pyFormex manual

Release 0.3

Tim Neels and Benedict Verhegghe

April 19, 2006

CHAPTER

ONE

Introduction

1.1 What is pyFormex?

You probably expect to find here a short definition of what pyFormex is and what it can do for you. I may have to disappoint you: describing the essence of pyFormex in a few lines is not an easy task to do, because the program can be (and is being) used for very different tasks. So I will give you two answers here: a short one and a long one.

The short answer is that pyFormex is a program to *generate large structured sets of coordinates by means of subsequent mathematical transformations gathered in a script.* If you find this definition too dull, incomprehensible or just not descriptive enough, read on through this section and look at some of the examples in this manual and on the pyFormex website[1]. You will then probably have a better idea of what pyFormex is.

The initial intention of pyFormex (and probably still its main use) was the rapid design of three-dimensional wireframe structures with a configuration that can easier be obtained through mathematical description than through interactive generation of its sub parts and assemblage thereof.

The example of the stent[2] in the figure below illustrates this clearly.

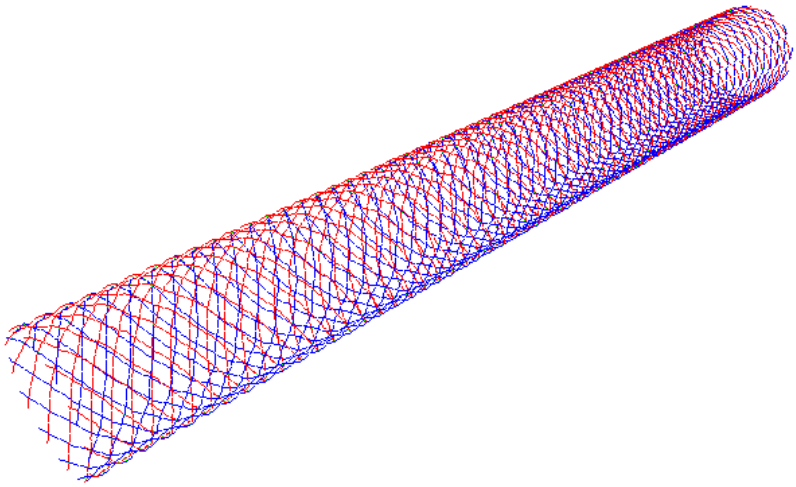


Figure 1.1: WireStent example.

The structure has ...

[1] http://pyformex.berlios.de

[2] A stent is a tube-shaped structure that is e.g. used to reopen (and keep open) obstructed blood vessels.

1.2 History

1.3 Quick tutorial for the pyFormex GUI

In the current version () the GUI mainly serves the following purposes:

- Display a structure in 3D. This includes changing the viewpoint, orientation and viewing distance. Thus you can interactively rotate, translate, zoom.
- Save a view in one of the supported image formats. Most of the images in this manual and on the pyFormex website were created that way.
- Changing pyFormex settings (though there aren't many yet that can be changed through the GUI).
- Running pyFormex scripts, possibly starting other programs and display their results.

The GUI does not (yet) provide a means to interactively design a structure, select parts of a structure or set/show information about (parts of) the structure. Designing a structure is done by writing a small script with the mathematical expressions needed to generate it. Any text editor will be suitable for this purpose. The author uses XEmacs, but this is just a personal preference. A Python aware editor is preferable though, because that is the language used in pyFormex scripts. A pyFormex editor integrated into the GUI remains on our TODO list, but it certainly has not our top priority, because general purpose editors are adequate for most of our purposes.

