MENDEL: an automated design tool for DNA nanotechnology

Jorge Guerrero
Department of Nanoegineering
Joint School of Nanoscience and Nanoengineering
North Carolina A&T State University, United States
jeguerrero@aggies.ncat.edu

Reza Zadegan [1]
Department of Nanoegineering
Joint School of Nanoscience and Nanoengineering
North Carolina A&T State University, United States
rzadegan@ncat.edu

Abstract

Structural DNA nanotechnology is a promising tool for bottom-up self-assembly. Scientists' imagination and ability to design sophisticated and larger structures inform the applications of the technology. In response to the need for the advanced design strategies, the community has developed a number of software to ease DNA nanostructures design process. Majority of the available software require manual manipulation and detailed visual inspection of the model, which decrease the success of making complex and large structures and increase the user error. To address these concerns, we developed an open-source software coined MENDEL that automates the process of designing nanostructures. MENDEL uses a sequence of commands that accurately and parametrically build the DNA nanostructure's geometry, size, and shape. Additionally, the commands are modular and therefore the construct can grow indefinitely from one repeated layer with a single instruction; hence MENDEL reduces the time, error, and computational cost of DNA nanostructures designing process. Also, when run as a module of Blender, MENDEL generates visual representation of the model. For convenience, MENDEL enables automatic generation of caDNAno and CanDo compatible files. We aim to keep MENDEL open source to allow community collaboration and the software's accessibility to a broader range of scientists.

2012 ACM Subject Classification

Keywords and phrases DNA, Origami, Automated design tool, MENDEL

Digital Object Identifier 10.4230/LIPIcs.DNA26.2020.

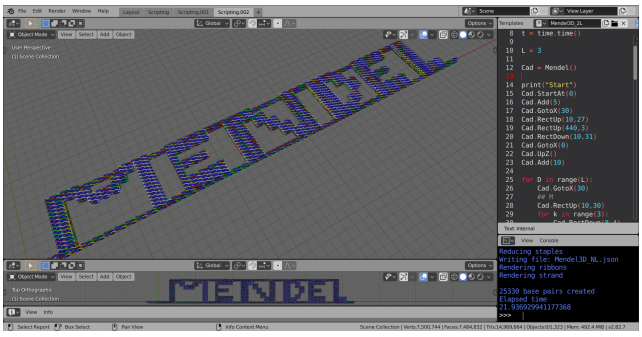

Figure 1 We used the MENDEL library to design a complex triple-layered origami consisting of 25,330 base pairs, in under 22 seconds. Blue and colored ribbons represent the scaffold and staples, respectively.

[1] Corresponding author

26th International Conference on DNA Computing and Molecular Programming.
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Designing complex DNA nanostructures is a complicated process that requires efficient software to calculate and populate structural details. Most of the published software require manual manipulation and careful inspection of the models that increase the time cost and user error and decrease the flexibility of designing process. We created a python library that we coined MENDEL as a flexible and robust solution for automatic design of complex DNA nanostructures. MENDEL receives a set of sequential commands and creates the structures by following logical steps. Each step instructs the growth of the DNA nanostructure either by adding new nucleotides or repeating sections of arbitrary size and shape. User is able to monitor the design progress by executing the commands in Blender software's scripting mode. Figure 1 shows an example of using MENDEL library to design a triple-layered origami that represents the word "MENDEL."

MENDEL generates the geometry preview, which helps to understand the design details. Moreover, for convenience, the exported file is compatible with caDNAno [1]. Figure 2 shows the exported model when opened in caDNAno, and Figure 3 shows the modeling results obtained from CanDo [2] for different number of layers. Future work include improving nucleotide twist and rise calculations, supporting honeycomb designs, detecting overlaps, inserting and skipping nucleotides, and generating molecular file formats such as Protein Data Bank (PDB).

2 Features

The following are the main features of the MENDEL library:

Open-sourced. The code can be expanded by modifying/adding Python scripts

Scripted design. The commands are stored as scripts and can be shared as separate files outside of the Blender environment. It is possible to create functions that call the MENDEL commands and hence parametrically enhance the design and perform with higher flexibility.

Low level primitives. MENDEL is supported by the basic primitives commands that provide single step growing of the structures.

Accurate folding location. MENDEL calculates the 3D coordinates of each nucleotide, and if a strand/layer/shape is added, the software will add enough nucleotides until the last nucleotide is properly aligned for folding according to arbitrary angles.

3 Example of commands

Table 1 shows the set of commands provided with MENDEL and a brief explanation of its use.

