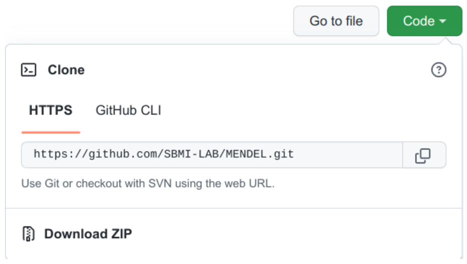
Quick start guide for MENDEL

October 25, 2022

1 Installation using Blender

The Blender interface of MENDEL provides convenient workflow to design DNA origami nanostructures. MENDEL takes advantage of the Python environment and 3D engine provided with Blender. To install blender go to the url: https://www.blender.org/download/ and download the version compatible with the user's operating system. MENDEL is available in github: https://github.com/SBMI-LAB/MENDEL.



https://github.com/SBMI-LAB/MENDEL.git

Use Git or checkout with SVN using the web URL.

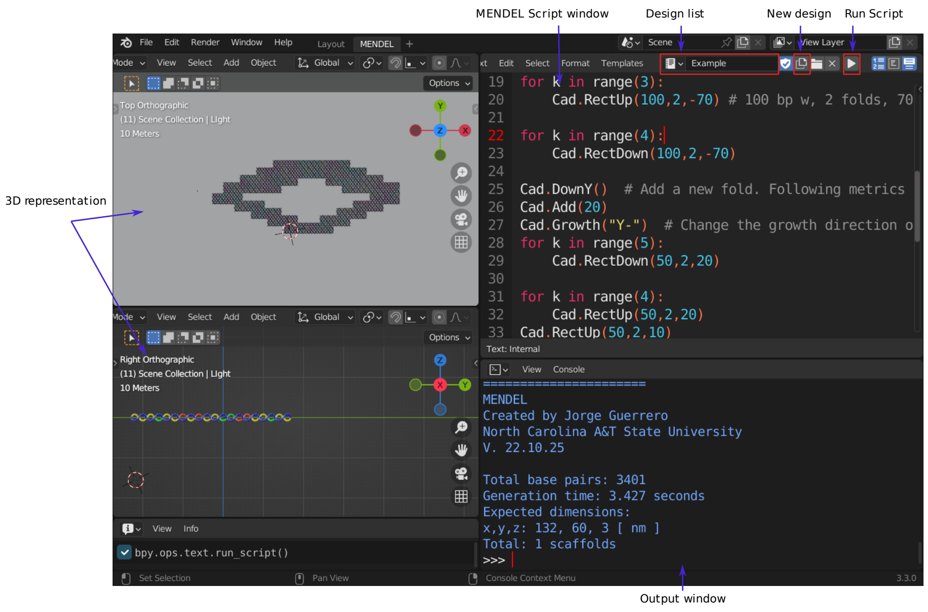
Figure 1: GitHub download page

The code can be easily downloaded as a zip file and uncompressed in the local computer, as shown in Figure 1. The folder contains the source files (.py format) and one Blender sample file (BlenderMendel.blend).

To start working with MENDEL, open the example file BlenderMendel.blend using Blender.

2 Blender Interface

The BlenderMendel.blend file is a completely functional example to work with MENDEL. Figure 2 shows the Blender interface for MENDEL.



```
for k in range(3):
    Cad.RectUp(100,2,-70) # 100 bp w, 2 folds, 70

for k in range(4):
    Cad.RectDown(100,2,-70)

Cad.DownY()  # Add a new fold. Following metrics
Cad.Add(20)
Cad.Growth("Y-")  # Change the growth direction o
for k in range(5):
    Cad.RectDown(50,2,20)

for k in range(4):
    Cad.RectUp(50,2,20)
Cad.RectUp(50,2,10)
```

```
=====================
MENDEL
Created by Jorge Guerrero
North Carolina A&T State University
V. 22.10.25

Total base pairs: 3401
Generation time: 3.427 seconds
Expected dimensions:
x,y,z: 132, 60, 3 [ nm ]
Total: 1 scaffolds
>>>
```

```
bpy.ops.text.run_script()
```

Figure 2: Blender interface for MENDEL

By using the Blender interface, the user has access to the Python scripting engine and execute the commands provided by MENDEL.

The included example creates a basic shape (as shown in Figure 2). To visualize the results of the MENDEL commands, it will required just to click on the Run Script button, or press the combination Alt + P when the mouse cursor is located in the Script Window.

If the code is correct, the output window will show statistics of the designed structure, and the 3D windows will render the geometry of the design.

It is recommended that the user make a copy of the BlenderMendel example file in the same folder.

3 Design workflow

MENDEL supports integration in Blender, and Blender files can store several scripts inside and incorporates a Python engine, which makes Blender the ideal platform for this task. We have to locate the Blender file in the same folder where the file Mendel.py is. In the scripting interface of Blender, the script files should contain the following lines to incorporate Mendel into the file:

```
import bpy
filepath = bpy.path.abspath("//Mendel.py")
exec(compile(open(filepath).read(), filepath, 'exec'))
```

To start the design, we need to create a MENDEL object:

```
Cad = Mendel()
```

Here, Cad is just the object name and can be changed, but I use the same variable name for all examples.

To add new nucleotides to the scaffold, we use the instruction *Add*:

```
Cad.Add(200)
```

The parameter is the number of nucleotides to add.

To create a new fold, we can use the commands *Down* or *Up* followed by the coordinates Y or Z:

```
Cad.DownY()
Cad.DownZ()
Cad.UpY()
Cad.UpZ()
```

Anytime we create a fold, MENDEL will compute the orientation of the nucleotides of the scaffold, and when the next is correctly aligned, MENDEL will create the corresponding fold.