The best way to learn to use pyFormex is by studying and changing some of the examples. I suggest that you first take a look at the examples included in the pyFormex GUI and select those that display structures that look interesting to you. Then you can study the source code of those examples and see how the structures got built. When starting up, pyFormex reads through the Examples directory (this is normally the 'examples' subdirecty located under the pyformex installation dir). Examples $>$ WireStent

1.4 Quick NumPy tutorial

This could be part of the tutorial in chapter 2

CHAPTER

TWO

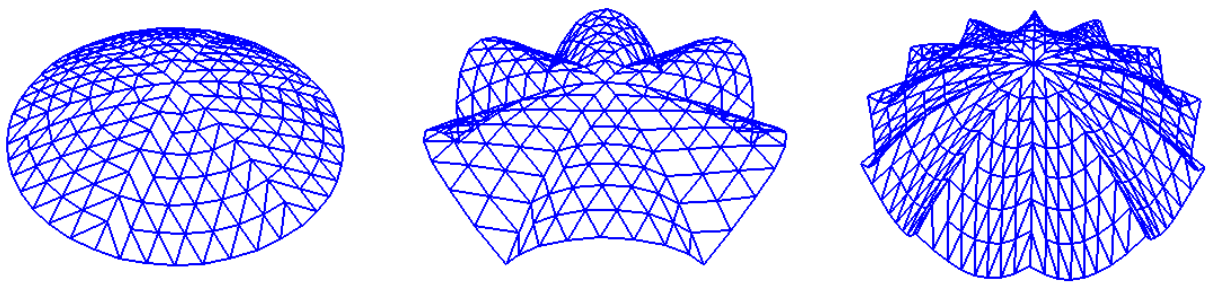
PYFORMEX— a tutorial

2.1 Introduction

pyFormex is a Python implementation of Formex algebra. Using pyFormex, it is very easy to generate large geometrical models of 3D structures by a sequence of mathematical transformations. It is especially suited for the automated design of spatial frame structures. But it can also be used for other tasks, like finite element preprocessing, or just for creating some nice pictures.

By writing a very simple script, a large geometry can be created by copying, translating, rotating,... Formices. pyFormex will interpret this script and draw what you have created. This is clearly very different than the traditional way of creating a model, like CAD. There are two huge advantages about using pyFormex

- It is especially suited for the automated design of spatial frame structures. A dome, arc, hypar,... can be rather difficult to draw with CAD, but when using mathematical transformations, it becomes a piece of cake!
- Using a script makes it very easy to apply changes in the geometry: you simply modify the script and let pyFormex play it again. For instance, you can easily change an angle, the radius of a dome, the ratio f/l of an arc,... Using CAD, you would have to make an entirely new drawing! This is also illustrated in fig 2.1: these domes where all created with the same script, but with other values of the parameters.



(a) A basic Scallopdome (b) Another Scallopdome (c) Yet another Scallopdome

Figure 2.1: Same script, different domes

As mentioned, pyFormex is based on the programming language Python [1]. This implies that the scripts are also Python-based. It's a very easy language, but if you're interested in reading more, there is a very good tutorial available on `http://docs.python.org/tut/`. However, if you're only using Python to write pyFormex-scripts, the tutorial you're reading right now should be enough.

[1] `http://www.python.org`

2.2 Getting started

This section holds some basic information on how to use Python and pyFormex.