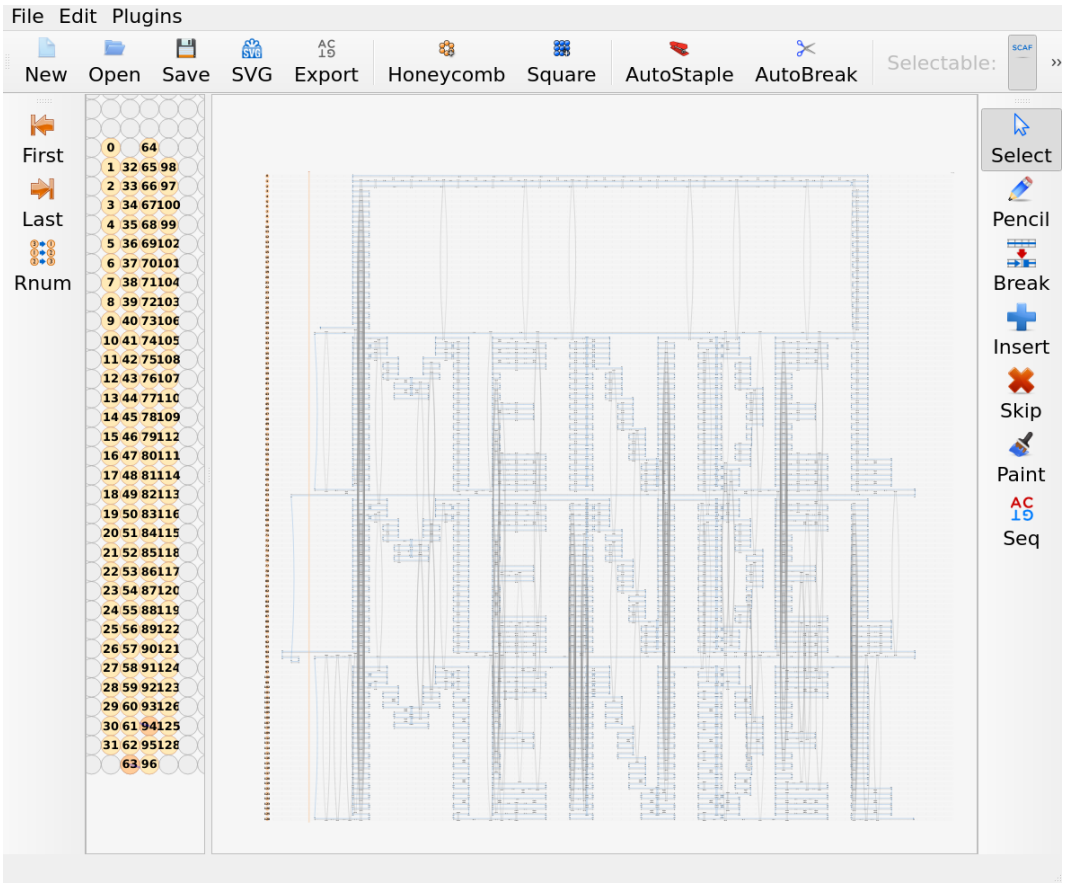

Figure 2 The exported files are compatible and can be opened with caDNAno.

