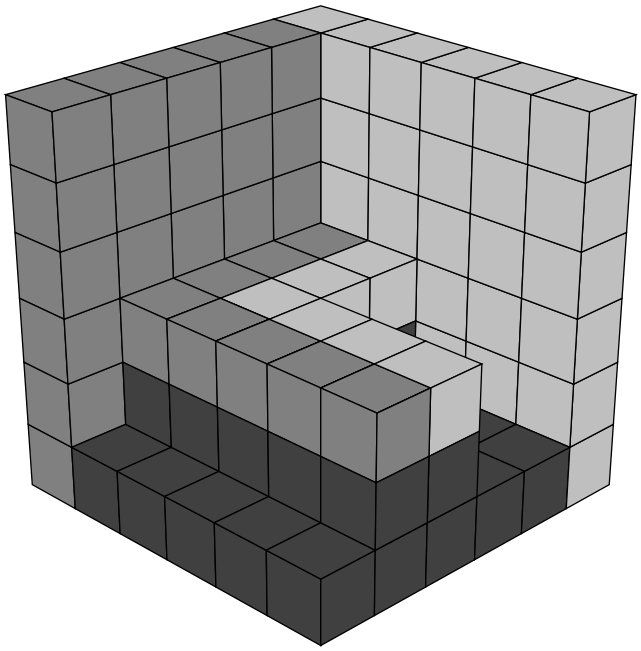
There is no end of additional features and capablities one might desire. First on my list would be the ability to place circles, ellipses, and arcs thereof, in space, but unfortunately `PSTricks` does not support ellipses whose principal axes are not horizontal and vertical. Much more ambitious would be some sort of automated handling of lines that become thinner as they recede into the distance, and more ambitious still would be automated control of shading in certain circumstances. Probably you have your own wish list.

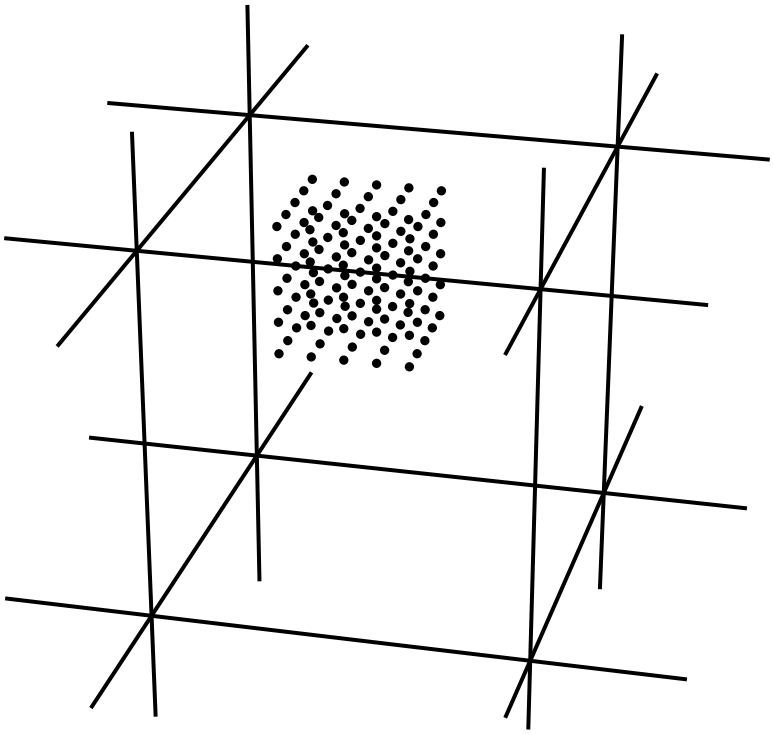
At this point the script is very simple and hopefully easy to understand, which should at least make it relatively easy for the user to modify or enhance it in various directions. In this sense all the bugs add up to one big feature! Please contact me if your work yields an improvement that seems unambiguously useful for a wide audience, and which does not greatly complicate the script or its usage, but be forewarned that esoterica is disparaged.

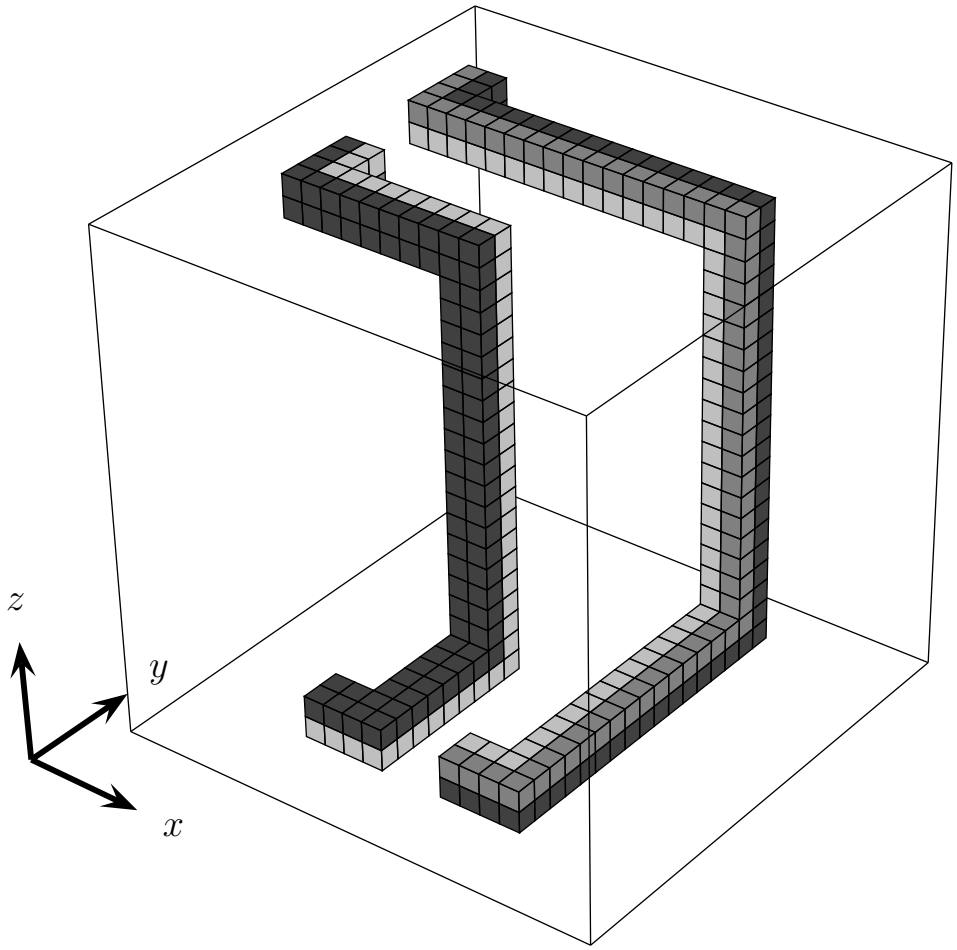
5 Gallery

Just for fun, here are some of the figures from my latest paper.








z
y
x