- Each script should begin with `#!/usr/bin/env pyformex`
- To open pyFormex, double click on the file 'pyformex' in the installation directory, or type *pyformex* in the terminal. Using the terminal can be very useful, because errors that are created while running the script will appear in the terminal. This can provide you with useful information when something goes wrong with your script.
- To create a new pyFormex-script, just open a new file with your favorite text-editor and save it as 'myproject.py'.
- To edit a script, you can
  - Open it with your favorite text-editor
  - File > Open
    At this point, the script will be loaded but nothing will happen.
    File > Edit
    The script will now open in the default text-editor. This default editor can be changed in the '.pyformexrc' file in the installation directory. This way of editing your script makes it possible to immediately draw changes you applied in the script (without reopening the file).
    File > Play
- To play a script, you can
  - File > Open
    File > Play
  - Type *pyformex myproject.py* in the terminal. This will open pyFormex and play your script at the same time.
  - To play a script without using the GUI (for example in finite element preprocessing, if you only want to write an output file, without drawing the structure), type *pyformex –nogui myproject.py*
- When writing a script in Python, there are some things you should keep in mind:
  - When using a function that requires arguments, an argument list must have any positional arguments followed by any keyword arguments, where the keywords must be chosen from the formal parameter names. It's not important whether a formal parameter has a default value or not. No argument may receive a value more than once – formal parameter names corresponding to positional arguments cannot be used as keywords in the same calls. Simply put: you can either set the arguments in the right order and only give their value, or you can give arguments by their name and value. This last option holds some advantages: not only is it easier to check what you did, but sometimes a function has many arguments with default values and you only want to change a few. If this isn't entirely clear yet, just look at the examples later in this tutorial or check the Python tutorial.
  - Indentation is essential in Python. Indentation is Python's way of grouping statements. In straight-forward scripts, indentation is not needed (and forbidden!), but when you use an if-statement for example, the body of the statement has to be indented. A small example might make this clear. Also notice the ':'

    ```
    print 'properties'
    for key, item in properties.iteritems():
        print key, item
    ```

  - If you want to use functions from a seperate module (like `properties`), you add a line on top of the script

```
from properties import *
```

All functions from that module are now available.

- The hash character, ”#”, is used to start a comment in Python.
- Python is case sensative.

2.3 Geometric model

2.3.1 Creating a Formex

What is a Formex

A Formex is a numarray of order 3 (axes 0,1,2) and type Float. A scalar element represents a coordinate (F:uniple).

A row along the axis 2 is a set of coordinates and represents a point (node, vertex, F: signet). For simplicity’s sake, the current implementation only deals with points in a 3-dimensional space. This means that the length of axis 2 is always 3. The user can create Formices (plural of Formex) in a 2-D space, but internally these will be stored with 3 coordinates, by adding a third value 0. All operations work with 3-D coordinate sets. However, a method exists to extract only a limited set of coordinates from the results, permitting to return to a 2-D environment.

A plane along the axes 2 and 1 is a set of points (F: cantle). This can be thought of as a geometrical shape (2 points form a line segment, 3 points make a triangle, ...) or as an element in FE terms. But it really is up to the user as to how this set of points is to be interpreted.

Finally, the whole Formex represents a set of such elements.

Additionally, a Formex may have a property set, which is an 1-D array of integers. The length of the array is equal to the length of axis 0 of the Formex data (i.e. the number of elements in the Formex). Thus, a single integer value may be attributed to each element. It is up to the user to define the use of this integer (e.g. it could be an index in a table of element property records). If a property set is defined, it will be copied together with the Formex data whenever copies of the Formex (or parts thereof) are made. Properties can be specified at creation time, and they can be set, modified or deleted at any time. Of course, the properties that are copied in an operation are those that exist at the time of performing the operation.

Simply put: a Formex is a set of elements, and every element can have a property number.

Creating a Formex using coordinates

The first and most useful way to create a Formex is by specifying it’s nodes and elements in a 3D-list.

```
F=Formex([[[0,0],[1,0],[1,1],[0,1]]])
```

This would create a Formex F, which has the nodes (0,0), (0,1), (1,1) and (0,1). These nodes are all part of a single element, thus creating a square plane. This element is also the entire Formex. On the other hand, if you would change the position of the square brackets like in the following example, then you’d create a Formex F which is different from the previous. The nodes are the same, but the connection is different. The nodes (0,0) and (1,0) are linked together by an element, and so are the nodes (1,1) and (0,1). The Formex is now a set of 2 parallel bars, instead of a single square plane.

```
F=Formex([[[0,0],[1,0]],[[1,1],[0,1]]])
```