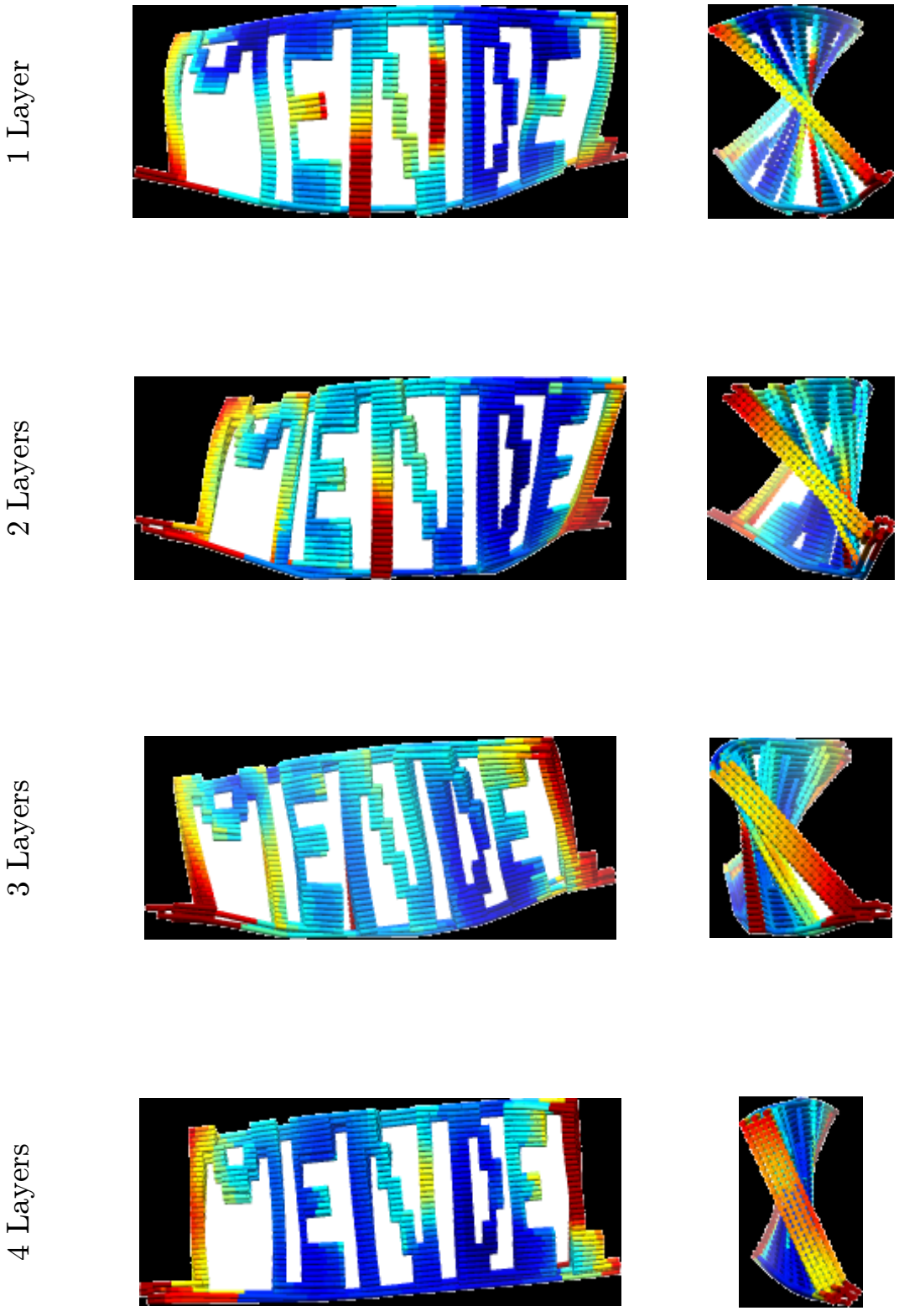

Figure 3 Representation of the modeling prediction of the exported files using CanDo; from one layer to four layers of thickness. The parameter L (number of layers) was the only parameter that we modified in the script to achieve the presented models.

Table 1 MENDEL command set

Command	Description
`Cad = Mendel()`	Create a new Mendel object named Cad. All commands should be applied to the object Cad.
`Cad.StartAt(position)`	Define the starting location along the X coordinates. This 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)`	Add a new nucleotide at a exact location (x,y,z). The orientation is automatically computed according to the YZ coordinates. The new nucleotide will start a new DNA scaffold.
`Cad.Add(number)`	Add a number of nucleotides to the current DNA scaffold. The next nucleotide will be shifted in the X coordinates according to the current direction of the scaffold, which changes each turn.
`Cad.UpY()`	Add a turn in the Y direction. That means, the scaffold will grow along the X direction until it will be aligned such that it can turn up in Y coordinates. The next nucleotides will be added in the opposite X direction
`Cad.DownY()`	Similar to UpY(), but it will turn down in Y coordinates.
`Cad.UpZ()`	Create a turn up in Z coordinates
`Cad.DownZ()`	Create a turn down in Z coordinates
`Cad.RectUp(width, height)`	Create a rectangle in which the width is measured in nucleotides and height 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 make turn down in the Y coordinates.
`Cad.GotoXUp(location)`	Similar to GotoX, but it will make the turn up in the Y coordinates if it is necessary.
`Cad.Split()`	Breaks the scaffold. The current nucleotide will belong to the next scaffold strand.
`Cad.Clean()`	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.
`Cad.RenderPDF(filename)`	Create a 2D representation of the design and export it to PDF.

4 Example code to construct the “MENDEL” origami

Listing 1 provides the example of the executed file in Blender. Initially it constructs a small frame, then one loop encloses the code to build each letter and spaces. At the end, the strand is returned to the origin and starts the next layer. The modifying loop parameter allows for growth of an arbitrary size and dimension.