To create rectangles, MENDEL provides the command *RectUp* and *RectDown*:

```
Cad.RectUp(80,4)
Cad.RectDown(80,4)
```

These commands differ in the way how to create the rectangle. Regarding the coordinates we are growing, the next fold will be *UpY* or *UpZ* for *RectUp*, and *DownY* or *DownZ* for *RectDown*. The first parameter is the number of nucleotides to add before creating a new fold. The second parameter is the number of folds to add to the design. The command RectUp(80,4) will generate a rectangle of at least 80 nucleotides width and 4 new helices; counting the current one will be 5 helices.

For the design of large structures, the rectangle is a handy command, and the number of folds of the rectangle is decisive for defining the sense of the scaffold. Using even folds will keep the direction of the scaffold, and odd folds will invert it, as shown in Figure 3.

The Rect commands use the parameters specified by *Growth*, which define the sense of growth of the rectangles, for instance:

```
Cad.Growth(Y+)
```

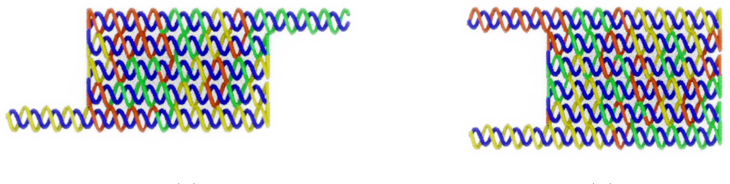

(a) (b)

Figure 3: Design differences when using a) 4 folds per rectangle and b) 5 folds per rectangle. The scaffold starts in the lower-left corner and grows to the right.

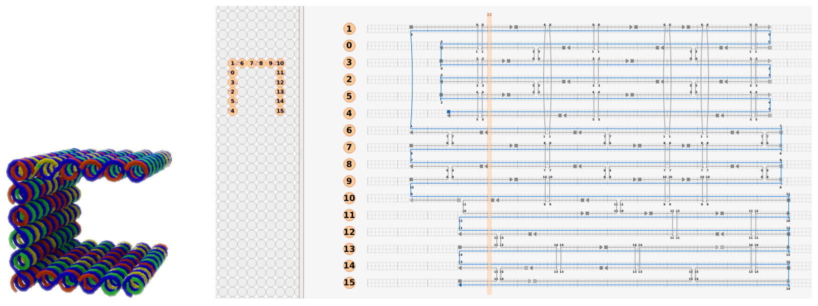

(a) (b)

Figure 4: Structure designed by modifying the Growth parameters. a) 3D render generated in Blender, b) Cadnano output file.

The rectangle will grow in Y, and the calculations will consider the current location as the left corner. Changing the parameter to "Y-" will consider the location as the right corner. The parameter can also be "Z+" and "Z-" to make the Rectangles grow for the Z coordinates. The Growth command is not required for every Rectangle, only when we modify their direction. By default, it is "Y+".

Figure 4 shows a more elaborate example. Here, we add three rectangles of 80 nucleotides wide and 5 helices. Using the Growth parameter, we change the growth direction of the next rectangle:

```
Cad.RectUp(80,5)
Cad.Growth(Z+)
Cad.RectUp(80,5)
Cad.Growth(Y-)
Cad.RectDown(80,5)
```

And the resulting design is a C-shaped 3D structure with 1487 bp and took 0.3 seconds to generate. Figure 4 shows the resulting structure and the output Cadnano file. Table 1 contains the completed set of commands for MENDEL.

Table 1: Commands provided with MENDEL

Command	**Description**
Cad = Mendel()	Create a new Mendel object named Cad. Use the object Cad to apply all the other commands.
Cad.StartAt(position)	Define the starting location along with the X coordinates. This command gives compatibility to define the initial position in a caDNAno design. The parameter position is an integer value, and by default, its value is 0.
Cad.AddAt(x,y,z)	Adds a new nucleotide at the exact location (x,y,z). The new nucleotide will start a new DNA scaffold.
Cad.Add(number)	Adds the selected number of nucleotides to the current DNA scaffold.
Cad.UpY()	Create a turn-up in Y coordinates.
Cad.DownY()	Create a turn-down in Y coordinates.
Cad.UpZ()	Create a turn-up in Z coordinates.
Cad.DownZ()	Create a turn-down in Z coordinates.
Cad.Growth(orientation)	It sets the growing orientation for the rectangles. The options are: "Y+", "Y-", "Z+", and "Z-".
Cad.RectUp(width, height)	Create a rectangle where the width is in nucleotides and the height is in helices. The turns of the scaffold are Up in the Y coordinates.
Cad.RectDown(width,height)	Create a rectangle that grows down in the Y coordinates.
Cad.GotoX(location)	Grow the scaffold to reach the X coordinate marked by the parameter location. If a turn is required, it will create a turn-down in the Y coordinates.
Cad.GotoXUp(location)	Like GotoX, but it will make the turn up in the Y coordinates if necessary.
Cad.Split()	It breaks the scaffold. The current nucleotide will belong to the next scaffold strand.
Cad.Clean()	It erases all Blender geometry objects.
Cad.RenderRibbons()	Creates the Blender geometry that represents the design as a set of ribbons, including scaffold and staples.
Cad.RenderCylinders()	Creates the Blender geometry that represents the design as a set of cylinders.
Cad.writeCadnano(filename)	Export the design to caDNAno compatible format.