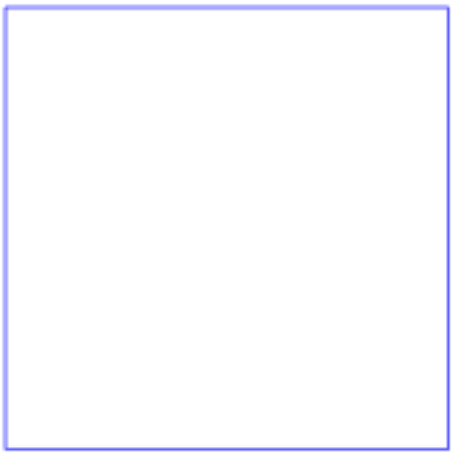


Figure 2.2: A very simple Formex

Figure 2.3: Same nodes, different Formex

If we want to define a Formex, similar to the square plane, but consisting of the 4 edges instead of the actual plane, we have to define four elements and combine them in a Formex. This is *not* the same Formex as fig 2.2, although it looks exactly the same.

```
F=Formex([[[0,0],[0,1]], [[0,1],[1,1]], [[1,1],[1,0]], [[1,0],[0,0]]])
```

The previous examples were limited to a 2-D environment for simplicity. Of course, we could add a third dimension. For instance, it's no problem defining a pyramid consisting of 8 elements ('bars').

```
F=Formex([[[0,0,0],[0,1,0]], [[0,1,0],[1,1,0]], [[1,1,0],[1,0,0]], [[1,0,0],
[0,0,0]], [[0,0,0],[0,1,0]], [[0,0,0],[0.5,0.5,1]], [[1,0,0],[0.5,0.5,1]],
[[1,1,0], [0.5,0.5,1]], [[0,1,0],[0.5,0.5,1]]])
```

However, as you can see, even in this very small example the number of nodes, elements and coordinates you have to declare becomes rather large. Defining large Formices using this method would not be practical. This problem is easily overcome by copying, translating, rotating,... a smaller Formex - as will be explained in the rest of this chapter

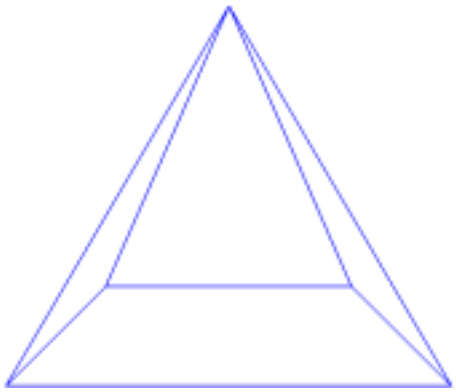


Figure 2.4: Pyramid

- or by using patterns.

Creating a Formex using patterns

The second way of creating a new Formex, is by defining patterns. In this case, a line segment pattern is created from a string.

This function creates a list of line segments where all nodes lie on the gridpoints of a regular grid with unit step. The first point of the list is [0,0,0]. Each character from the given string is interpreted as a code specifying how to move to the next node. Currently defined are the following codes: 0 = goto origin [0,0,0]
1..8 move in the x,y plane
9 remains at the same place
When looking at the plane with the x-axis to the right, 1 = East, 2 = North, 3 = West, 4 = South, 5 = NE, 6 = NW, 7 = SW, 8 = SE. Adding 16 to the ordinal of the character causes an extra move of +1 in the z-direction. Adding 48 causes an extra move of -1. This means that 'ABCDEFGHI', resp. 'abcdefghi', correspond with '123456789' with an extra z +/-= 1. The special character 'ć can be put before any character to make the move without making a connection. The effect of any other character is undefined.