Listing 1 Script to generate the word "MENDEL" on DNA

```
import bpy
import time
filepath = bpy.path.abspath("//Mendel.py")

exec(compile(open(filepath).read(), filepath, 'exec'))

t = time.time()    #     Start timer
L = 3              #     Number of layers

Cad = Mendel()     #     Cad is the name of the variable
print("Start")     #     we use here.
Cad.StartAt(0)
Cad.Add(5)         #     Definition of the initial Frame:
Cad.GotoX(30)      #     Starting location
Cad.RectUp(10,27)  #     Left border of the frame
Cad.RectUp(440,3)  #     Upper border
Cad.RectDown(10,31) #    Right border
Cad.GotoX(0)         #   Return to starting location
Cad.UpZ()            #   Grow a new layer
Cad.Add(10)          #   Add 10 bases
                     #    Begin of the loop:  range(L) means it will
for D in range(L):    #  create L layers for the word MENDEL
    Cad.GotoX(30)     #  Starting location of M character
    ## M
    Cad.RectUp(10,30)      #     Left arm
    for k in range(3):     #
        Cad.RectDown(8,4)  #     Diagonal down
    for k in range(3):
        Cad.RectUp(8,4)    #     Diagonal up
    Cad.RectDown(10,30)    #     Right arm

    Cad.Add(20)            #     Separation of characters
    ## E ##                #     Definition of character E
    Cad.RectUp(40,7)       #     Lower horizontal arm
    Cad.RectUp(8,6)        #     Grow up to the center
    Cad.RectUp(30,4)       #     Central horizontal arm
    Cad.RectUp(8,6)        #     Grow up to the upper
    Cad.RectUp(40,7)       #     Upper horzitonal arm

    Cad.Add(20)            #     Separation. Location is upper

    ## N ##                 #    Definition of character N
    Cad.RectDown(10,30)     #    Location is up, left arm goes down
    Cad.RectUp(5,30)        #    and up to start diagonal

    for k in range(5):
```

```
        Cad.RectDown(8,6)   #    Diagonal down

    Cad.RectUp(10,30)       #    Right arm

    Cad.Add(20)             #    Separation. Location is up

    ## D  ##                #    Definition of character D
    Cad.RectDown(10,30)     #    Left arm goes down
    Cad.RectUp(5,30)        #    and up
    Cad.RectDown(10,4)      #    Upper horziontal line
    for k in range(2):
        Cad.RectDown(5,6)   #    Diagonal down

    Cad.DownY()             #    Add new row. Switch direction
    Cad.Add(10)             #    Diagonal down
    for k in range(3):      #
        Cad.Add(5)          #    moving left
        Cad.RectDown(5,4)   #
    Cad.DownY()             #    Fill last block
    Cad.Add(50)             #    Align with next letter

    ## E ##                 #    Defintion of second character E
    Cad.RectUp(40,7)        #
    Cad.RectUp(8,6)         #    Exact same code of the
    Cad.RectUp(30,4)        #    first  E
    Cad.RectUp(8,6)         #
    Cad.RectUp(40,7)        #

    Cad.Add(20)             #

    ## L ##                 #    Definition of character L
    Cad.RectDown(10,23)     #    Left arm
    Cad.RectDown(30,7)      #    Lower horizontal arm

                            # Link to the next layer
    Cad.Add(20)             #    Adds 20 nucleotides as a small bar

    if D != L-1 :           #
        Cad.DownY()         #    Switch direction
        Cad.GotoX(0)        #    Return to initial location
                            #
        if D%2 == 0:        #    Adjustement for some layers
            Cad.Add(20)     #
            Cad.UpZ()       #    if is an even layer, add another
            Cad.RectUp(5,2) #    row in Y
        else:               #
            Cad.UpZ()       #    Otherwise, just add the new layer
            Cad.Add(10)     #

Cad.writeCaDNAno("Mendel3D_NL.json") # Export file
Cad.Clean()                 #    Clean Blender existing geometries
Cad.RenderRibbons()         #    Create ribbon representations
elapsed = time.time() - t   #    Calculate execution time
print("")                   #
print(str(Cad.GetNumber()) + " base pairs created")
```

```
print("Elapsed time")
print(elapsed)                #   Print elapsed time
```

References

1 Shawn M. Douglas, Adam H. Marblestone, Surat Teerapittayanon, Alejandro Vazquez, George M. Church, and William M. Shih. Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucleic Acids Research*, 37(15):5001–5006, 2009. doi:10.1093/nar/gkp436.

2 Do Nyun Kim, Fabian Kilchherr, Hendrik Dietz, and Mark Bathe. Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures. *Nucleic Acids Research*, 40(7):2862–2868, 2012. doi:10.1093/nar/gkr1173.