This method has important restrictions, since it can only created lines, in a 3-D environment, on a regular grid. However, it can be a much easier and shorter way to define a simple Formex. This is illustrated by the difference in length between code 3 and code ref 4, although they define the same Formex.

```
F=Formex(pattern('1234'))
```

Some simple patterns are defined in `simple.py` and are ready for use. These patterns are stacked in a dictionary called 'Patterns'. Items of this dictionary can be accessed like `Patterns['cube']`.

```
#!/usr/bin/env pyformex
from simple import *
c=Formex(pattern(Pattern['cube']))
draw(c)
```

Creating a Formex using coordinates from a file

In some cases, you might want to read coordinates from a file an combine them into a Formex. This is possible with the module `file2formex` and it's function `fileFormex()`. Each point is connected to the following, forming an

element (bar). The next file ('square.txt') would create the same Formex as the previous example.

```
0,0,0
0,1,0
1,1,0
1,0,0
```

```
#!/usr/bin/env pyformex
from file2formex import *
F=fileFormex('square.text', closed='yes')
```

2.3.2 Adding property numbers

Each Formex element can have a property number. Each property number is represented by another color when the Formex is drawn. This is the first reason why you could use property numbers: to make your drawing more transparent or just more beautiful. However, these numbers can also be used as an entry in a dictionary of properties - thus linking the element with a property. This property can be about anything, but in finite element processing this would be the element section, material, loads,... The use of properties in this way will be futher explained in 2.4. Property numbers can be specified at creation time, and they can be set, modified or deleted at any time.

```
>>> #!/usr/bin/env pyformex
>>> F=Formex(pattern('1234'),[5])
>>> print F.prop()
>>> G=Formex(pattern('1234'),[6,8,2,4])
>>> print G.prop()
>>> F.setProp([6,7])
>>> print F.prop()
>>> G.setProp([6,7,8,9])
>>> print G.prop()

[5 5 5 5]
[6 8 2 4]
[6 7 6 7]
[6 7 8 9]
```

2.3.3 Drawing a Formex

Of course, you'd want to see what you have created. This is accomplished by the `draw()` function.

```
F=Formex([[[0,0,0],[0,1,0]], [[0,1,0],[1,1,0]], [[1,1,0],[1,0,0]], [[1,0,0],
[0,0,0]], [[0,0,0],[0,1,0]], [[0,0,0],[0.5,0.5,1]], [[1,0,0],[0.5,0.5,1]],
[[1,1,0], [0.5,0.5,1]], [[0,1,0],[0.5,0.5,1]]])
draw(F)
```

It also possible to draw multiple Formices at the same time.

```
F=Formex([[[0,0,0],[0,1,0]], [[0,1,0],[1,1,0]], [[1,1,0],[1,0,0]], [[1,0,0],
[0,0,0]], [[0,0,0],[0,1,0]], [[0,0,0],[0.5,0.5,1]], [[1,0,0],[0.5,0.5,1]],
[[1,1,0],[0.5,0.5,1]], [[0,1,0],[0.5,0.5,1]]])
G=Formex(pattern('1234'))
draw(F+G)
```

It might be important to realize that even if you don't draw a particular Formex, that doesn't mean you didn't create it!

Now, when you are creating a large geometry, you might be interested in seeing the different steps in the creation. To remove all previously drawn Formices, you can use `clear()` what sweepes the screen clean. If you want to see a certain step in the creation longer than the default time, use `sleep(t)`, with 't' the delay (in seconds) before executing the next command.

```
F=Formex(pattern('164'))
draw(F)
G=F.replic(5,1,0)
clear()
draw(G)
```

2.3.4 Information about a Formex

There are a number of functions available that return information about a Formex. Especially when using pyFormex as finit element preprocessor, the most useful functions are:

Function	**Description**
`F.nelems()`	Return the number of elements in the formex.
`F.nnodes()`	Return the number of nodes in the formex.
`F.prop()`	Return the properties as a numpy array.
`F.bbox()`	Return the bounding box of the Formex.
`F.center()`	Return the center of the Formex.
`F.nodesAndElements()`	Return a tuple of nodal coordinates and element connectivity.

`nodesAndElements` is very important in finite element processing. It returns all nodes and all elements of the Formex in a format useful for FE processing. A tuple of two arrays is returned. The first is float array with the coordinates of the unique nodes of the Formex. The second is an integer array with the node numbers connected by each element.

```
>>> #!/usr/bin/env pyformex
>>> from simple import *

>>> c = Formex(pattern(Pattern['cube']))
>>> draw(c)
>>> nodes,elems = c.nodesAndElements()
>>> print 'Nodes'
>>> print nodes
>>> print 'Elements'
>>> print elems

Nodes
[[ 0.  0. -1.]
 [ 1.  0. -1.]
 [ 0.  1. -1.]
 [ 1.  1. -1.]
 [ 0.  0.  0.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 1.  1.  0.]]
Elements
[[4 5]
 [5 7]
 [7 6]
 [6 4]
 [4 0]
 [5 1]
 [7 3]
 [6 2]
 [0 1]
 [1 3]
 [3 2]
 [2 0]]
```

2.3.5 Changing the Formex

Untill now, we've only created simple Formices. The strength of pyFormex however is that it is very easy to generate large geometrical models by a sequence of mathematical transformations. After initiating a basic Formex, it's possible to transform it by using copies, translations, rotations, projections,...

There are many transformations available, but this is not the right place to describe them all. This is what the reference manual in chapter 3 is for. A summary of all possible transformations and functions can be found there.

To illustrate some of these transformations and the recommended way of writing a script, we will analyse some of the examples. More of these interesting examples are found in 'installdir/examples'. Let's begin with the example 'Spiral.py'.

```
#!/usr/bin/env pyformex
# $Id$
##
## This file is part of pyFormex 0.3 Release Mon Feb 20 21:04:03 2006
## pyFormex is a python implementation of Formex algebra
## Homepage: http://pyformex.berlios.de/
## Distributed under the GNU General Public License, see file COPYING
## Copyright (C) Benedict Verhegghe except where stated otherwise
```

```
##
#
"""Spiral"""

m = 36 # number of cells along torus big circle
n = 10 # number of cells along torus small circle

def drawit(F,view='front'):
    clear()
    draw(F,view)

F = Formex(pattern("164"),[1,2,3]); drawit(F)
F = F.replic(m,1,0); drawit(F)
F = F.replic(n,1,1); drawit(F)
F = F.translate1(2,1); drawit(F,'iso')
F = F.cylindrical([2,1,0],[1.,360./n,1.]); drawit(F,'iso')
F = F.replic(5,m,2); drawit(F,'iso')
F = F.rotate(-10,0); drawit(F,'iso')
F = F.translate1(0,5); drawit(F,'iso')
F = F.cylindrical([0,2,1],[1.,360./m,1.]); drawit(F,'iso')
drawit(F,'right')
```

During this first read-through, you will have noticed that every step is drawn. Of course, this is not necessary but can be useful. And above all, it is very educational for use in a tutorial...

The next important thing is that parameters were used. It's recommended to always do this, especially when you want to do a parametric study of course, but it can also be very convenient if at some point you want to change the geometry (for example when you want to re-use the script for another application).

A simple function `drawit()` is defined for use in this script only. This function only provides a shorter way of drawing Formices, since it combines `clear()` and `draw`.

Now, let's dissect the script.

```
#!/usr/bin/env pyformex
"""Spiral"""

#This is a small function that is only defined in this script. It clears the
#screen and draws the Formex at the same time.
def drawit(F,view='front'):
    clear()
    draw(F,view)

#These are the parameters. They can easily be changed, and a whole new
#spiral will be created without any extra effort.
m = 36 # number of cells along torus big circle
n = 10 # number of cells along torus small circle

#The first step is to create a basic Formex. In this case, it's a triangle which
#has a different property number for every edge.
F = Formex(pattern("164"),[1,2,3]); drawit(F)

#This basic Formex is copied 'm' times in the 0-direction with a translation
#step of '1' (the length of an edge of the triangle). After that, the new
#Formex is copied 'n' times in the 1-direction with a translation step of '1'.
#Because of the recursive definition (F=F.replic), the original Formex F is
#overwritten by the transformed one.
F = F.replic(m,1,0); drawit(F)
F = F.replic(n,1,1); drawit(F)
```

```
#Now a copy of this last Formex is translated in direction ’2’ with a
#translation step of ’1’. This necessary for the transformation into a cilinder.
#The result of all previous steps is a rectangular pattern with the desired
#dimensions, in a plane z=1.
F = F.translate1(2,1); drawit(F,’iso’)

#This pattern is rolled up into a cilinder around the 2-axis.
F = F.cylindrical([2,1,0],[1.,360./n,1.]); drawit(F,’iso’)

#This cilinder is copied 5 times in the 2-direction with a translation step of
#’m’ (the lenght of the cilinder).
F = F.replic(5,m,2); drawit(F,’iso’)

#The next step is to rotate this cilinder -10 degrees around the 0-axis.
#This will determine the pitch angle of the spiral.
F = F.rotate(-10,0); drawit(F,’iso’)

#A copy of this last formex is now translated in direction ’0’ with a
#translation step of ’5’.
F = F.translate1(0,5); drawit(F,’iso’)

#Finally, the Formex is rolled up, but around a different axis then before.
#Due to the pitch angle, a spiral is created. If the pitch angle would be 0
#(no rotation of -10 degrees around the 0-axis), the resulting Formex
#would be a torus.
F = F.cylindrical([0,2,1],[1.,360./m,1.]); drawit(F,’iso’)
drawit(F,’right’)
```

2.4 Adding properties

2.5 Export to finite-elements program

2.6 Tips and Tricks

Some of these tips are already mentioned in the previous sections, but.....

- Starting pyFormex using the terminal can be very useful. Errors that are created while running the script will appear in the terminal, you can use the ’print’-function to check a result,... This can be a very useful when controlling your script
- nog wat hints

CHAPTER

THREE

PYFORMEX— reference manual

3.1 formex — the base module

This module contains all the basic functionality for creating, structuring and transforming sets of coordinates.

class `Formex`(*data=[[[]]],prop=None*)

A class to hold a structured set of coordinates. A `Formex` is a three dimensional array of float values. The array has a shape `(nelems,nnodel,3)`. Each slice `[i,j]` of the array contains the three coordinates of a point in space. We will also call this a *node*. Each slice `[i]` of the array contains a connected set of nnodel points: we will refer to it as an *element*.

It is up to the user on how to interprete this connection: two connected nodes will usually represent a line segment between these two points. An element with three nodes however could just as well be interpreted as a triangle or as a (possibly curved) line. And if it is a triangle, it could be either the circumference of the triangle or the part of the plane inside that circumference. As far as the Formex class concerns, each element is just a set of points.

All elements in a `Formex` must have the same number of points, but you can construct `Formex` instances with any (positive) number of nodes per element. When `nnodel==1`, the `Formex` contains only unconnected nodes (each element is just one point).

One way of attaching other data to the `Formex`, is by the use of the 'property' attribute. The property is an array holding one integer value for each of the elements of the Formex. The use of this property value is completely defined by the user. It could be a code for the type of element, or for the color to draw this element with. Most often it will be used as an index into some other (possibly complex) data structure holding all the characteristics of that element.

By including this property index into the Formex class, we make sure that when new elements are constructed from existing ones, the element properties are automatically propagated.

`f`

A three dimensional array of float values. The array has a shape (nelems,nnodel,3). Each slice [i,j] of the array contains the three coordinates of a point in space. We will also call this a *node*. Each slice [i] of the array contains a connected set of nnodel points: we will refer to it as an *element*. It is up to the user on how to interprete this connection: two connected nodes will usually represent a line segment between these two points. An element with three nodes however could just as well be interpreted as a triangle or as a (possibly curved) line. And if it is a triangle, it could be either the circumference of the triangle or the part of the plane inside that circumference.